UNIVERSITETET I OSLO

RAPPORT 5

Å arbeide algoritmisk i matematikk

Algoritmisk tenkning og programmering på 8. trinn, 10. trinn og Vg1

EDUCATE

EVALUERING AV FAGFORNYELSEN I FAG

Roar B. Stovner, Ove E. Hatlevik, Peter N. Aashamar, Lisbeth M. Brevik, Greta B. Gudmundsdottir og Gunnar Lid

Å arbeide algoritmisk i matematikk

Algoritmisk tenkning og programmering på 8. trinn, 10. trinn og Vg1

Roar B. Stovner, Ove E. Hatlevik, Peter N. Aashamar,
Lisbeth M. Brevik, Greta B. Gudmundsdottir og Gunnar Lid

Rapport 5 fra forsknings- og evalueringsprosjektet EDUCATE
ved Institutt for lærerutdanning og skoleforskning,
Universitetet i Oslo

PROSJEKT: EDUCATE – EVALUERING AV FAGFORNYELSEN I FAG
(LIVSMESTRING, UTFORSKING, DIGITAL KOMPETANSE)
RAPPORT NR. 5
UTGIVELSESÅR: 2025
UTGIVER: INSTITUTT FOR LÆRERUTDANNING OG SKOLEFORSKNING,
DET UTDANNINGSVITENSKAPELIGE FAKULTET, UNIVERSITETET I OSLO
ADRESSE: POSTBOKS 1099, BLINDERN, 0317 OSLO
NETTSTED: HTTPS://WWW.UV.UIO.NO/ILS/FORSKNING/PROSJEKTER/EDUCATE/INDEX.HTML
LAYOUT: SHANE COLVIN, UNIVERSITETET I OSLO
FOTO OMSLAG: SHANE COLVIN, UNIVERSITETET I OSLO

ISBN 978-82-94119-02-8 (trykt)
ISBN 978-82-94119-03-5 (digital)

OPPDRAGSGIVER: UTDANNINGSDIREKTORATET
OM OPPDRAGET: RAPPORTEN ER EN DEL AV UTDANNINGSDIREKTORATETS EVALUERINGSPROGRAM FOR FAGFORNYELSEN. EVALUERINGEN PÅGÅR I PERIODEN 2021 TIL 2025.

PROSJEKTLEDELSE:
Lisbeth M. Brevik, professor i engelskdidaktikk (prosjektleder)
Greta Björk Gudmundsdottir, professor i pedagogikk (prosjektnestleder)
Nora E. H. Mathé, førsteamanuensis i samfunnsfagdidaktikk (prosjektnestleder)

PROSJEKTMEDLEMMER:
Peter N. Aashamar, forsker, samfunnsfagdidaktikk
Rebecca L. S. Barreng, forsker/høyskolelektor, engelskdidaktikk
Katherina Dodou, postdoktor, engelskdidaktikk
Gerard Doetjes, leder, Nasjonalt senter for engelsk og fremmedspråk i opplæringen
Ingrid Evertsen, offentlig ph.d., engelskdidaktikk
Kirsten M. Hartvigsen, førsteamanuensis, religions- og etikkdidaktikk
Ove E. Hatlevik, professor, pedagogikk
Anja R. Isaksen, offentlig ph.d., engelskdidaktikk
Inga Staal Jenset, førsteamanuensis, pedagogikk
Camilla G. Magnusson, postdoktor, norskdidaktikk
Astrid Roe, forsker, kvantitative analyser
Henriette Siljan, førsteamanuensis, norskdidaktikk
Kaja G. Skarpaas, førsteamanuensis, engelskdidaktikk
Roar Bakken Stovner, førsteamanuensis, matematikkdidaktikk
Mai Lill Suhr, førsteamanuensis, naturfagdidaktikk

TEACHING LEARNING VIDEO LAB:
Gunnar Lid, overingeniør

EKSTERN LESER:
Øyvind Aas, førsteamanuensis, Høyskolen Kristiania

REFERERES TIL SOM: STOVNER, R. B., HATLEVIK, O. E., AASHAMAR, P. N., BREVIK, L. M., GUDMUNDSDOTTIR, G. B. OG LID, G. (2025). Å ARBEIDE ALGORITMISK I MATEMATIKK. ALGORITMISK TENKNING OG PROGRAMMERING PÅ 8. TRINN, 10. TRINN OG VG1. RAPPORT 5 FRA FORSKNINGS- OG EVALUERINGSPROSJEKTET EDUCATE VED INSTITUTT FOR LÆRERUTDANNING OG SKOLEFORSKNING, UNIVERSITETET I OSLO.

Forord

Institutt for lærerutdanning og skoleforskning (ILS) ved Universitetet i Oslo ble våren 2021 tildelt midler av Utdanningsdirektoratet (UDIR) for å gjennomføre en forskningsbasert evaluering av fagfornyelsen i fagene. Prosjektets tittel er EDUCATE – Evaluering av fagfornyelsen i fag (livsmestring, utforsking og digital kompetanse). Prosjektet gjennomføres i perioden 2021–2025. Over 20 forskere og ingeniører ved ILS og samarbeidende institusjoner er involvert i evalueringen og bidrar med ulik faglig kompetanse.

Dette er Rapport 5 fra EDUCATE. Denne rapporten kommer i tillegg til EDUCATEs Rapport 4 som bidrar til økt kunnskap om i hvilken grad og på hvilken måte digital kompetanse gjennomføres i fagene. Vi skilte ut algoritmisk tenkning og programmering i matematikkfaget på grunn av stor relevans for praksisfeltet.

Vi presenterer her resultater fra hele prosjektets datainnsamling i matematikk, gjennom både kvalitative og kvantitative analyser. Rapporten beskriver kjennetegn ved klasseromspraksiser når det arbeides med algoritmisk tenkning og programmering i matematikkfagene på 8. trinn, 10. trinn og Vg1 (1T, 1P og 1P-Y). Matematikk er et obligatorisk fag på ungdomstrinnet og alle elever må velge mellom de tre matematikkfagene på Vg1. Disse trinnene er relevante med tanke på at de representerer overgangen fra barnetrinnet til ungdomstrinnet, fra ungdomstrinnet til videregående skole, og første år på både studiespesialiserende og yrkesfaglige studieprogram.

Dataene ble samlet inn gjennom to skoleår: 2021–22 (8. trinn og Vg1) og 2023–24 (10. trinn). På vg2 og vg3 samlet vi ikke inn data fra matematikkfagene (for eksempel S1, S2, R1 eller R2), ettersom dette ikke er obligatoriske fellesfag. Vi ønsker å takke alle skoler, lærere og elever som har deltatt i EDUCATE-prosjektet og som har gitt oss muligheten for å følge implementeringen av LK20 over tid.

Dette er femte og siste delrapport fra prosjektgruppen før sluttrapporten leveres ved prosjektets avslutning høsten 2025.

God lesing!

Oslo, 10. februar 2025
Lisbeth M. Brevik og Greta Björk Gudmundsdottir
Ledere av EDUCATE-prosjektet

Innhold

Bokser

Figurer

Tabeller

Sammendrag

Formålet med denne rapporten har vært å svare på UDIRs oppdrag om å bidra til økt kunnskap om implementeringen av algoritmisk tenkning og programmering på 8. trinn, 10. trinn og Vg1, og gi indikasjoner på hva som kjennetegner slike klasseromspraksiser i de obligatoriske matematikkfagene, både innenfor og på tvers av trinn.

Dette gjøres i tre trinn. Først, for skoleåret 2021–22 har vi sammenlignet undervisningen på 8. trinn og Vg1. Dette er det første skoleåret i LK20 for elevene på disse trinnene. Deretter har vi også for skoleåret 2021–22 sammenlignet undervisningen på tvers av obligatoriske matematikkfag på Vg1: 1T og 1P på studieforberedende program, og 1P-Y på yrkesfaglige studieprogram. Til slutt har vi vurdert endringer i undervisningen over tid for de samme klassene som vi har fulgt fra 8. trinn (skoleåret 2021–22) til 10. trinn (skoleåret 2023–24). Programmering er også del av læreplanene i naturfag, kunst og håndverk, Duodji og musikk, men vi har sett på matematikkfaget fordi det er bare i matematikklæreplanen at algoritmisk tenkning er nevnt.

Som referanseramme for våre analyser har vi laget *EDUCATEs protokoll for observasjon av algoritmisk tenkning og programmering i undervisningen* (versjon 1.0). Denne protokollen presenteres i sin helhet i Del 2 og bygger på teoretiske og empiriske definisjoner av algoritmisk tenkning og programmering i litteraturen, tidligere rammeverk for algoritmisk tenkning og definisjoner i læreplanverket LK20.

EDUCATEs tre hovedfunn om algoritmisk tenkning og programmering

EDUCATE-prosjektet benytter et komplekst forskningsdesign som tar i bruk flere datakilder og metoder for å få fram ulike perspektiver på algoritmisk tenkning og programmering i de matematikkfagene vi har undersøkt. Til denne rapporten har vi brukt data fra 8. trinn, 10. trinn og Vg1 i 29 klasserom, blant 17 matematikklærere ved fire ungdomsskoler og tre videregående skoler. To av lærerne fulgte sine klasser fra 8. til 10. trinn, og vi trekker frem endringer i deres syn på og erfaring med å undervise i algoritmisk tenkning og programmering på tvers av de to skoleårene, som også innebærer å se på progresjon i implementeringen av tematikken.

❖ Hovedfunn 1: Algoritmisk tenkning forekommer i matematikkundervisningen, men lærerne synes konseptet er uklart og definerer det på ulike måter

Den videofilmede undervisningen i de obligatoriske matematikkfagene på 8. trinn, 10. trinn og Vg1 viser at algoritmisk tenkning forekom i halvparten av timene. Det vi omtaler som algoritmisk tenkning er delt inn i algoritmisk problemløsning og algoritmiske arbeidsmåter.

I matematikkundervisningen besto den *algoritmiske problemløsningen* oftest av figurtallundervisning og problemløsningsoppgaver, og i noen klasserom utviklet eller forklarte elevene en algoritme eller regneregel. Dette arbeidet var hovedsakelig preget av dekomponering og resonnering, og i svært liten grad av at elever faktisk lagde en algoritme som løste et problem. De *algoritmiske arbeidsmåtene* besto oftest av fikling, altså leken utprøving for å komme videre med en oppgave, og skaping, gjerne mens elevene samarbeidet. Derimot jobbet de lite med feilsøking, noe som kan være fordi feilsøking ofte gjøres individuelt og kan derfor være utfordrende å fange på lyd og film.

Lærerne planla å jobbe med algoritmisk tenkning i en tredel av timene. Samtidig var det lite samsvar mellom det lærerne loggførte etter undervisningen, og det vi observerte i de videofilmede timene. Lærerne uttrykte en usikkerhet når det gjaldt forståelsen av *begrepet* algoritmisk tenkning. Fire av de 17 lærerne vi intervjuet forbant algoritmisk tenkning med en problemløsningsmetode, i likhet med det som står i LK20. Samtidig presenterte 12 lærere andre definisjoner og én lærer sa han ikke visste hva begrepet innebar. Ingen lærere nevnte de algoritmiske arbeidsmåtene som en del av algoritmisk tenkning. Her bør skoleeier og skoleledelsen ta ansvar for å utarbeide felles forståelse for algoritmisk tenkning i tråd med læreplanen. Se forslag i EDUCATEs algoritmiske agenda til slutt i dette sammendraget og mer utdypet i Del 6.

Våre funn tyder derfor på at innføringen av begrepet al*goritmisk tenkning* i LK20 har ført til begrensede praksisendringer, og at de praksisendringene vi ser skyldes at programmering er kommet inn i læreplanen snarere enn at algoritmisk tenkning er det.

❖ Hovedfunn 2: Programmering forekom sjelden i undervisningen og lærerne sa de manglet programmeringskompetanse, men de hadde likevel positive holdninger til programmering

Lærerne underviste i programmering i 3% av undervisningssegmentene, eller i 7% av undervisningssegmentene hvis vi regner med bruk av vilkår i regneark og graftegner. Programmeringsundervisningen var hovedsakelig at elevene lærte seg programmering ved å følge digitale veiledere (*tutorials*) mens læreren hjalp elevene, og kun sjeldent løste elevene programmeringsoppgaver eller benyttet programmering som verktøy. Elevene benyttet noen ganger programmeringsliknende syntaks (spesielt vilkår) da de arbeidet i regneark og graftegner. Av de fire konseptene elevene skal kunne om programmering etter 6. trinn (*variable*, *vilkår*, *løkker* og *funksjoner*) identifiserte vi kun arbeid med variable og vilkår i undervisningen.

I intervjuene fortalte lærerne at de programmerte mindre enn de hadde planlagt, og de fleste sa at de underviste programmering som et eget tema, løsrevet fra matematikken. Dette underbygges av våre observasjoner av timene vi filmet. Det er viktig å påpeke at andelen programmering i de filmede matematikktimene trolig underestimerer andelen programmering i disse klassene. Flere lærere nevnte i intervjuene at de ofte underviste om programmering som et eget tema rett før sommerferien. EDUCATE-teamet filmet undervisning andre tider på året for ikke å forstyrre skolene i eksamensforberedelser. Vi kan derfor ha gått glipp av programmeringsundervisningen på tampen av året som lærerne fortalte om i intervjuene.

Lærerne var positive til at programmering hadde kommet inn i skolen, både fordi de anså det som en viktig kompetanse i arbeidslivet og et tema som kunne bidra til elevenes motivasjon i faget. Derimot var de kritiske til egen programmeringskompetanse. De fleste lærerne mente de ikke kunne å programmere godt nok, mens noen sa de hadde akkurat nok kompetanse til å *prøve* programmering i undervisningen. Med tanke på lærernes eksplisitte uttrykk for manglende programmeringskompetanse, slik det fremkommer både i denne og andre studier, fremstår det som viktig å tilby kompetanseheving i programmering for lærere. Se forslag i EDUCATEs digitale agenda til slutt i dette sammendraget og utdypet i Del 6.

❖ Hovedfunn 3: Uklar progresjon i algoritmisk tenkning og programmering mellom fag og trinn

Det tredje hovedfunnet i denne rapporten peker på en uklar progresjon i algoritmisk tenkning og programmering mellom trinn og mellom ulike matematikkfag, noe som er bekymringsfullt. Verken den videofilmede undervisningen eller lærerintervjuene viste klare forskjeller mellom matematikkfag og trinn. For eksempel ble det undervist om figurtall på alle trinn og fag, og til dels samme oppgaver på 8. trinn som i matematikk 1T.

Vi så også de samme programmeringskonseptene bli undervist til samme klasse på både 8. og 10. trinn. Også i intervjuene med de to lærerne som fulgte klassene sine fra 8. til 10. trinn, ga lærerne relativt like svar om algoritmisk tenkning og programmering på tvers av trinnene, selv om de rapporterte at de hadde undervist noe mer programmering skoleåret 2023–24 enn de gjorde i 2021–22.

Den manglende progresjonen kan være forårsaket av at våre data er fra de første årene etter innføringen av LK20, og at elevene derfor ikke kan det LK20 forventer fra tidligere trinn. Da er det naturlig at vi, for eksempel, ser den samme programmeringsundervisningen på 8. trinn som på Vg1. Likevel, i våre data er det en alvorlig mangel på progresjon mellom trinnene.

EDUCATEs oppsummering av algoritmisk tenkning og programmering i matematikkfagene

Vi presenterer først hovedtrekk når det gjelder undervisning om algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering. Beskrivelsene av algoritmisk tenkning (skår 1–4) viser til EDUCATEs observasjonsprotokoll (se Del 2).

Algoritmisk problemløsning i matematikk

- Vi fant at elevene jobbet med algoritmisk problemløsning i rundt halvparten av den filmede undervisningen (52 %). I litt over en tredel (36%) av segmentene observerte vi at elevene jobbet med en kjent algoritme for å løse en gitt type oppgave (skår 2). I 15% av segmentene jobbet de med enkel algoritmisk problemløsning (skår 3), og i under 1 % av segmentene jobbet de med avansert algoritmisk tenkning (skår 4).
- Når elevene brukte algoritmisk problemløsning på en måte som ble skåret 3 eller 4, fant vi følgende temaer og fordeling: figurtallundervisning (43 %), diverse problemløsning (38 %), og det å utvikle en algoritme eller regel (19 %).
- Noen av oppgavene om algoritmisk problemløsning som elevene jobbet med i den filmede undervisningen presenteres i Del 4: *Figurtallsoppgaver* (Figur 4.5), *Kjeksoppgaven* (Boks 7), *Smileyoppgaven* (Boks 8) og *Lyspæreoppgaven* (Boks 9).

Algoritmiske arbeidsmåter i matematikk

- Vi fant at elevene jobbet med algoritmiske arbeidsmåter i 40 % av undervisningen. Elevene jobbet ofte med fikling, altså leken utprøving for å komme videre med en oppgave (skår 2; 16 %), eller med både fikling og skaping eller feilsøking i 20% av segmentene (skår 3), men de benyttet sjelden de algoritmiske arbeidsmåtene i utstrakt grad (skår 4; 4 %).
- Når elevene brukte algoritmiske arbeidsmåter på en måte som ble skåret 3 eller 4, fant vi følgende fordeling: fikle (65 %), skape (30 %) og feilsøke (5 %).
- Noen av oppgavene om algoritmiske arbeidsmåter som elevene jobbet med i den filmede undervisningen presenteres i Del 4: *Surpriseparty* (Boks 11) og *Finn feilene i koden* (Boks 12).

Programmering i matematikk

- Vi fant at elevene jobbet med programmering kun i en liten del av undervisningen (7 %). Vi observerte i 6 % av segmentene at elevene skulle lære seg syntaksen til et programmeringsspråk (skår 2), og svært sjeldent at de løste programmeringsoppgaver (skår 3–4; 1 %).
- Når elevene arbeidet med programmering, fant vi følgende fordeling: digital veileder (35 %), regneark (31 %), graftegner (19 %), programmeringsoppgaver og programmering som verktøy (15 %).
- Noen av oppgavene om programmering som elevene jobbet med i den filmede undervisningen presenteres i Del 4: *Funksjoner i Excel* (Figur 4.12), *Programmeringskonsepter* (Boks 13), *Hvis-setning* (Boks 14), *Antall sekunder i et døgn* (Boks 15), *Omregning* (Boks 16) og *Lyspæreoppgaven ved bruk av programmering* (Boks 17).

Boks 1 EDUCATEs algoritmiske agenda rettet mot ungdomstrinnet og videregående skole

Lærere

- Lærerne bør vurdere å benytte programmering når de løser figurtallsoppgaver.

Skoleledelse

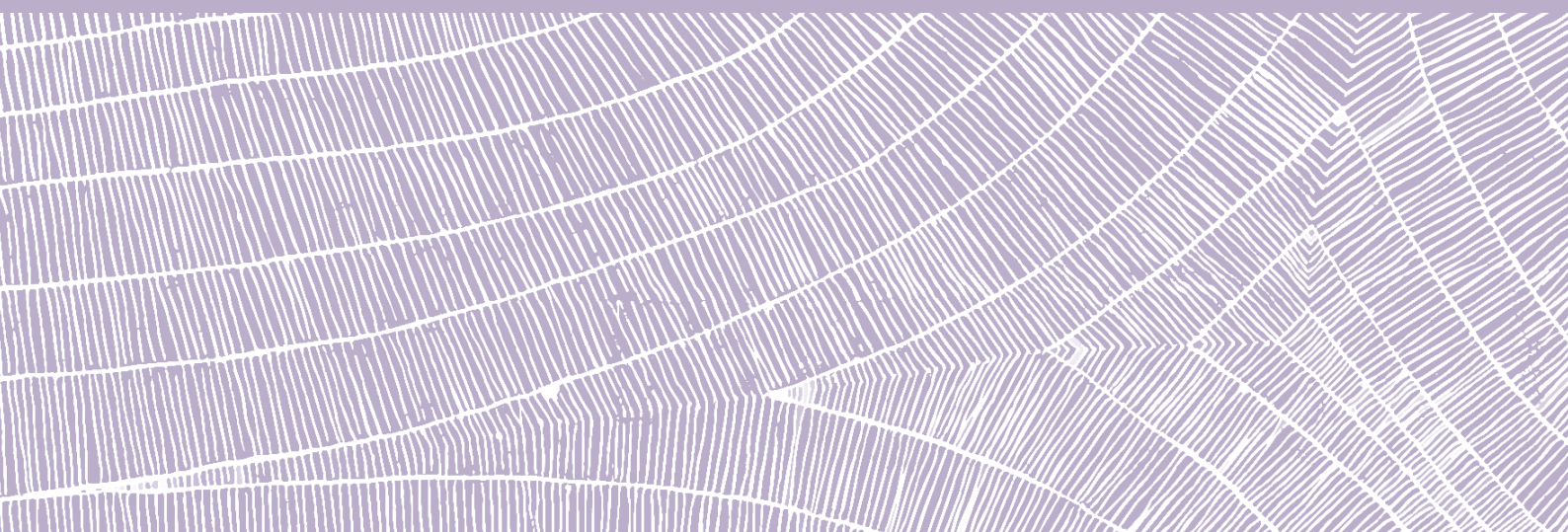
- EDUCATE oppmuntrer skoleledelsen til å ta ansvar for å kartlegge og utarbeide felles forståelse blant matematikklærerne for algoritmisk tenkning i tråd med læreplanen.
- EDUCATE ser et tydelig behov for at matematikklærere får mulighet til å utvikle programmeringskompetansen sin gjennom kurstilbud og kollegasamarbeid i arbeidstiden.

Myndigheter og skoleeiere

- EDUCATE ser behov for at myndigheter kartlegger programmeringsundervisning ved lærerutdanningene og går i dialog med lærerutdanningene om kompetanseheving i programmering og programmeringsdidaktikk blant lærerstudenter.
- EDUCATE ser sterkt behov for at myndigheter og skoleeiere utarbeider undersøkelser og kartlegger læreres programmeringskompetanse.
- EDUCATE oppfordrer myndighetene til å øke satsingen på kompetanseheving i programmering og programmeringsdidaktikk for matematikklærere.
- EDUCATE anbefaler myndighetene å klargjøre hva som forventes av programmeringsundervisningen i læreplanverket og veiledninger.
- EDUCATE oppfordrer myndighetene til å følge nøye med på elevenes programmeringskompetanse.

DEL 1

Økt kunnskap om algoritmisk tenkning og programmering i klasserommet



1 Økt kunnskap om algoritmisk tenkning og programmering i klasserommet

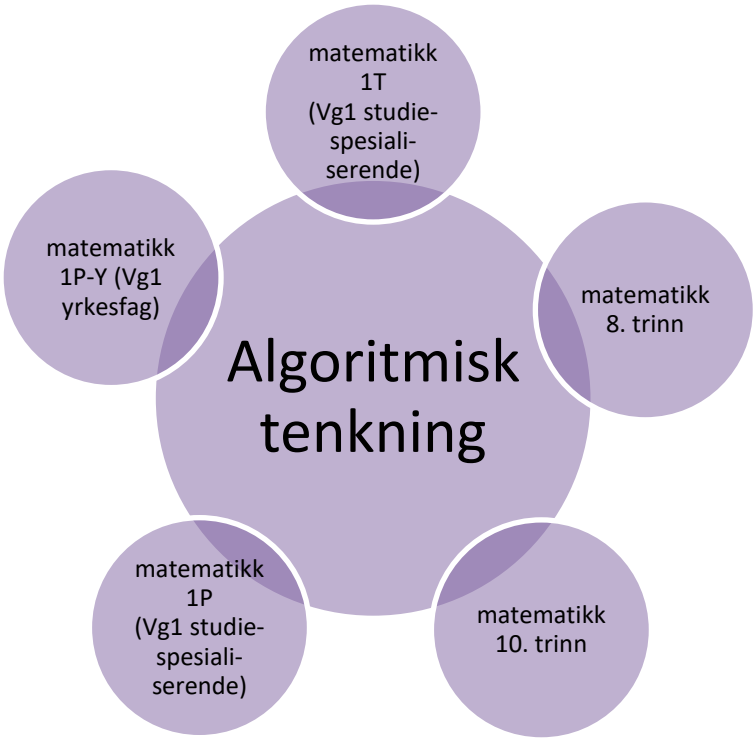
I Del 1 presenterer vi målsettingen med evalueringen som EDUCATE presenterer i denne rapporten. Vi gjør først rede for oppdraget om å bidra til økt kunnskap om algoritmisk tekning i matematikklasserommet på 8. trinn, 10. trinn og Vg1 (1.1). Deretter oppfordrer vi til åpenhet og involvering i evalueringsarbeidet (1.2), før vi presenterer rapportens oppbygging (1.3).

1.1 Oppdraget

EDUCATE-prosjektet gjennomfører en forskningsbasert evaluering av LK20 i utvalgte fag i grunnskolen og videregående skole. Målet er å bidra med økt kunnskap om hvordan lærere tar i bruk det nye læreplanverket i fagene og i klasserommet. Evalueringen er longitudinell, vi følger klasser over tid i LK20 (2021–25) og sammenligner med data vi har samlet inn under Kunnskapsløftet LK06 (2014–20). Rapport 1 gir utdypende informasjon om oppdraget i sin helhet (Brevik m.fl., 2023a). Rapport 2 presenterer funn om innføring av livsmestring på 8. trinn (Brevik m.fl., 2023b). Rapport 3 presenterer funn om utforsking på Vg1 og vg2 (Brevik m.fl., 2024). Rapport 4 presenterer funn om digital kompetanse på 10. trinn og vg3 (Gudmundsdottir m.fl., 2024).

Formålet med Rapport 5 er å svare på UDIRs oppdrag om å bidra til økt kunnskap om hvordan algoritmisk tenkning og programmering gjennomføres i matematikkfagene. Vi gjør det ved å presentere funn fra 8. trinn, 10. trinn og Vg1.

Figur 1.1 EDUCATEs evaluering av algoritmisk tenkning i matematikk



Ved å trekke inn matematikkfagene på disse tre trinnene får vi en bredde som representerer (1) fellesfag som undervises på ungdomstrinnet og i videregående skole og som (2) samtidig undervises i lærerutdanningen ved Institutt for lærerutdanning og skoleforskning (ILS). Videre har medlemmer i EDUCATE prosjektgruppen fagdidaktisk ekspertise i disse fagene. Rapport 5 undersøker undervisning i 29 klasserom ved fire grunnskoler og tre videregående skoler i ulike geografiske områder av landet skoleårene 2021–22 og 2023–24 for å se hvordan det jobbes med algoritmisk tenkning på 8. trinn, 10. trinn og Vg1 og både på studiespesialiserende og yrkesfaglige studieretninger. Målsettingen med EDUCATEs evaluering i denne rapporten er å

- bidra til kunnskap om hvordan realiseringen av algoritmisk tenkning i LK20 eventuelt endrer klasseromspraksis i matematikkfagene
- bidra til ny og etterspurt kunnskap om hvordan lærere gjennomfører klasseromspraksiser knyttet til algoritmisk tenkning i naturlig forekommende undervisning
- undersøke erfaringer med slike praksiser blant lærere på ulike trinn og i ulike matematikkfag
- foreslå implikasjoner for fremtidig klasseromspraksis og lærerutdanning

1.2 Åpenhet og involvering

Forskerteamet i EDUCATE legger vekt på å være transparent og åpen for innspill fra ulike fagfelt som kan bidra til å videreutvikle metoder for å forstå klasseromspraksis. Det er viktig at det omfattende arbeidet som er gjort i utviklingen og kvalitetssikringen av EDUCATEs protokoll for observasjon av algoritmisk tenkning i undervisningen blir gjort tilgjengelig for andre. Protokollen er derfor publisert under CC-BY-NC-SA lisens, som innebærer at den kan deles og brukes, men ikke kommersielt, og at enhver bearbeidelse av protokollen skal være delt med samme lisens. I Rapport 5 publiseres EDUCATEs protokoll for algoritmisk tenkning (versjon 1.0). Den publiseres også i en egen publikasjon samt som nedlastbar PDF-fil på vår nettside[1] og i forskningsarkivet Zenodo.org. All bruk av protokollen skal henvise til EDUCATE. Vi inviterer alle som er interessert i klasseromsforskning til å ta den i bruk i egen forskning, undervisning og utviklingsarbeid. Vi ønsker tilbakemeldinger på hvordan den fungerer og hva som evt. ikke fungerer, slik at vi kan forbedre den innen prosjektslutt i 2025. Vi har opprettet en egen epost for tilbakemeldinger: EDUCATE-prosjektet@ils.uio.no.

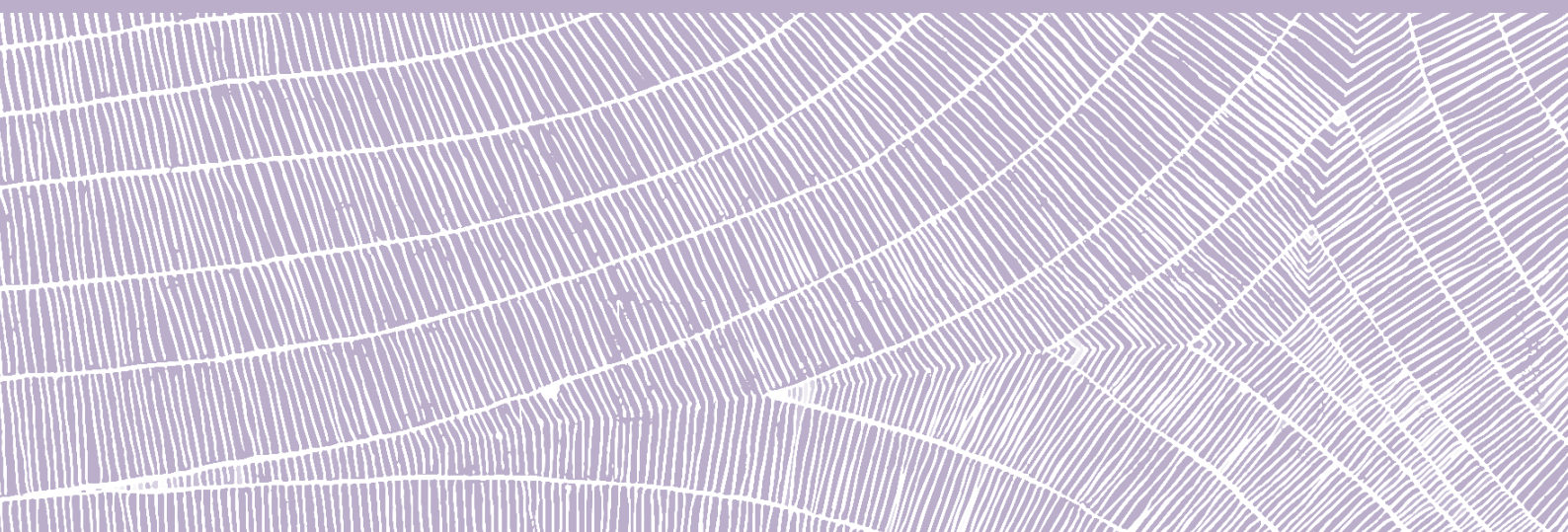
1.3 Rapportens oppbygging

I Del 1 gjør vi kort rede for EDUCATEs oppdrag om å evaluere LK20 i fagene, med mål om økt kunnskap om algoritmisk tenkning og ønsket praksisendring i klasserommet. I Del 2 presenterer vi en oversikt over hvordan algoritmisk tenkning beskrives i litteraturen og i læreplanverket. Dette knyttes til EDUCATEs protokoll for algoritmisk tenkning, som presenteres i slutten av Del 2. I Del 3 presenteres forskningsdesignet som er utviklet for å forstå hvordan det jobbes med algoritmisk tenkning i fagene. I Del 4 og 5 presenteres resultater fra hvert fag basert på empiriske funn. I Del 6 diskuterer vi funnene og implikasjoner de kan ha.

[1] https://www.uv.uio.no/ils/forskning/prosjekter/educate/index.html

DEL 2

Algoritmisk tenkning og programmering i litteraturen og læreplanverket



2 Algoritmisk tenkning og programmering i litteraturen og læreplanverket

Del 2 gir en beskrivelse av *algoritmisk tenkning* slik det presenteres i litteraturen og i læreplanverket og kobler disse sammen. Først presenterer vi en begrepsavklaring av algoritmisk tenkning (2.1) og trekker opp skillet mellom algoritmisk tenkning og programmering. Deretter presenterer vi algoritmisk tenkning i forskningslitteraturen (2.2) og trekker inn programmering i læreplanen og til eksamen (2.3). Videre går vi inn på algoritmisk tenkning og programmering i matematikkfaget (2.4). Til slutt kobler vi algoritmisk tenkning i litteraturen til læreplanverkets beskrivelser av algoritmisk tenkning i matematikkfagene som EDUCATE evaluerer i denne rapporten (2.5). Basert på denne koblingen, forklarer vi hvordan vi har operasjonalisert algoritmisk tenkning i EDUCATEs protokoll for observasjon av algoritmisk tenkning i klasserommet (2.6). Til slutt oppsummeres denne delen (2.7).

2.1 Algoritmisk tenkning i læreplanverket

Algoritmer har alltid vært en del av matematikken. For fire tusen år siden ble en algoritme for løsning av andregradslikninger risset inn i steintavler fordi det var viktig kunnskap for rettferdig fordeling av landområder. For femti år siden ble en algoritme for å regne ut en tilnærming til den deriverte, tilbakepropagering, skrevet i en forskningsartikkel fordi det var viktig kunnskap for å trene nevrale nettverk til kunstig intelligens. Algoritmer har også spilt en vesentlig rolle i skolematematikken, og mye av skolematematikken har dreid seg om å lære å utføre nyttige algoritmer som man trenger til hverdag og jobb. Begrepet *algoritmisk tenkning*, derimot, er en nyvinning som ikke handler om å lære seg å utføre nyttige algoritmer, som har sitt historiske opphav i informatikk, ikke matematikk, og som har kommet inn i læreplaner flere steder i verden det siste tiåret (Bocconi m.fl., 2022). Vi presenterer her læreplanverkets definisjon av algoritmisk tenkning og viser hvordan begrepet er knyttet opp til andre områder i matematikklæreplanen og overordnet del i læreplanverket.

Algoritmisk tenkning i læreplanen

Begrepet *algoritmisk tenkning* benyttes i læreplanen som en oversettelse av begrepet *computational thinking* (UDIR, 2024). Læreplanen forklarer begrepet *algoritmisk tenkning* kun én gang, under kjerneelementet *Utforsking og problemløysing* (Pachel m.fl., 2024). Der forklares det at

> *Algoritmisk tenking er viktig i prosessen med å utvikle strategiar og framgangsmåtar for å løyse problem og inneber å bryte ned eit problem i delproblem som kan løysast systematisk. Vidare inneber det å vurdere om delproblema best kan løysast med eller utan digitale verktøy. (Kunnskapsdepartementet, 2020a, s. 2)*

I LK20 beskrives altså algoritmisk tenkning som en problemløsningsmetode som innebærer *dekomponering*, å bryte ned et problem i delproblemer som kan løses systematisk, for deretter å vurdere om delproblemene best løses med eller uten digitale verktøy.

I tillegg har UDIR (2024) utarbeidet en veileder på nettsidene sine som utbroderer begrepet. I veilederen presiserer UDIR at algoritmisk tenkning er den norske oversettelsen av *computational thinking*, og at dette er en problemløsningsmetode der man «tenker som en informatiker» (UDIR, 2024, første avsnitt). Hva dette innebærer, slås fast i et avsnitt som ganske eksplisitt knytter algoritmisk tenkning til tre forskjellige kjerneelementer i matematikkfaget. Sitatet i Figur 2.1 er fra UDIR (2024, andre avsnitt) og markeringen av kjerneelementene er vår.

Figur 2.1 Overlapp mellom kjerneelementene i matematikk og algoritmisk tenkning

Problemløsning

Modellering og anvendelser

Abstraksjon og generalisering

Algoritmisk tenkning innebærer å bryte ned komplekse problem til mindre, mer håndterlige delproblemer som lar seg løse. Det inkluderer å organisere og analysere informasjon på en logisk måte og å lage fremgangsmåter (algoritmer) for å komme fram til ønsket løsning. Det handler også om å lage abstraksjoner og modeller av den virkelige verden ved å fjerne unødvendige detaljer og fokusere på det som er relevant for den aktuelle problemstilling og løsning. En løsning på et spesifikt problem kan ofte generaliseres, slik at den kan brukes til å løse lignende problemer, og løsninger på flere delproblemer kan kombineres for å løse mer komplekse problem.

Veilederen viser videre noen «viktige nøkkelbegrep som inngår i algoritmisk tenkning» og «typiske arbeidsmåter den algoritmiske tenkeren bruker» (UDIR, 2024), se Figur 2.2. Begrepene til venstre i Figur 2.2, som logikk, mønstre og abstraksjon finner vi igjen i kjerneelementene *resonnering og argumentasjon*, *utforsking og problemløsing* og *abstraksjon og generalisering*. Begrepene til høyre i Figur 2.2 finner vi igjen fra overordnet del: *fikle (utforske og eksperimentere)* og *skape* kan knyttes til *Skaperglede, engasjement og utforskertrang*. Arbeidsmåtene *samarbeide* og *holde ut* samsvarer med deler av innholdet i *Prinsipper for læring, utvikling og danning* (Kunnskapsdepartementet, 2017). De ordene som ikke er enkle å knytte til andre deler av LK20, er *algoritmer*, *dekomposisjon*, *evaluering* og å *feilsøke*. Som begrep er altså algoritmisk tenkning tett knyttet til andre mål i læreplanen, men som samtidig tilfører nye elementer, slik som å bryte ned problemer i mindre deler (dekomposisjon), lage regler og steg-for-steg-løsninger (algoritmer), gjøre vurderinger av dem (evaluering) og søke etter feil (feilsøking).

Figur 2.2. Den algoritmiske tenkeren[2]

Mange av begrepene i Figur 2.2 kjenner vi igjen fra andre steder i LK20 (se Tabell 2.1). Tabell 2.1 viser hvordan begreper om algoritmisk tenkning er koblet til overordnet del (Kunnskapsdepartementet, 2017) og til læreplanen i matematikk (Kunnskapsdepartementet, 2020).

Tabell 2.1 Tilknytning mellom Den algoritmiske tenkeren og andre deler av LK20

Begreper fra Den algoritmiske tenkeren	Tilknytning til andre deler av læreplanen
Logikk	Kjerneelement: Resonnering og argumentasjon
Algoritmer	*Lite tilknytning til andre deler av læreplanen*
Dekomposisjon	Kjerneelement: Utforsking og problemløsning
Mønstre	Kjerneelement: Utforsking og problemløsning
Abstraksjon	Kjerneelement: Abstraksjon og generalisering
Evaluering	*Lite tilknytning til andre deler av læreplanen*
Fikle (utforske og eksperimentere)	Kjerneelement: Utforsking Overordnet del: Skaperglede, engasjement og utforskertrang
Skape	Overordnet del: Skaperglede, engasjement og utforskertrang
Feilsøke	*Lite tilknytning til andre deler av læreplanen*
Holde ut	Overordnet del: Prinsipper for læring
Samarbeide	Overordnet del: Prinsipper for læring

[2] Basert på et bilde fra Barefoot Computing (2024). Publisert med lisensen OGL.
https://www.udir.no/kvalitet-og-kompetanse/digitalisering/algoritmisk-tenkning/

Oppsummert defineres algoritmisk tenkning i LK20 og UDIRs veileder som en problemløsnings-kompetanse som innebærer å tenke som en informatiker. Dette dreier seg om å dekomponere problemer i enklere delproblemer og finne en fremgangsmåte som lager en løsning. Dette krever matematiske ferdigheter som logikk, modellering, abstraksjon og generalisering, men også mer generelle ferdigheter som å fikle, skape, holde ut og samarbeide. Dette samsvarer med andre deler av læreplanen, som kjerneelementene i matematikk og prinsippene for opplæring i overordnet del av LK20.

2.2 Algoritmisk tenkning i forskningslitteraturen

For å vite hvordan forskning om algoritmisk tenkning passer i en norsk utdanningskontekst, må vi vite hvordan LK20s definisjon av algoritmisk tenkning samsvarer med forskningens definisjoner. Dette er ikke enkelt, siden algoritmisk tenkning er et begrep som ikke har allment godtatte definisjoner i forskningslitteraturen. For eksempel fant en internasjonal litteraturgjennomgang hele 59 forskjellige definisjoner i 96 artikler om algoritmisk tenkning (Haseski m.fl., 2018).

Felles for definisjonene av *Algoritmisk tenkning* er at de forsøker å beskrive problemløsningsmetoder som informatikere er gode til og har spesielt stor nytte av (Wing, 2006). Det er påstått at algoritmisk tenkning er nyttig også innenfor andre områder enn informatikk, spesielt innenfor matematikk. Slike tanker er ikke nye (Babbage, 1864, s. 137; Knuth, 1974), men fikk fornyet kraft etter en meningsartikkel i et ledende tidsskrift innen datavitenskap (Wing, 2006). Siden den gang er det forsket mye på hva algoritmisk tenkning er og hvordan man kan bli flinkere til det. Algoritmisk tenkning er blitt innlemmet i læreplaner, spesielt i de nordiske landene (Vinnervik & Bungum, 2022), og med LK20 skal matematikklærere i Norge bidra til å utvikle elevenes algoritmiske tenkning. En definisjonen av algoritmisk tenkning som er mye benyttet de siste årene, og som er gjengitt nesten direkte i UDIRs (2024) definisjon av algoritmisk tenkning er en senere presisering:

> Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out (Wing, 2017, s. 8).

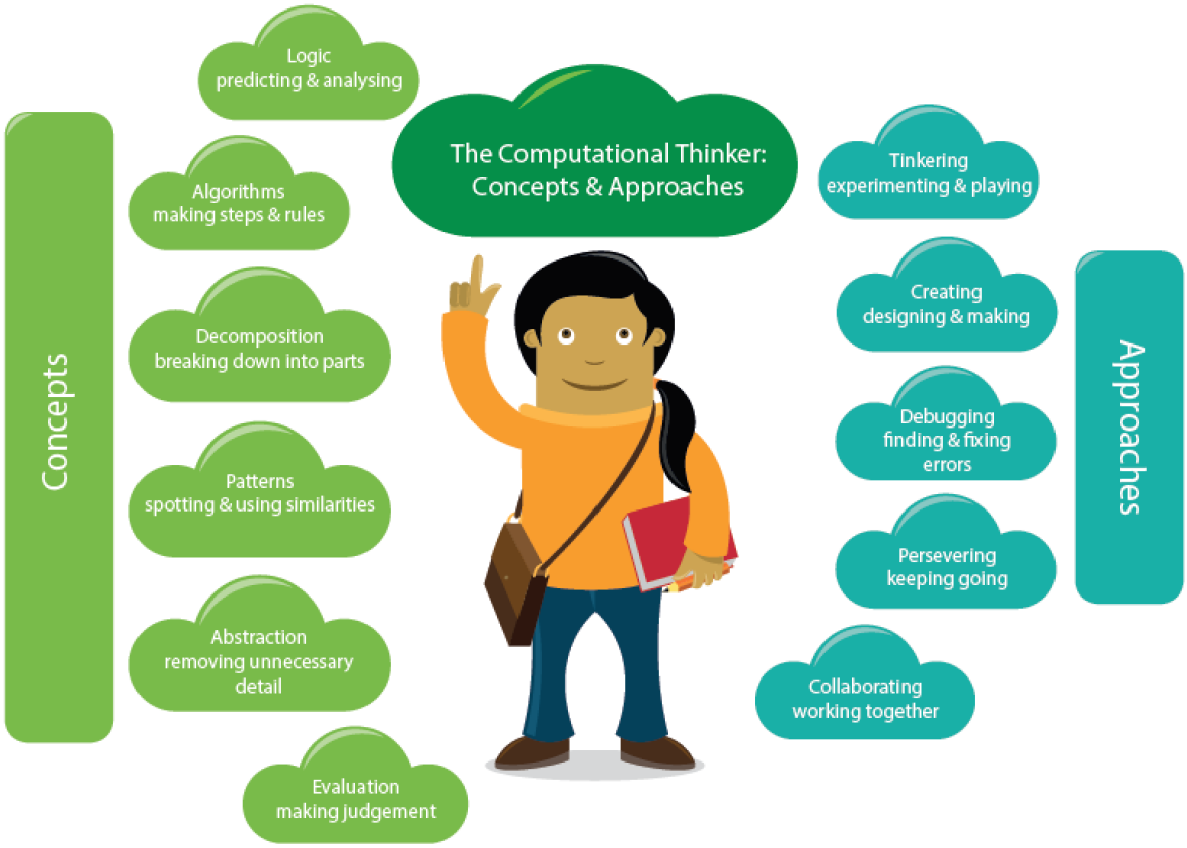
For å konkretisere hva algoritmisk tenkning innebærer må man altså finne ut hva slags tankeprosesser som er involvert i å formulere problemer slik at en maskin kan løse dem. Dette er et empirisk spørsmål, men det finnes lite empiri på det. De eksisterende rammeverkene for begrepet algoritmisk tenkning er rent teoretiske. Forskere og andre eksperter innenfor informatikk, matematikk, naturfag, problemløsning og liknende felt har tenkt ut hva algoritmisk tenkning kan innebære (Barr & Stephenson, 2011; Kallia m.fl., 2021; Weintrop m.fl., 2016). Andre og mer empiriske innfallsvinkler har vært å finne ut hva slags kompetanse skolebarn trenger når de løser programmeringsoppgaver (Brennan & Resnick, 2012) eller hvordan informatikere tenker når de løser programmeringsoppgaver (Boom m.fl., 2022). Foreløpig baserer forskningsfeltet seg nesten utelukkende på de teoretiske rammeverkene.

I det følgende beskriver vi to slike rammeverk for algoritmisk tenkning. Først går vi inn på rammeverket som LK20s definisjon og veileder er inspirert av. Dette belyser tradisjonen og tankegodset som ligger bak LK20. I og med at LK20s rammeverk er matematikknært, trekker vi deretter frem et annet rammeverk fra en annen tradisjon, for å kontrastere med et informatikknært rammeverk, som også er laget for å passe med skolens realfag (inspirert av en inndeling i Bocconi m.fl., 2022, s. 26).

Et matematikknært og konstruktivistisk rammeverk: Læreplanens og veilederens inspirasjon

Organisasjonen *Computing At School* lagde et rammeverk (Csizmadia m.fl., 2015) som mye av språket i LK20s beskrivelse av algoritmisk tenkning er basert på, og som nøkkelbegrepene og arbeidsmåtene er hentet fra (se Figur 2.2). *Computing At School* er en del av organisasjonen British Computer Society og er dels drevet av statlige midler og dels av donasjoner fra selskaper som Google, EPIC Games og ARM (Computing at School, 2024). Det er ikke beskrevet hvordan rammeverket er laget eller hva det er basert på, verken der det først ble presentert (Csizmadia m.fl., 2015) eller i en bok senere utgitt av samme organisasjon (Beecher, 2017), og vi har ikke klart å spore opp noe forskningsgrunnlag for rammeverket.

Figur 2.3 Et matematikknært rammeverk for algoritmisk tenkning[3]



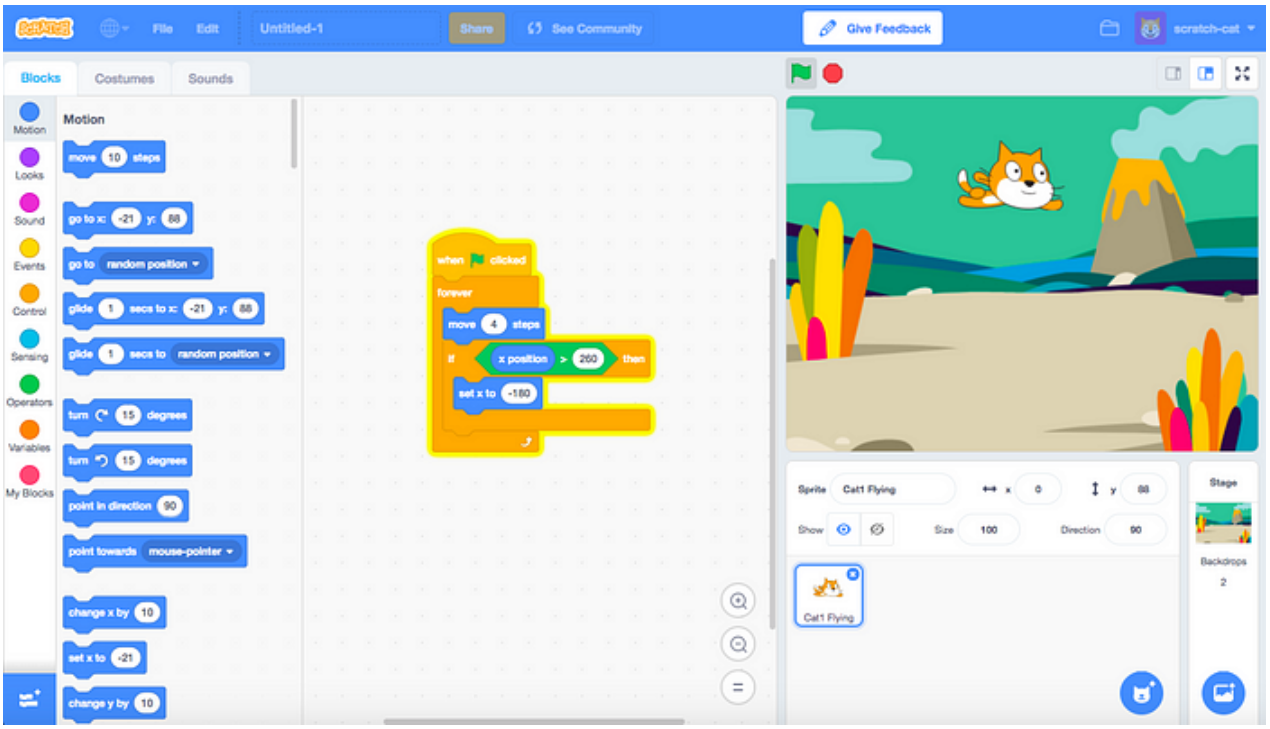
Rammeverkets venstre side viser *Concepts* som svarer til *Nøkkelbegreper* i UDIRs algoritmiske tenker (se Figur 2.2). Matematikkdidaktikere vil kjenne igjen alle begrepene som viktige innenfor matematikkundervisning, og det er uklart om nøkkelbegrepene beskriver noe som ikke allerede er vektlagt i matematikkfagene eller i utdanningen generelt. For eksempel beskriver nøkkelbegrepet *Algorithms: making steps & rules* et område som lenge har vært sentralt innenfor matematikkdidaktikk. Selv om matematikkfaget blir sett på som et fag som handler om å benytte seg av og å forstå algoritmer, ikke å lage dem, har matematikkdidaktikken de siste 40 årene fokusert på at elevene skal lage og oppdage de vanlige algoritmene under veiledning av læreren (Leinhardt, 1987).

[3] Hentet fra Csizmadia m.fl. (2015, s. 8), utgitt med lisensen CC-BY.

Dette var også vektlagt under LK06, med formuleringer som: «elevane skal kunne utvikle […] reknemetodar for addisjon og subtraksjon» (Kunnskapsdepartementet, 2013). Man kan argumentere tilsvarende for de andre nøkkelbegrepene. Dette rammeverket er altså matematikknært fordi det er uklart om nøkkelbegrepene beskriver (a) noe som er distinkt fra vanlig matematisk problemløsning, eller (b) innholdet i tidligere matematikklæreplaner.

Rammeverkets høyre side beskriver *Approaches* som den algoritmisk tenkende benytter seg av, og svarer til *Arbeidsmåter* i UDIRs algoritmiske tenker (se Figur 2.2). Disse arbeidsmåtene er lite kjent fra matematikkdidaktikken og benytter ord fra en konstruktivistisk tradisjon innenfor programmeringsdidaktikk. Seymour Paperts pionerarbeider på 1970-tallet viste at elever kan tilegne seg programmeringskunnskap uten særlig lærerstyrt undervisning ved å slippe dem løs i et velegnet miljø, i hans tilfelle et miljø med en robotskilpadde som tegnet traséen den hadde gått (Papert, 1980). Disse ideene inspirerte mange, blant annet Mitchel Resnick og hans miljø på Massachusetts Institute of Technology (MIT) som utviklet *Scratch* i 2003 (Resnick m.fl., 2009).

Figur 2.4 Blokkbasert programmering i Scratch[4]



Scratch er et grafisk programmeringsmiljø der man bygger programmer ved å sette sammen forskjellige typer blokker, såkalt blokkbasert programmering (se Figur 2.4). Blokkene kan styre figurer, lyder, musikk og annet, og man kan enkelt lage animasjoner, små spill og mye annet. Elevene kan prøve seg frem ved å trekke ulike blokker og sette dem sammen til programmer. De kan umiddelbart se hva de har laget i et vindu ved siden av blokkene.

[4] Hentet fra *The Scratch Blog*, utgitt under CC-BY

Det er i denne konstruktivistiske tradisjonen i programmeringsdidaktikken at arbeidsmåter som *tinkering* (fikling), *creating* (skaping), *persevering* (holde ut) og *collaborating* (samarbeid) blir benyttet. *Tinkering* er ifølge forskerne en «leken, utprøvende og iterativ måte å tilnærme seg et prosjekt eller problem på» (Resnick & Rosenbaum, 2013, s. 164, vår oversettelse). Gjennom *fikling* kan man gjøre små endringer, se hva som skjer og iterativt fullføre større prosjekter. Ved at elevene kan *skape* ting og se endringene underveis er tanken at de blir motiverte og utholdende. Dette synet på læring er oppsummert i slagord om at programmeringsundervisningen må være preget av *Projects, Passion, Peers and Play* og en *Lifelong kindergarten* (Resnick & Robinson, 2013). Det er fra disse miljøene at *Makerspaces* (skaperverksteder) har dukket opp ved universiteter, høyskoler, skoler og ellers. UDIRs beskrivelser av algoritmiske arbeidsmåter (Figur 2.2) har altså opphav i en skapende, lekende og elevsentrert tilnærming til læring.

Et informatikknært rammeverk

Et eksempel på et informatikknært rammeverk er fra Weintrop m.fl. (2016). Forskerteamet utviklet rammeverket ved å identifisere ulike praksiser som inngår i algoritmisk tenkning, søke i forskningslitteraturen og analysere eksisterende undervisningsopplegg. Deretter prioriterte de 27 praksiser som de sendte til lærere og forskere for evaluering. Resultatet kan ses i Tabell 2.2.

Tabell 2.2 Et informatikknært rammeverk for algoritmisk tenkning[5]

Data-praksiser	Modellerings- og simuleringspraksiser	Algoritmiske problemløsningspraksiser	Systemtenknings-praksiser
❖ Samle data ❖ Lage data ❖ Manipulere data ❖ Analysere data ❖ Visualisere data	❖ Utforme, lage og vurdere beregnings-modeller og benytte dem til å forstå begreper og finne løsninger	❖ Omforme problemer for algoritmiske løsninger. ❖ Programmering ❖ Velge beregningsverktøy og vurdere forskjellige tilnærminger ❖ Utvikle modulære beregnings-baserte løsninger og abstraksjoner ❖ Oppdage og rette feil	❖ Undersøke og håndtere komplekse systemer ❖ Forstå sammenhenger innad i systemer

Ordet *computational* oversatte vi til enten *algoritmiske eller beregnings-* (dvs. beregningsmodeller, beregningsverktøy, beregningsbaserte).

Det informatikknære rammeverket inneholder begreper som vil være godt kjent for programmerere og andre med informatikkbakgrunn og mindre kjent for de uten slik bakgrunn. For eksempel vil en informatiker forstå at å «utvikle modulære beregningsbaserte løsninger og abstraksjoner» kan bety å refaktorere en kode inn i funksjoner og datastrukturer som kan gjenbrukes, mens en matematikklærer uten informatikk-bakgrunn kanskje vil ha problemer med å forstå det samme. Rammeverket nevner *programmering* eksplisitt og inneholder flere praksiser som er mest naturlige å løse ved hjelp av programmering, eller i det minste ved hjelp av et dataverktøy, slik som de forskjellige datapraksisene og simuleringspraksisene. Dermed beskriver rammeverket problemløsningsmetoder som er distinkte fra de vi vanligvis forbinder med matematisk problemløsning og knytter seg tettere opp til problemløsning innenfor informatikk.

[5] Basert på Weintrop m.fl. (2016). Forenklet og oversatt av oss.

Oppsummert legger LK20s definisjon og operasjonalisering av algoritmisk tenkning lite vekt på programmering og informatikkbegreper, men mye vekt på matematiske problemløsningsstrategier (som logikk og abstrahering) og arbeidsmåter (som å skape, fikle, samarbeide og å holde ut). Dette er det viktig å ha i tankene når man leser forskningslitteraturen. Hvis en artikkel sier at den omhandler *computational thinking,* kan den dreie seg om informatikknære parametere som forståelse av løkker og vilkår, og ikke om problemløsningsstrategier og arbeidsmåter som er henvist til i LK20. I neste delkapittel viser vi hvordan de informatikknære begrepene om algoritmisk tenkning gjenspeiles i LK20s kompetansemål om programmering.

2.3 Programmering i læreplanen og til eksamen

Programmering er nevnt i kompetansemålene fra 5. trinn, se Tabell 2.3. Etter 5. trinn skal elevene programmere algoritmer med *variabler*, *vilkår* og *løkker*, og etter 6. trinn skal de i tillegg programmere med *funksjoner* og benytte programmering til å utforske geometriske mønstre. På 7. trinn skal de bruke programmering på datasett, på 8. trinn skal de utforske algoritmer med programmering, på 9. trinn skal de simulere tilfeldige forsøk og på 10. trinn skal de utforske matematiske egenskaper og sammenhenger. Det er ikke spesifisert i LK20 hvordan elevene skal programmere, for eksempel med blokkbasert- eller tekstbasert programmering. I kompetansemålene omtales mer informatikkspesifikke algoritmiske begreper som ikke nevnes i LK20s definisjon av algoritmisk tenkning, men som nevnes i det mer informatikknære rammeverket i Tabell 2.2. For eksempel svarer *variabler*, *vilkår*, *løkker* og *funksjoner* til praksisen *Programmering* i og kompetansemålene på 7. og 9. trinn svarer til *Datapraksiser*.

Tabell 2.3 Kompetansemål med programmering i matematikkfagene

Trinn	Kompetansemål
5.	lage og programmere algoritmer med bruk av variabler, vilkår og løkker
6.	bruke variabler, løkker, vilkår og funksjoner i programmering til å utforske geometriske figurer og mønstre
7.	bruke programmering til å utforske data i tabeller og datasett
8.	utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering
9.	simulere utfall i tilfeldige forsøk og beregne sannsynligheten for at noe skal inntreffe, ved å bruke programmering
10.	utforske matematiske egenskaper og sammenhenger ved å bruke programmering
Vg1 (1T)	formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsingsstrategier, digitale verktøy og programmering
Vg1 (1T-Y)	*programmering er ikke nevnt i kompetansemålene*
Vg1 (1P)	*programmering er ikke nevnt i kompetansemålene*
Vg1 (1P-Y)	*programmering er ikke nevnt i kompetansemålene*

Programmeringskompetansen skal også måles på sentralt gitt skriftlig eksamen. Lærere ser gjerne til eksamen for å vurdere hva de skal legge vekt på i undervisningen, noe som kan forklare hva vi ser av programmeringsundervisning i timene. Tidligere gitte eksamensoppgaver spesifiserer ikke blokk- eller tekstprogrammering, men i den nyeste eksamensveiledningen fra våren 2024 på 10. trinn[6] står det at elevene kan velge å bruke som hjelpemiddel «Programmeringsverktøy (både blokk og tekst)». I tillegg kan de «velge å bruke programmering som løsningsstrategi for å løse et matematisk problem. Det kan også være oppgaver hvor et matematisk problem er representert som en programkode. Kandidaten skal vurdere innholdet i koden og bruke denne til å løse det matematiske problemet.» Elevene kan altså benytte programmering dersom de vil, og de må forvente oppgaver der noe er representert ved programkode. Algoritmisk tenkning er ikke nevnt i eksamensdokumentene.

2.4 Algoritmisk tenkning og programmering i matematikkfaget

Forskningen om undervisning av algoritmisk tenkning og programmering i matematikkfaget har stort sett blitt gjort på barneskolen (Lv m.fl., 2023). På ungdomstrinnet og i videregående skole består forskningen stort sett av kasusstudier gjennomført av enkeltlærere (Lv m.fl., 2023), gjerne der forskere og lærere sammen prøver ut undervisningsmetoder. Disse kasusstudiene viser frem muligheter lærere kan benytte når de underviser algoritmisk tenkning og programmering. Samtidig er det uklart i hvilken grad disse kasusstudiene er overførbare, altså om de gir god kunnskap om hvordan det kan gå hvis andre lærere forsøker å undervise på samme måte i sine klasser. Nedenfor presenterer vi noen slike eksperimenter, og trekker frem studier som har undersøkt både (a) læring av matematikk og (b) algoritmisk tenkning eller programmering.

Munthe (2022) analyserte læringspotensialet i forskjellige utfordringer elever møter på når de skal programmere i Matematikk R1 og R2 på videregående skole. Han fant at *kodeutfordringer*, altså hvordan man omgjør matematiske løsninger til kode, fører til matematikklæring, men at *konseptutfordringer*, altså å forstå ulike programmeringskonsepter, ikke gjør det. Både *syntaksutfordringer*, å forstå hvordan betingelser og løkker endrer programflyten, og *outpututfordringer*, å forstå hva programmeringsmiljøet sier når du kjører programmet, kan føre til læring hvis elevene utforsker uventede resultater. Munthe (2022) fulgte kun én klasse med 28 elever, men den lange varigheten (10 økter på 90 minutter hvert skoleår) og dybden på eksperimentet (skjermopptak og lyd fra alle elever) styrker troen på at disse resultatene er overførbare til liknende undervisningssituasjoner.

Noen av studiene undersøkte symboler eller begreper som har ulik – men likevel nært beslektet – betydning innenfor programmering og matematikk. De symbolene eller begrepene som er analysert er *variabler* (Kilhamn m.fl., 2022), *likhetstegnet* (Jones & Pratt, 2006) og *funksjoner* (Bråting & Kilhamn, 2021). Et gjentagende argument i diskusjonsdelen av disse studiene er at elever allerede strever med at matematiske begreper har ulike betydninger avhengig av kontekst, slik som at ordet *variabel* kan betegne en plassholder, en ukjent og en variabel, og at det derfor må en kunnskapsrik lærer til for å skape større forståelse og ikke mer forvirring når programmeringen introduserer flere betydninger av de samme begrepene.

[6] https://www.udir.no/eksamen-og-prover/eksamen/

Et eksempel er Kilhamn m.fl. (2022) sin analyse av programmeringsaktiviteter som lærere i Sverige benyttet for 10- til 12-åringer. De fant tre ekstra sider ved ordet *variabel* som elevene måtte beherske da de programmerte: (1) at det blokkbaserte programmeringsmiljøet Scratch har mange implisitte variabeltilordninger, (2) at tellevariable er enkle og naturlige i programmeringsmiljøer, men abstrakte og tungvinte i matematikk, og (3) at variabler må være tilordnet en verdi i programmering, men ikke i matematikk. Vi har kun funnet studier som identifiserer hvordan programmering innfører flere betydninger av matematiske begreper og ingen studier som undersøker hvordan man kan hjelpe elevene med å forstå de nye betydningene.

Et interessant eksempel på at programmering kan gi matematisk innsikt er i Kilhamns (2022) kasusstudie av læreren John. John underviser figurtall og sliter med at elevene ikke ser verdien i å finne den generelle formelen som beskriver figurtallene. Elevene fikk en aha-opplevelse da John lagde en løkke som regnet ut mange figurtall:

> Then I showed the class how to modify this code so that instead of asking for a certain figure it prints the first 50 figures with the number of sticks in. And then you can see that: 'Well there are 55 sticks in figure number 17'. And this – when they suddenly saw – they had it all there! It was like 'wow' they could do anything, all information was there in this table that was printed. (Kilhamn, 2022, s. 5).

Programmering hjalp altså elevene med læring av algebra, fordi elevene innså at de måtte uttrykke det de tenkte algebraisk for å kunne kommunisere med datamaskinen.

Elever kan også lære algoritmisk tenking i matematikkfaget ved bruk av regneark (Sanford, 2018), for eksempel Microsoft Excel. Fra et rent teoretisk perspektiv argumenterer Sanford (2018) for at regneark er godt egnet til å lære seg algoritmisk tenkning fordi det er så visuelt. Når man løser et matematisk problem med regneark, blir omformingen eleven gjør for å kunne lage en beregningsbasert løsning synlig på skjermen: radene og kolonnene eleven velger å bruke *er* selve omformingen. Forsøk har vist at regneark kan benyttes for å øve på begreper som er både matematiske og programmeringsbegreper, slik som *parameter*, *funksjon*, *formel* og *vilkår* (van Borkulo m.fl., 2023). Regneark kan også benyttes slik at elevene synes statistikkundervisningen er nyttig og har overføringsverdi til sitt fremtidige liv (van Borkulo m.fl., 2023). Regneark er heller ikke begrenset til enkel algoritmisk tenkning eller enkle programmeringskonsepter (Csapó m.fl., 2020).

2.5 Algoritmisk tenkning og programmering i matematikkfaget etter LK20

Forholdsvis mye forskning gjort i Norge etter LK20 har kartlagt hva lærere mener om programmering og algoritmisk tenkning i matematikkfaget og hvordan de vurderer egen kompetanse. I tillegg har noe forskning i Norge undersøkt elevenes holdninger til og kunnskap om algoritmisk tenkning og programmering, samt endringen i programmeringskompetansen til førsteårsstudenter i høyere utdanning etter LK20.

Læreres holdninger til og kunnskaper om algoritmisk tenkning og programmering

Den studien som trolig er mest representativ for norske matematikklæreres holdninger og (selvrapporterte) kompetanse etter innføringen av LK20 er en spørreundersøkelse blant 365 matematikklærere i Norge (Turgut m.fl., 2024). Utvalget ble rekruttert ved å spørre rektorer ved tilfeldig valgte skoler om de kunne videresende undersøkelsen til lærere. Denne utvalgsprosessen er god, men kan likevel gi et skjevt utvalg av flere grunner, for eksempel hvis rektorer som har programmeringsinteresserte lærere er mer tilbøyelige til å være med på studien. Trolig er utvalget ikke representativt, siden 60 % av lærerne i det endelige utvalget jobbet i videregående skole.

Vi har valgt ut noen av spørsmålene i undersøkelsen som illustrerer de overordnede resultatene og viser hvordan lærerne svarte (se Figur 2.5). Oppsummert hadde lærerne stort sett positive holdninger til algoritmisk tenkning og programmering, men mange lærere rapporterte om utfordringer og dårlig selvtillit. Lærere i videregående skole hadde mindre positive holdninger til algoritmisk tenkning og programmering enn lærere på lavere trinn, men ellers svarte erfarne og uerfarne lærere og lærere i forskjellige skoleslag likt.

Figur 2.5 viser at lærerne stort sett hadde positive holdninger til programmering og algoritmisk tenkning. Over halvparten av lærerne var svært enige eller ganske enige i at algoritmisk tenkning og programmering i matematikk kunne utvikle elevenes problemløsningsferdigheter, hjelpe elevene å tenke abstrakt, og gjøre matematikken morsommere, men en ganske betydelig andel (17 %) mente at elever i mindre grad burde bruke programmeringsverktøy. Mange lærere uttrykte at de hadde utfordringer med algoritmisk tenkning og programmering, spesielt med å knytte det til matematikkundervisningen. Rundt halvparten av lærerne var enige i at de hadde utfordringer med å bruke verktøy for algoritmisk tenkning og programmering i matematikkundervisningen og at det var vanskelig å lage oppgaver. Rundt en tredel var enige i at det var vanskelig å feilsøke kode og at de manglet selvtillit med å bruke verktøy for algoritmisk tenkning og programmering i undervisningen. En tidel var enige i at de ofte måtte spørre elevene om hjelp.

Troen på at resultatene fra denne spørreundersøkelsene er representative, er at liknende resultater er funnet i en lang rekke masteroppgaver som gjengir intervjuer av lærere (se for eksempel Voldmo, 2022; Sandal, 2024).

Én annen masteroppgave (Thune, 2023), som er mer gjengitt nedenfor, har data fra 16 lærere rundt om i landet. Da de ble spurt hva de mener med algoritmisk tenking, krysset nesten alle lærerne av for *designe algoritmer som løser et problem* (94 %), *dekomposisjon* (75 %), *evaluere algoritmer* (50 %), mens nesten ingen kjente igjen de algoritmiske arbeidsmåtene, slik som *utforske og eksperimentere med et problem* (13 %), *dele og jobbe sammen om et problem* (6 %), *oppdage og rette feil* (6 %) og *utholdenhet* (0 %). Altså forbinder lærere algoritmisk tenkning mer med nøkkelbegrepene enn med arbeidsmåtene i UDIRs veileder om den algoritmiske tenkeren (se Figur 2.2).

Figur 2.5 Læreres holdninger og egenrapporterte kompetanse til algoritmisk tenkning- og programmerings-verktøy (AToP)[7]

Elevenes holdninger til og kunnskap om algoritmisk tenkning og programmering

Norge var med på den internasjonale undersøkelsen ICILS 2023, som blant annet måler elevenes kompetanse innenfor algoritmisk tenkning. Norske elever presterte likt med gjennomsnittet for deltagerlandene, likt med svenske elever og dårligere enn danske og finske elever (Rohatgi m.fl., 2024). Til sammen 14 % av elevene skåret under nivå 1, som betyr at de har veldig lite kunnskap om algoritmisk tenkning, mens 26 % av elevene skåret på nivå 1, som betyr at de kan gjenkjenne løkker og vilkår og utvikle helt elementære algoritmer. 41 % av elevene oppnådde nivå 2, som blant annet betyr at de kan benytte ulike kommandoer i blokkprogrammering til å løse problemer som krever betingelser og repetisjoner, mens 18 % oppnådde nivå 3 (løse moderat komplekse problemer med en kombinasjon av kommandoer) og 1 % oppnådde nivå 4 (dekomponere komplekse problemer og løse delproblemene ved å lage algoritmer).

[7] Laget på bakgrunn av data i Turgut m.fl. (2024) sin spørreundersøkelse. Prosentene viser andel "enig" eller "ganske enig". Se Appendix.

Foreløpig er kun den norske kortrapporten fra ICILS utgitt. Elevenes erfaring med datamaskiner og deres digitale ressurser hjemme (f.eks. antall maskiner, maskin tilgjengelig for skolearbeid og kvalitet på internettforbindelse) har positiv sammenheng med elevens prestasjoner på prøven i algoritmisk tenkning (Rohatgi m.fl., 2024, s. 25). I rapporten forklares dette med at disse kjennetegnene er knyttet til elevenes sosiale bakgrunn, og dette vil bli nærmere undersøkt i antologi som er planlagt publisert i slutten av 2025. Når videre rapportering fra den norske ICILS 2023 blir offentliggjort kommer denne til å gi mer detaljert kunnskap om norske elevers kunnskaper om algoritmisk tenkning. Norge deltok ikke i ICILS 2018, som var den første versjonen av ICILS som målte algoritmisk tenkning. Dette gjør at ICILS ikke gir mulighet for å undersøke utviklingen av norske elevers kompetanse i algoritmisk tenkning og programmering rundt innføringen av LK20.

Vi vet foreløpig lite om norske elevers holdninger til algoritmisk tenkning og programmering. Det største arbeidet er en masteroppgave som ønsket å rekruttere minst tre matematikk R1-klasser fra hvert fylke til en spørreundersøkelse og som lyktes med å rekruttere fra store deler av landet (Thune, 2023). Våren 2023 svarte 260 elever og 16 lærere på spørreundersøkelsen. Rundt 60 % av elevene kjente igjen følgende ord som algoritmisk tenkning: *koding*, *løse problemer med algoritmer*, *systematisk databehandling* og *lage algoritmer*, mens under 20 % av elevene kjente igjen arbeidsmåter som *utholdenhet* og *samarbeid* som algoritmisk tenkning. Dette tyder på at verken lærere eller elever kjenner igjen de algoritmiske arbeidsmåtene som algoritmisk tenkning. Resultatene viste at elevene var litt mindre motivert for algoritmisk tenkning og programmering enn for resten av matematikkfaget, og de jobbet mindre utenom skoletid med programmeringstemaene enn med de andre temaene i faget. Elevene rapporterte at de i klasserommet *leser og tolker kode* (82 %), *kopierer og tilpasser kode* (72 %), *kopierer lærers kode* (65 %) eller *utvider utdelt program* (54 %). Langt færre elever (26 %) svarte at det har blitt *overlatt til elevene* å lage koden selv.

Programmeringskompetansen til førsteårsstudenter i høyere utdanning har økt etter LK20

Ved starten av hvert skoleår gis *Nordisk forkunnskapstest i programmering*[8] til studenter ved universiteter og høyskoler som tar fag som inneholder et grunnkurs i programmering. Den har blitt gitt siden høsten 2023, som samsvarer med når det første kullet som hadde algoritmisk tenkning og programmering på videregående skole startet å studere ved høyskoler og universiteter. Dette gjør at forkunnskapstesten ikke har data fra før og etter LK20, men ved å sammenlikne studentene som startet rett fra videregående skole med de som gikk i videregående skole før LK20, kan testene fra 2023 og 2024 gi et bilde på om programmeringsferdighetene har økt.

Analysene vi rapporterer på er basert på 3 038 studenter som startet på et studium høsten 2024 som inneholdt et grunnkurs i informatikk. De studentene som gikk på videregående skole etter LK20 skåret høyere enn studentene som gikk på videregående skole før LK20 (Bolland m.fl., 2024; Bolland & Strømme, 2025). Dette gjaldt både for hele studentmassen og for den delen av studentmassen som ikke hadde tatt valgfag med programmering eller hadde erfaring med programmering utenfor skolen. Etter LK20 var det færre studenter med lite til ingen programmeringskunnskap, mange flere med middels programmeringskunnskaper og veldig mange flere med god programmeringskunnskap.

[8] www.programmeringstesten.no

Kjønnsforskjellene har blitt mindre etter LK20, guttene hadde nesten dobbel så høy skår som jentene før LK20, men etter LK20 hadde de halvannen gang så høy. Det var kun studentene med R- og S-matematikk som har forbedret skåren etter LK20, studentene med 2P-matematikk hadde faktisk svakere resultater. Analysene tar hensyn til forskjeller i erfaring med programmering utenom skolen og om studentene hadde pauseår mellom Vgs og studium. Vi vurderer dette som tydelige bevis på at LK20 har forbedret programmeringskompetansen betraktelig blant elever som søker seg til utdanninger med et grunnkurs i informatikk i første semester.

2.6 Protokoll for algoritmisk tenkning og programmering (EDUCATE 1.0)

Et hovedbidrag i EDUCATEs Rapport 5 er utviklingen av *EDUCATEs observasjonsprotokoll for algoritmisk tenkning og programmering* for matematikkundervisningen. Vi baserte protokollen først og fremst på beskrivelsene av algoritmisk tenkning og programmering i læreplanene for 1–10, 1P og 1T og yrkesfaglig matematikk (Kunnskapsdepartementet, 2020b, 2020c, 2020a) slik at det vi observerer samsvarer med læreplanverket LK20.

Fra et matematikknært rammeverk for algoritmisk tenkning til *Protokoll for algoritmisk tenkning*

For å beskrive hva som ligger i de ulike problemløsningsmetodene (abstraksjon, logikk) og arbeidsmåtene (fikle, feilsøke) så vi til rammeverket til Csizmadia m.fl. (2015), som læreplanene i matematikk er inspirert av. Vi kunne utarbeidet observasjonsprotokollen basert på andre rammeverk for algoritmisk tenkning, for eksempel det mer informatikknære rammeverket til Weintrop m.fl. (2016) som vist i Tabell 2.2, eller Brennan og Resnicks (2012) empiriske innfallsvinkel med tanke på programmeringsoppgaver. Begge disse har styrker knyttet til at kategoriene deres er mer konkrete og lettere å observere. Det er for eksempel lettere å vurdere om elevene *simulerer data* (Weintrop m.fl., 2016) enn om de *holder ut* (UDIR, 2019). Vi besluttet likevel å bygge på UDIRs beskrivelse av den algoritmiske tenkeren (2019) fordi det er dette lærere er bedt om å forholde seg til i LK20.

Vi baserte protokollen først og fremst på beskrivelsene av algoritmisk tenkning og programmering i læreplanene, slik at vi måler noe som samsvarer med læreplanen. Dette førte til at vi deler begrepet *algoritmisk tenkning* i to; *algoritmisk problemløsning* og *algoritmiske arbeidsmåter*. Vi lagde også en kode for *programmering*, for å kunne beskrive hvordan elevene jobber med programmering uavhengig av algoritmisk tenkning. Dette er de tre elevkodene, som måler hva elevene gjør. Til sist har vi en lærerkode, som måler hva læreren gjør, også når det ikke er elevaktivitet. For å beskrive hva som ligger i de ulike problemløsningsmetodene (abstraksjon, logikk) og arbeidsmåtene (fikle, feilsøke) så vi til Csizmadia og kollegaers rammeverk (2015) som læreplanen er inspirert av. Vi operasjonaliserte de fire kategoriene på fire nivåer: fravær av arbeid med algoritmisk tenkning (nivå 1), tilstedeværelse av algoritmisk tenkning (nivå 2), skår 3 betyr at koden er tydelig til stede og noenlunde etter intensjonene i læreplanen. Skår 4 betyr at undervisningen inneholder et meget godt eksempel på koden etter intensjonene i læreplanen. Kjernen i protokollen ble dermed en observasjonsmatrise som består av fire nivåer (1–4), og fire kategorier, der én gjelder læreren og tre gjelder elevene.

Algoritmisk problemløsning: Denne kategorien er basert på nøkkelordene i Den Algoritmiske Tenkeren (se Figur 2.2): *logikk*, *algoritmer*, *dekomposisjon*, *mønstre*, *abstraksjon* og *evaluering.* Under utviklingen av kategorien, vektla vi de fire første nøkkelordene mest, fordi det var vanskeligere å kode elevenes abstraksjon og evaluering dersom dette ikke ble satt ord på i undervisningen av enten lærer eller elever. Kategorien fanger opp når det ikke drives med algoritmisk problemløsning (skår 1), og når elevene øver på, bruker eller prøver å forstå algoritmer (skår 2). Da tenker vi på kjente algoritmer som man lærer seg i løpet av skolegangen, slik som å stille opp utregninger med de fire regneartene, regne felles nevner, regne prosenten av et tall med veien om én, tegne en graf fra en verditabell, polynomdividere, eller utføre delvis integrasjon. Kategorien fanger også når elevene selv benytter dekomponering, mønstergjenkjenning eller logikk i noen grad (skår 3), og gjerne til å skape algoritmer med iterasjon eller rekursjon som kan automatiseres (skår 4).

Algoritmiske arbeidsmåter: I denne kategorien er det å *fikle* den grunnleggende algoritmiske arbeidsmåten, og manglende fikling indikerer fravær (skår 1). Uten dette kravet kan alt karakteriseres som en algoritmisk arbeidsmåte; elevene kan *holde ut* uten at det er spesielt algoritmisk, samarbeide uten at det er spesielt algoritmisk, skape noe uten at det er spesielt algoritmisk, og så videre. Hvis elevene driver med noen grad av fikling uten at det virkelig preger elevenes arbeid gis skår 2. Hvis elevene i større grad fikler, gjerne i det elevene skaper et produkt eller driver med feilsøking, alene eller sammen med andre, gisskår 3. Elevenes arbeid kan også være sterkt preget av at de fikler sammen for å skape noe, søker etter feil og holder ut (skår 4).

Programmering: Programmering på 8. trinn, 10. trinn og Vg1 dreier seg ikke lenger om å sette seg inn i et programmeringsspråks syntaks, fordi elevene har lært om variable, vilkår, løkker og funksjoner på mellomtrinnet. De må trolig lære seg syntaksen til et tekstbasert programmeringsspråk på ungdomstrinnet, i hvert fall hvis de kun har benyttet blokkbasert programmering på barnetrinnet, men de skal også kunne benytte programmering til andre ting. Derfor kan vi observere at elevene ikke programmerer (skår 1) eller at de gjør oppgaver ment for å lære seg et programmeringsspråks syntaks (skår 2). Elevene kan også løse korte programmeringsoppgaver (skår 3) eller omfattende programmeringsoppgaver (skår 4). Et eksempel på en kort programmeringsoppgave er å ta inn verdier for variabel `a` og `b` og printe `a^2+b^2` til skjerm. Et eksempel på en omfattende programmerings-oppgave er en oppgave som krever større kombinasjon av programmeringskonsepter, som løkker, vilkår eller funksjoner, for eksempel å lage en funksjon som printer alle heltallsdivisorene til et tall.

Oppsummert tar *EDUCATEs observasjonsprotokoll for algoritmisk tenkning og programmering* høyde for både lærerens undervisning og hvordan elevene arbeider med dette i matematikkfagene. De fire kategoriene gjør det mulig å fange situasjoner der læreren underviser om eller legger til rette for elevenes arbeid med algoritmisk tenkning og programmering (skår 2–4), uten at vi nødvendigvis observerer elevenes arbeid med dette. Det motsatte kan også skje, at vi observerer elevenes arbeid med algoritmisk tenkning og programmering (skår 2–4), uten at vi observerer at læreren underviser om det eller legger til rette for det. Likevel vil lærerkategorien som regel skårer likt eller høyere enn den høyeste av de tre elevkategoriene, i og med at dersom elevene arbeider med algoritmisk tenkning i matematikktimen har læreren oftest lagt til rette for det.

I tillegg til selve observasjonsmatrisen (se side 2 av 5 i Boks 3) inkluderer protokollen definisjoner og beskrivelser av algoritmisk tenkning og programmering, samt avklaringer av begreper og skåringsprosessen. Protokollen er *ikke* ment å være normativ, men bevisstgjørende. Målet er å skape et felles språk og et felles analytisk utgangspunkt når vi ser etter algoritmisk tenkning og programmering i matematikklasserommet.

Boks 2 EDUCATEs operasjonalisering av algoritmisk tenkning og programmering

EDUCATEs observasjonsprotokoll for algoritmisk tenkning og programmering operasjonaliseres slik:

- **Lærerens undervisning:** Denne kategorien skåres på bakgrunn av hva læreren sier og gjør i timen. Det kan være å legge til rette for at elevene arbeider med algoritmisk problemløsning, algoritmiske arbeidsmåter eller programmering, eller å forklare noe relatert til dette. *Første kategori (første horisontale linje) i protokollen handler derfor om lærerens undervisning.*
- **Algoritmisk problemløsning:** Denne kategorien fanger algoritmisk tenkning som en problemløsningsmetode, basert på nøkkelordene *logikk*, *algoritmer*, *dekomposisjon*, *mønstre*, *abstraksjon* og *evaluering*. Elevene kan bedrive algoritmisk problemløsning også uten å programmere. *Andre kategori (andre horisontale linje) handler derfor om elevenes algoritmiske problemløsning.*
- **Algoritmiske arbeidsmåter:** Denne kategorien fanger arbeidsmåter som forbindes med algoritmisk tenkning, inkludert å *fikle*, *skape*, *feilsøke*, *holde ut*, og *samarbeide*. Å fikle handler om å prøve ut og eksperimentere på en leken måte, ikke nødvendigvis med dyp tenkning, men for å lære ved å prøve, se hva som skjer og deretter iterere videre. Å skape handler om å lage et produkt, gjerne som skal brukes til noe eller vises frem, og ikke bare være svar på en oppgave. Å feilsøke handler om å finne ut om de har gjort noen feil, lete etter dem og rette dem opp. Å holde ut og samarbeide har ingen betydning utover hverdagsbetydningen. *Tredje kategori (tredje horisontale linje) handler derfor om elevenes algoritmiske arbeidsmåter.*
- **Programmering:** Denne kategorien fanger ulike nivåer av programmering. Det kan handle om å lære syntaksen til et tekstbasert programmeringsspråk, men også å kunne benytte programmering til andre ting, både korte og omfattende programmeringsoppgaver. *Fjerde kategori (fjerde horisontale linje) handler derfor om elevenes arbeid med programmering.*

Boks 3 EDUCATE 1.0 Protokoll for observasjon av algoritmisk tenkning og programmering

Denne protokollen legger til grunn definisjonen av algoritmisk tenkning fra UDIR (2019):

> Algoritmisk tenkning innebærer å bryte ned komplekse problem til mindre, mer håndterlige delproblemer som lar seg løse. Det inkluderer å organisere og analysere informasjon på en logisk måte og å lage fremgangsmåter (algoritmer) for å komme fram til ønsket løsning. Det handler også om å lage abstraksjoner og modeller av den virkelige verden ved å fjerne unødvendige detaljer og fokusere på det som er relevant for den aktuelle problemstilling og løsning. En løsning på et spesifikt problem kan ofte generaliseres, slik at den kan brukes til å løse lignende problemer, og løsninger på flere delproblemer kan kombineres for å løse mer komplekse problem. (UDIR, 2019)

Videre legger vi i denne protokollen til grunn UDIRs beskrivelse av den algoritmiske tenkeren (2019):

> Den algoritmiske tenkeren må være systematisk og analytisk i sitt arbeid, men det er minst like viktig å være skapende, eksperimenterende og åpen for alternative løsninger. Det krever en nysgjerrig og utforskende tilnærming for å formulere og løse problemer. Å gjøre feil underveis er en viktig del av prosessen, og den algoritmiske tenkeren må ha strategier for å oppdage at noe er feil og rette feilene. Dette krever en «kognitiv kondis» for å ikke gi opp, og det er viktig å trene denne ved å jobbe med kontinuerlig forbedring underveis i prosessen. Gode løsninger oppstår ikke et vakuum, og samarbeid og deling er derfor sentrale arbeidsmetoder for den algoritmiske tenkeren. (UDIR, 2019)

I. Overblikk: Hva handler dette temaet om?

Definisjonen vektlegger algoritmisk tenkning som en problemløsningsmetode som er forbundet med noen spesifikke arbeidsmåter som er typisk for «den algoritmiske tenkeren». Definisjonen er ment å fange opp hvordan man kan «tenke som en informatiker» når man møter et problem. Algoritmisk tenkning trenger ikke å knyttes til teknologi, men når elever jobber med programmering vil mange av oppgavene ha innslag av algoritmisk tenkning. For å evaluere *om* og i tilfellet *hvordan* algoritmisk tenkning operasjonaliseres i undervisningen, vil vi derfor se etter bevis for (1) *om* undervisningen knyttes til arbeidsmåter som har betydning for algoritmisk problemløsning, og i så tilfelle (2) *hvordan* dette gjøres.

Protokollen omfatter fire kategorier der den første fanger opp (1) innholdet i *lærerens undervisning*, og de resterende fanger opp forskjellige sider ved elevenes arbeid, (2) *algoritmisk problemløsning*, (3) *algoritmiske arbeidsmåter* og (4) *programmering*.

1. *Innhold*: Denne kategorien analyserer innholdet i timen og retter oppmerksomheten mot lærerens undervisning.

2. *Algoritmisk problemløsning*: Denne kategorien analyserer elevenes deltakelse i læringsaktiviteten i timen. Vi ser om elevene jobber med algoritmisk problemløsning med eller uten programmering.

3. *Algoritmiske arbeidsmåter*: Denne kategorien analyserer hvordan elevene arbeider. UDIRs algoritmiske arbeidsmåter er å *fikle*, *skape*, *feilsøke*, *holde ut* og *samarbeide*.

4. *Programmering*: Denne kategorien fanger opp når elevene programmerer eller analyserer eksisterende programmer.

II. Matrise: Hvordan skårer vi dette temaet?

ALGORITMISK TENKNING OG PROGRAMMERING					
EDUCATEs protokoll for observasjon av **Algoritmisk tenkning og programmering** evaluerer om og hvordan undervisningen bidrar til at elevene utvikler sin algoritmiske tenkning og programmeringsferdighet. Dette kan observeres i ulike fag, men denne protokollen er rettet spesielt mot matematikkfagene. For å komme tett på undervisning, deles hver undervisningstime inn i 15-minutters segmenter og hvert segment gis en skår på 1–4. Algoritmisk tenkning og programmering omfatter ikke nødvendigvis alle kategorier. Den som evaluerer, bør derfor skåre hver kategori separat for hvert segment.					
Hovedkategorier		**1** **Ingen observasjon av algoritmisk tenkning**	**2** **Tilstedeværelse av algoritmisk tenkning**	**3** **Eksplisitt arbeid med algoritmisk tenkning**	**4** **Aktualisert arbeid med algoritmisk tenkning**
Lærer	*Lærerens undervisning*	Lærer bruker ikke og legger ikke til rette for algoritmisk problemløsning, programmering eller bruk av algoritmiske arbeidsmåter.	Lærer viser eller legger til rette for at elevene lærer en kjent algoritme for å løse en gitt type oppgave, syntaksen til et programmeringsspråk eller enkel bruk av algoritmiske arbeidsmåter.	Læreren viser eller legger til rette for enkel algoritmisk problemløsning, forståelse av programmeringskonsepter, eller algoritmiske arbeidsmåter.	Lærer viser eller legger til rette for løsning av problemer som krever avansert algoritmisk problemløsning eller omfattende programmeringsarbeid og legger til rette for bruk av algoritmiske arbeidsmåter.
Elever	*Algoritmisk problemløsning*	Elevene jobber ikke med læringsaktiviteter som krever algoritmisk problemløsning.	En eller flere elever jobber med en algoritme for å løse en gitt type oppgave.	En eller flere elever bedriver enkel algoritmisk problemløsning.	En eller flere elever bedriver avansert algoritmisk problemløsning.
	Algoritmiske arbeidsmåter (fikle, skape, feilsøke, holde ut, samarbeide)	Elevene benytter ikke algoritmiske arbeidsmåter.	En eller flere elever fikler, alene eller sammen med andre.	En eller flere elever fikler og skaper eller feilsøker, alene eller sammen med andre.	En eller flere elever benytter de algoritmiske arbeidsmåtene i utstrakt grad.
	Programmering	Elevene jobber ikke med programmering.	En eller flere elever jobber med oppgaver for å lære syntaksen til et programmeringsspråk.	En eller flere elever løser korte programmerings-oppgaver.	En eller flere elever løser omfattende programmeringsoppgaver.

III. Avklaring av begreper

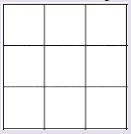
1. Hva er en *algoritme*?
   - En algoritme er en nøyaktig beskrivelse av en steg-for-steg-fremgangsmåte for å løse en bestemt oppgave. Dette kan være standardalgoritmer i matematikk (f.eks. å gange inn i parenteser, sette på felles nevner) eller andre algoritmer (f.eks. for å sortere en liste alfabetisk eller en prosedyre for å finne veien ut av en labyrint).

2. Hva er «en kjent algoritme for å løse en gitt type oppgaver»?
   - Det kan være standardprosedyrer i matematikk (oppstilt subtraksjon, gange inn i parenteser, finne største felles nevner, regne prosenten av et tall) eller noen vanlige programmeringsoperasjoner (finne det største tallet i en liste).
   - Algoritmen må presenteres som kjent og fokuset er på at elevene skal gjengi eller benytte seg av den.

3. Hva er algoritmisk problemløsning?
   - Algoritmisk problemløsning er en problemløsningsmetode som benytter *dekomponering* (å bryte ned problemet i steg eller deler) og *logikk* eller *mønstergjenkjenning* for å lage en løsning som potensielt kan *automatiseres*, gjerne ved *iterasjon*.

4. Hva er «enkel algoritmisk problemløsning»?
   - Dette er problemløsning som i begrenset grad benytter seg av dekomponering, mønstergjenkjenning, og logikk. Det trenger ikke å benytte programmering. I følgende oppgaver er det naturlig at elevene benytter enkel algoritmisk problemløsning:
     - Finn de første tallene i følgen definert ved $a_1 = 3$ og $a_n = a_{n-1} + 4$.
     - Hvordan fortsetter følgen $-1, 0, 2, 5, 9$?
     - Lag et program som finner summen av alle heltall fra 1 til 100.
   - Mange korte og omfattende programmeringsoppgaver krever enkel algoritmisk problemløsning.
   - Det kan også være enkel algoritmisk problemløsning hvis man analyserer en enkel algoritme for å finne ut hva den gjør eller hvordan den fungerer.

5. Hva er «avansert algoritmisk problemløsning»?
   - Dette er arbeid som krever stor grad av dekomponering, logikk og mønstergjenkjenning, og der løsningen kan automatiseres, gjerne ved iterasjon. Det trenger ikke å benytte programmering. I følgende oppgaver er det naturlig at elevene benytter avansert algoritmisk problemløsning:
     - Eksemplene på *III.9 Hva er omfattende programmeringsoppgaver* på neste side*?*
     - En robot skal kjøre fra øverste venstre hjørne til nederste høyre hjørne i 3x3-rutenettet nedenfor. Roboten må følge strekene og kan bare kjøre til høyre og nedover a) Hvor mange forskjellige ruter kan den kjøre? b) Hvor mange forskjellige ruter har roboten i et 5x5-nett?

       
   - Ikke all problemløsning er algoritmisk problemløsning. Her er eksempler på problemer som vanligvis ikke løses algoritmisk, fordi de ikke krever dekomponering og iterasjon i tilstrekkelig grad:
     - Et kvadrat med sidelengde 2 er omsluttet av en sirkel slik at hjørnene i kvadratet tangerer sirkelen. Hva er sirkelens areal?
     - Er summen av tre påfølgende heltall alltid delelig på seks? Hvorfor (ikke)?
   - Ikke alle omfattende programmeringsoppgaver krever avansert algoritmisk problemløsning.
   - Det kan være avansert algoritmisk problemløsning hvis elevene analyserer en avansert algoritme dypt.

III. Avklaring av begreper – fortsatt –

6. Hva er algoritmiske arbeidsmåter?
    - UDIRs algoritmiske arbeidsmåtene er *fikle, skape, feilsøke, holde ut,* og *samarbeide.*
    - *Fikle* er å teste ut løsninger eller prøve forskjellige fremgangsmåter og trenger ikke å inneholde analyse. Et eksempel er å foreslå ulike formler for et figurtall og teste om de stemmer.
    - *Skape* er å designe eller lage et produkt. Det kan for eksempel være en kode, en plakat, en modell eller noe annet. Et eksempel er å lage et dataprogram.
    - *Feilsøke* er å oppdage, analysere og rette feil.
    - *Benytte de algoritmiske arbeidsmåtene i utstrakt grad* er at alle arbeidsmåtene er til stede over tid. Et eksempel er hvis elevene får oppgaven å lage en figur som rommer 7 dL med saks, teip og en papp-plate. De må skape, fikle, teste om de har rett, holde ut over tid og kanskje samarbeide.

7. Hva er «syntaksen til et programmeringsspråk»?
    - Dette er hvordan man uttrykker betingelser, løkker, variabler, funksjoner og liknende i et programmeringsspråk. Fork eksempel hvordan man uttrykker *hvis*-setninger i Python eller Excel.

8. Hva er «korte programmeringsoppgaver»?
    - Dette er oppgaver som går forbi det å lære syntaks, men som for alderstrinnet kun krever enkel bruk av syntaktiske elementer (variabler, løkker, betingelser). Dette kan for eksempel være
        - En oppgave i Excel som krever litt regning og en hvis-betingelse
        - Ta inn verdier for a, b og c og print $a^2 + b^2$ og c^2 til skjerm
        - Lag et program som tar inn et tall fra brukeren og returnerer om det er et odde- eller partall.
    - Det kan også være analyse av korte programmer.

9. Hva er «omfattende programmeringsoppgaver»?
    - Dette er oppgaver som krever at man kombinerer forskjellige syntaktiske elementer (variabler, løkker, betingelser) på en ikke helt enkel måte for å løse oppgaven.
        - Lag et program som spør brukeren om et heltall og regner ut hvor mange divisorer det har.
        - Lag et program som simulerer sannsynlighetsfordelingen til summen av tre sekssifrede terninger.
        - Hva er det hundrede tallet i rekken -1,0,2,5,9?
    - Det kan også være analyse av omfattende programmeringskode.
    - Å avgjøre hva som er *korte* og *omfattende* programmeringsoppgaver krever skjønn.

IV. Om skåringen

1. Generelt
   - Hvis vi ikke hører eller ser hva elevene gjør, skåres segmentet vanligvis som 1.
   - Hvis vi ikke hører hva elevene sier, men hører lærerens instruks og ser at en eller flere elever gjør det, skåres det som 1–4, avhengig av hvor tydelig aktiviteten handler om algoritmisk tenkning eller programmering.
   - Det er nok at vi observerer en elev som deltar i en aktivitet for å gi skår 2–4.
   - Ved skår 1 på alle kategorier kan vi ikke si at undervisningen omfatter algoritmisk tenkning eller programmering.
   - Ved skår 2–4 på en eller flere av kategoriene kan vi si at undervisningen omfatter algoritmisk tenkning eller programmering.
   - Hvis lærer forklarer en hjemmelekse som er relevant for elevenes algoritmiske tenkning eller programmering, skårer segmentet vanligvis som 2 på «undervisning», men det gir ikke utslag på elevkategoriene (skår 1).

2. Informasjon fra andre segmenter
   - Som hovedregel bruker vi ikke informasjon fra et segment for å skåre et annet segment.
   - Vi bruker informasjon kronologisk, det vil si at vi koder fra starten av timen. Informasjon fra tidligere segmenter ligger dermed til grunn for kodingen i et senere segment.
   - Et unntak er at vi kan bruke informasjon som fremgår i et senere segment om hvilke oppgaver elevene jobber med i skåringen av tidligere segmenter innenfor samme time.

3. Lærerens undervisning:
   - Hvis læreren underviser en algoritme eller gir oppgaver for å trene på en algoritme, vil segmentet vanligvis få minst skår 2.
   - Hvis læreren underviser om syntaktiske elementer i et programmeringsspråk (variabler, betingelser, løkker), vil segmentet vanligvis få minst skår 2.
   - Hvis læreren oppfordrer elevene til å «se etter mønstre», «bare prøv dere frem, det gjør ikke noe om det ikke går», «hvis svaret er feil må dere prøve å finne ut hvor det gikk galt» eller annet som kan knyttes til algoritmiske arbeidsmåter, vil segmentet vanligvis få minst skår 2.
   - Hvis læreren gir oppgaver som krever algoritmisk problemløsning, vil segmentet vanligvis få minst skår 2.

4. Elevenes deltakelse i læringsaktiviteter:
   - Læringsaktivitet inkluderer alle aktiviteter læreren legger opp til som er en del av undervisningen.
   - Vi deler inn elevenes deltakelse i fire kategorier: algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering.
   - *Algoritmisk problemløsning*: Sentrale skiller mellom nivåene er om algoritmer ikke er til stede (nivå 1), om kjente algoritmer benyttes (nivå 2), om algoritmisk tenkning benyttes som en problemløsningsmetode (nivå 3 og 4), eller om elevene analyserer hvordan algoritmer fungerer (nivå 3 og 4).
   - *Algoritmiske arbeidsmåter (fikle, skape, feilsøke, holde ut, samarbeide):* Man kan skåre høyt på algoritmiske arbeidsmåter uavhengig av nivået på algoritmisk problemløsning. For eksempel kan oppgaven «klipp ut og lim sammen pappbiter slik at du får en lukket beholder med volum 8 L» skåre høyt på algoritmiske arbeidsmåter selv om den ikke inneholder algoritmisk problemløsning. Det er ikke krav om at alle de algoritmiske arbeidsmåtene er til stede for å få høy skår, men fikle, skape eller feilsøke må være til stede.
     Benytter en eller flere elever seg av algoritmiske arbeidsmåter? (minst skår 2)
     Benytter en eller flere elever seg av i betydelig grad av algoritmiske arbeidsmåter? (skår 3)
     Benytter de fleste elevene seg i noen grad av algoritmiske arbeidsmåter? (skår 3)
     Er segmentet preget av at de fleste elevene benytter algoritmiske arbeidsmåter i betydelig grad? (skår 4)
   - *Programmering:* Sentrale skiller mellom nivåene er om elevene øver på programmeringssyntaks (nivå 2), eller om de skriver programmer eller analyserer virkemåten til et program (nivå 3 og 4). Forskjellen mellom en kort og en omfattende programmeringsoppgave er om flere syntaktiske elementer må kombineres for å løse oppgaven.

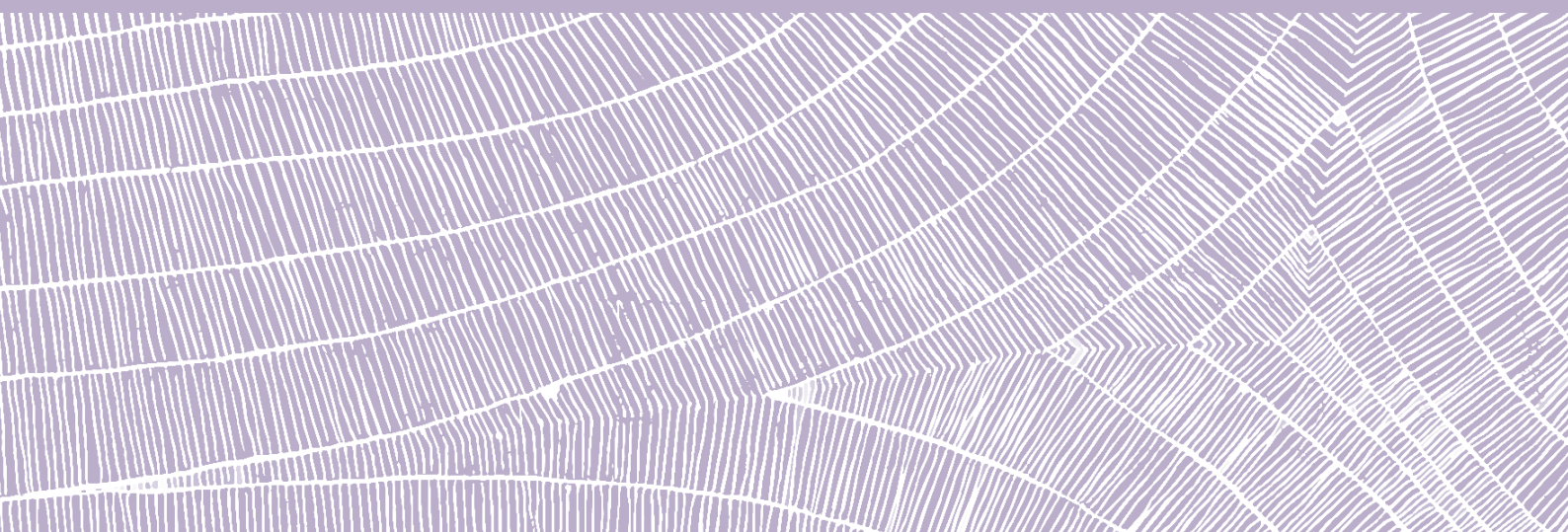
2.7 Oppsummering

Oppsummert handler denne delen om algoritmisk tenkning og programmering i matematikkfagene innenfor forskningslitteratur og i læreplanverket LK20. Vi har både identifisert kjennetegn ved og utfordringer knyttet til slike undervisningspraksiser i forskningslitteraturen. Basert på denne gjennomgangen forklarer vi hvordan vi har anvendt beskrivelsene i litteraturen og i læreplanverket i utviklingen av *EDUCATEs protokoll for observasjon av algoritmisk tenkning og programmering* (versjon 1.0).

I det følgende presenterer vi EDUCATEs forskningsdesign for å forstå algoritmisk tenkning og programmering i klasserommet.

DEL 3

Forskningsdesign for å forstå algoritmisk tenkning og programmering i klasserommet



3 Forskningsdesign for å forstå algoritmisk tenkning og programmering i klasserommet

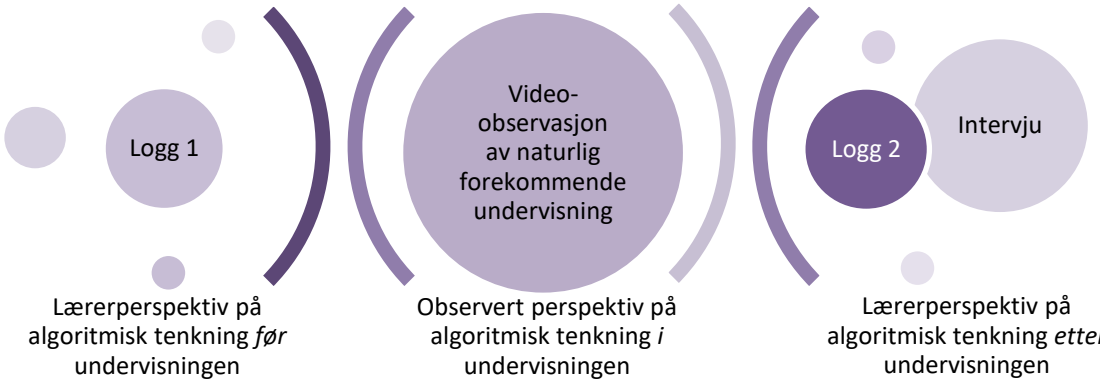
I Del 3 gjør vi rede for forskningsdesignet som EDUCATE har brukt for å studere algoritmisk tenkning i klasserommet. Vi redegjør først for forskningsdesignet (3.1) og utvalget (3.2). Deretter følger en beskrivelse av hvordan vi gjennomfører videoobservasjoner av undervisningen (3.3) og hvordan vi samler inn læreres perspektiver på undervisningen (3.4). Deretter gir vi oversikt over datamaterialet som er samlet inn på 8. trinn, 10. trinn og Vg1 skoleårene 2021–24 (3.5) og hvordan vi har analysert de empiriske dataene (3.6), fulgt av en diskusjon av forskningsetiske hensyn (3.7). Til slutt gir vi en kort oppsummering (3.8).

3.1 Mixed methods forskningsdesign

EDUCATE bruker et *mixed methods* casedesign (Brevik m.fl., 2023; Brevik & Mathé, 2021; Tashakkori m.fl., 2020; Yin, 2009). Det betyr at vi integrerer kvantitativ og kvantitativ informasjon for å få rik informasjon om hvordan algoritmisk tenkning blir undervist og forstått av matematikklærere. Dette styrker muligheten for å kunne overføre funn fra denne forskningen til praksiser ved andre skoler (Maxwell, 2021; Yin, 2009). EDUCATEs forskningsdesign skiller seg fra intervensjonsstudier ved at vi samler inn videoopptak fra naturlig forekommende undervisning (Brevik m.fl., 2023a), noe som styrker studiens økologiske validitet.

Læreplanverket LK20 er preget av metodefrihet og høy grad av handlingsrom for lærere. Dette innebærer at det i stor grad er opp til lærerne å bestemme hvordan de vil jobbe med algoritmisk tenkning i matematikkfagene. For å forstå hvordan dette blir gjort i klasserommet, utgjorde filming av undervisningen kjernen i forskningsdesignet og vi ba lærerne føre en logg umiddelbart før og etter hver filmede time. I tillegg intervjuet vi dem etter at filmingen i den enkelte klassen var ferdig, for å få deres perspektiver på loggene og undervisningen og gi dem anledning til å utdype temaet. Se Figur 3.1.

Figur 3.1 EDUCATEs forskningsdesign for algoritmisk tenkning og programmering



Figur 3.1 viser hvordan EDUCATE integrerer informasjon fra de videofilmede timene med lærerlogger og intervjuer. Ved å dra nytte av forskjellige metodiske innfallsvinkler, danner vi et bredere og mer detaljert bilde av hvordan algoritmisk tenkning blir forstått og undervist i fagene enn med færre datakilder. Det gir oss samtidig mulighet for å sammenligne systematisk på tvers av skoler, fag, lærere, klasserom og timer. Boks 4 utdyper EDUCATEs forskningsdesign for å forstå algoritmisk tenkning i matematikk.

Boks 4 EDUCATEs forskningsdesign for algoritmisk tenkning og programmering

- EDUCATE har samlet systematiske videodata om algoritmisk tenkning i matematikkfagene på 8. trinn, 10. trinn og Vg1, både på studiespesialiserende og yrkesfaglige programmer (1P, 1T og 1P-Y).
- Disse klasseromsdataene er analysert med EDUCATEs observasjonsprotokoll for algoritmisk tenkning. Dette bidrar med en felles analytisk linse slik at forskerne i EDUCATE ser etter det samme når de ser etter algoritmisk tenkning i ulike fag og klasserom.
- EDUCATE ber lærere loggføre om de har planlagt å inkludere algoritmisk tenkning i undervisningen umiddelbart før timen og om de faktisk inkluderte algoritmisk tekning umiddelbart etter timen. På denne måten får vi et situasjonsbilde som er tett på undervisningen.
- EDUCATE intervjuer lærerne om deres syn på algoritmisk tenkning og programmering etter å ha filmet. Dette bidrar med læreres refleksjoner algoritmisk tenkning i klasserommet.
- EDUCATE sammenligner videoobservasjoner fra klasserommet med lærernes selvrapportering. Dermed får vi et rikt materiale for å forstå hvordan algoritmisk tenkning i matematikkfagene på 8. trinn, 10. trinn og Vg1 kan forstås fra ulike perspektiver.

3.2 Utvalg og datamateriale

Til Rapport 5 analyserte vi data fra alle de sju deltakende skolene i EDUCATE-prosjektet, fire på ungdomstrinnet og tre videregående skoler. Av disse var én skole studieforberedende, én var yrkesfaglig og én var kombinert studieforberedende og yrkesfaglig. Skolene ligger i tre forskjellige fylker i ulike deler av Norge og inkluderer både urbane skoler og forstadsskoler. Vi samlet inn data fra og med høsten 2021 til og med våren 2024 for å følge utviklingen i undervisningen etter at LK20 ble innført. I denne rapporten bruker vi alle data fra alle matematikklasserom. Vi samlet inn data på 8. trinn og Vg1 skoleåret 2021–22 og 10. trinn i skoleåret 2023–24. Vi har fulgt de samme klassene fra 8. til 10. trinn.

Tabell 3.1 viser at det på grunn av tidspunktene for filmingen er noen forskjeller mellom elevkullene. Elevene på 8. og 10. trinn er født i 2008, og elevene på Vg1 er født i 2005. Historisk sett er dette to helt spesielle elevkull. Deres første møte med henholdsvis ungdomsskolen og videregående skole var etter to år med varierende grad av nedstenging som følge av koronapandemien. De hadde derfor en annerledes overgang til ungdomsskolen og videregående skole enn elevkullene før dem. Dette diskuterte vi i Rapport 2 (Brevik m.fl., 2023), der vi så på livsmestring da 2008-kullet gikk på 8. trinn, i Rapport 3 (Brevik m.fl., 2024), der vi så på utforsking for 2005-kullet da de gikk i Vg1 og vg2, og i Rapport 4 (Gudmundsdottir m.fl., 2024), der vi så nærmere på begge kullene da de gikk på henholdsvis 10. trinn og vg3, etter at de hadde vært tilbake på skolen i to år.

I denne rapporten ser vi på begge kullene skoleåret 2021–22, det andre året LK20 var gjeldende, på henholdsvis 8. trinn og Vg1. På grunn av koronapandemien kan man tenke seg at lærerne hadde annet å stri med enn å iverksette nyvinningene i læreplanen, som algoritmisk tenkning jo er. Deretter filmet vi kullet født i 2008 på 10. trinn skoleåret 2023–24, det fjerde året LK20 var gjeldende og det andre nesten pandemifrie skoleåret. Det er altså flere grunner til at vi kan forvente mer algoritmisk tenkning og programmering på 10. trinn enn på 8. trinn og Vg1.

Tabell 3.1 Erfaring med LK20 blant EDUCATEs deltakere

	8. og 10. trinn (født 2008)	**Vg1 (født 2005)**
Skoleåret 2021–22	LK20 ble innført på 8. trinn høsten 2020. Elevene på 8. trinn i vårt materiale fulgte LK06 på 7. trinn og var derfor nye i LK20. Noen lærere var også nye i LK20, andre hadde brukt LK20 ett år.	LK20 ble innført på Vg1 høsten 2020. Elevene i vårt materiale fulgte LK06 på 10. trinn og var derfor nye i LK20. Noen lærere var også nye i LK20, andre hadde brukt LK20 ett år.
Skoleåret 2023–24	Elevene på 10. trinn i vårt materiale har dermed fulgt LK20 gjennom hele ungdomstrinnet (2021–24).	

Tabell 3.2 viser at vi de to skoleårene samlet data ved sju skoler (fire på ungdomstrinnet og tre i videregående skole). Totalt deltok 19 matematikklærere, og vi intervjuet 17 av dem. Lærerne underviste i fire ulike matematikkfag i til sammen i 29 klasser fordelt på 8. trinn, 10. trinn og Vg1. Vi filmet totalt 117 timer. Lærerne fylte ut logger både før (logg 1) og etter (logg 2) undervisningen, henholdsvis 95 og 97 ganger (82 % av timene).

Tabell 3.2 EDUCATEs empiriske data i matematikk (skoleårene 2021–24)*

Trinn	Skoleår	Skoler	Lærere	Intervju	Fag	Klasser	Filmede timer	Lærer-logg 1	Lærer-logg 2
Vg1 (SF)	2021–22	A	4	3	1T	4	16	16	14
					1P	4	16	16	10
		B	1*	1*	1P	1	4	4	4
Vg1 (YF)	2021–22	B	3 (1)*	1 (1)*	1P-Y	3	12	8	12
		C	2	2	1P-Y	2	8	4	4
8. trinn	2021–22	D, E, F, G	6	6	Mate-matikk	9	36	26	29
10. trinn	2023–24	D, E, F	6(4)*	4	Mate-matikk	6	25	21	24
3	**2**	**7**	**19**	**17**	**4**	**29**	**117**	**95**	**97**

**Skoler*: En av skolene trakk seg fra deltakelse på 10. trinn. *Lærere/intervjuer*: Blant de 19 deltakende lærerne, ble én filmet i to klasser (SF og YF) og intervjuet om begge klassene i samme intervju. På 10. trinn var fire av lærerne nye, og to ble intervjuet. To lærere var de samme som på 8. trinn, og ble intervjuet begge år.

Det er omtrent like mange deltagende lærere fra ungdomstrinnet og fra videregående skole, og i videregående skole er studieforberedende og seks yrkesfaglige studieprogram representert. Ingen av de yrkesfaglige klassene vi filmet underviste i 1T-Y. Alle lærerne som deltok, hadde lærerutdanning og alle aldersgrupper var representert. De fleste var erfarne lærerne, men fire av dem hadde mindre enn fem års erfaring som lærer. Se Tabell 3.3.

Tabell 3.3 EDUCATEs deltakende matematikklærere (skoleårene 2021–24)

Trinn	**Alder**				**Lærerutdanning**			**År som lærer**			
	20–29 år	30–39 år	40–49 år	50–59 år	4-årig LU	5-årig LU	PPU[1]	0-5 år	6-10 år	11-20 år	21-30 år
8.	1	2	1	2	2	0	4	1	0	2	3
10.[2]	0	0	3	2	1	1	3	0	0	3	2
Vg1	3	4	1	1	1	3	5	3	3	3	0
Total	20				20			20			

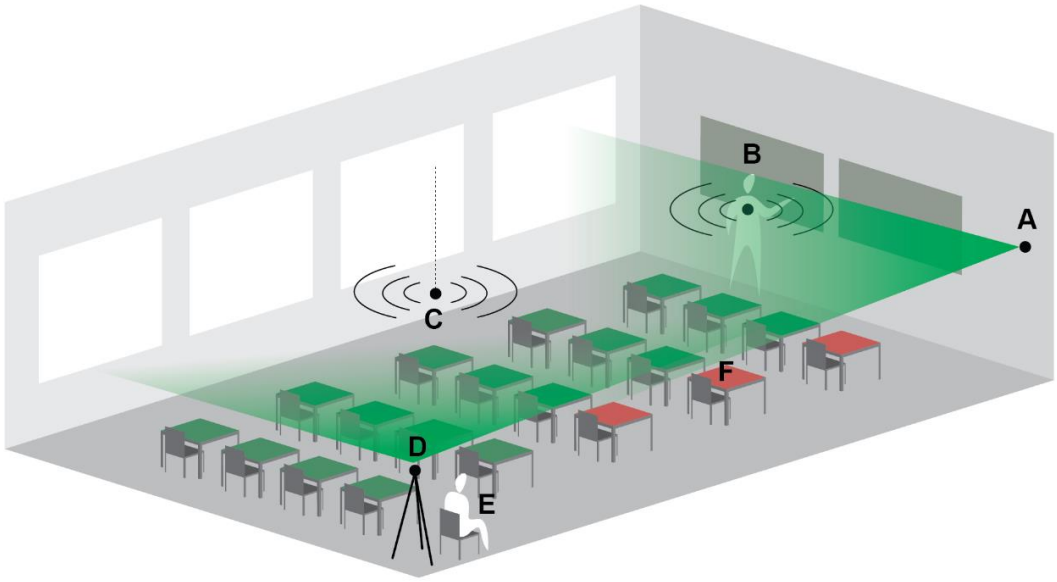
LU = lærerutdanning. PPU = praktisk-pedagogisk utdanning. [1] I denne kategorien inngår en lærer som var under utdanning. [2] Blant 10. trinnslærerne inngår fire nye lærere og to lærere som ble filmet på 8. trinn. En lærer på 10. trinn fylte ikke ut bakgrunnsinformasjon.

3.3 Videoopptak av digital kompetanse i klasserommet

Videoobservasjon er kjernen i EDUCATEs forskningsdesign (se Figur 3.1). Vi filmet omtrent én ukes matematikkundervisning i hver klasse hvert år, som stort sett tilsvarte fire påfølgende undervisningstimer. Det argumenteres ofte med at fire timer er tilstrekkelig for å få et godt bilde av en lærers undervisningskvalitet (Ball & Hill, 2009; Bill & Melinda Gates Foundation, 2012; Klette, 2009; Klette et al., 2017). Samtidig er det viktig å ikke overgeneralisere og tenke at fire timer også er tilstrekkelig for å beskrive hver enkelt lærers undervisning av algoritmisk tenkning og programmering. Årsaken er at dersom en lærer underviser algoritmisk tenkning og programmering som et eget tema lagt til en bestemt tid på året kan 100 % av timene vi filmer omhandle programmering hvis vi tilfeldigvis treffer på de timene, selv om læreren egentlig underviser programmering bare 5 % av tiden i løpet av et skoleår. Selv om dataene ikke nødvendigvis er representative for hver enkelt lærers undervisning, kan det gi et godt totalbilde av undervisningen i algoritmisk tenkning og programmering siden vi har flere lærere som blir filmet på forskjellige tidspunkt i løpet av skoleåret og på forskjellige trinn.

Det er likevel viktig å være klar over noen ulemper med videoobservasjon: Tilstedeværelsen av kameraer, lydutstyr og observatører kan påvirke dynamikken i klasserommet, for eksempel ved at lærere er bevisst på at vi filmer dem visse uker, og derfor planlegger annerledes enn om vi ikke filmet (Brevik m.fl., 2023a). Likevel tyder mye på at slike effekter er relativt små og kortvarige (Lahn & Klette, 2023). En annen og intensjonell påvirkning, er mulighetene for ikke-deltakende elever til å delta i undervisningen, selv om de ikke ønsker å fanges på video (Brevik m.fl., 2023a). Dette tar vi hensyn til ved å skape blindsoner i klasserommet (se Figur 3.2).

Figur 3.2 EDUCATEs videodesign med kameravinkler og blindsoner



A illustrerer kameraet som fanger lærerperspektivet. B illustrerer lærermikrofonen. C illustrerer helklassemikrofonen. D illustrerer kameraet som fanger elevperspektivet. E illustrer observatøren som filmer undervisningen og ivaretar etiske og tekniske hensyn. F illustrer blindsonen (Illustrert av Enger og Aashamar; se Aashamar m.fl., 2024; Brevik m.fl., 2024; Gudmundsdottir m.fl., 2024).

3.4 Lærernes selvrapporterte perspektiv: logger og intervju

For å ivareta lærerperspektivet, ba vi lærerne føre logger umiddelbart før og etter de filmede timene. I forkant av timen rapporterte de om hvorvidt de planla å arbeide med algoritmisk tenkning og programmering (logg 1), og etter timen rapporterte de i hvilken grad de hadde inkludert det i timen (logg 2). Etter den siste filmede timen i hver klasse, ble lærerne intervjuet. Med bakgrunn i loggene ba vi lærerne beskrive deres undervisningspraksiser og deres forståelse av algoritmisk tenkning generelt, i matematikk og i LK20. EDUCATE benyttet individuelle dybdeintervjuer på ca. én time. På denne måten fikk vi innsikt i lærernes refleksjoner, oppfatninger og fortellinger om egen planlegging, undervisning og forklaringer på eventuelle endringer som ble gjort underveis i timene. Intervjuguiden inneholdt spørsmål til alle temaene som EDUCATE evaluerer, blant annet følgende spørsmål om algoritmisk tenkning:

Boks 5 EDUCATEs intervjuguide om algoritmisk tenkning

EDUCATE-teamet utarbeidet følgende spørsmål i intervjuguiden til matematikklærere. Målet med denne delen var å fokusere på lærerens undervisning og erfaring med algoritmisk tenkning.

1. I loggene du fylte ut, skrev du at du (ikke) brukte/tematiserte algoritmisk tenkning i matte. Kan du fortelle oss litt om dette?
2. Er dette [dvs. bruk/ikke bruk] vanlig/typisk for din undervisning?
   [Første prioritet er å snakke om timene vi filmet]
3. Hva legger du i begrepet *algoritmisk tenkning* – hva betyr det for deg i din undervisning?
   [Har din forståelse endret seg det siste året?]
4. Hvordan vektlegger du *algoritmisk tenkning* i undervisningen din?
   [Hvordan vil vi kunne se om du jobber med *algoritmisk tenkning* i undervisningen din?]
5. Hvordan oppfatter du elevenes (kunnskap om) *algoritmiske tenkning*? Kan du konkretisere – gi eksempler?
6. Har vektleggingen av *algoritmisk tenkning* i Fagfornyelsen endret din undervisning i matte [det siste året]?
7. Hvilke muligheter og utfordringer ser du med å knytte *algoritmisk tenkning* til ditt/dine fag?
   [Hvordan er det for deg å jobbe med algoritmisk tenkning i undervisningen nå, sammenlignet med for et år siden?]
8. [Hvordan vurderer du din *egen kompetanse* for å jobbe med algoritmisk tenkning i undervisningen?]

På 10. trinn intervjuet vi noen av de samme lærerne som på 8. trinn. Vi vektla da å få frem om deres syn på algoritmisk tenkning hadde endret seg siden 8. trinn. Dette er indikert med klammer over [].

3.5 Kvantitativ analyse av den videofilmede undervisningen

Vi analyserer filmene av undervisningen i to trinn; først med en observasjonsprotokoll, deretter med en kvalitativ analyse basert på resultatene i observasjonsprotokollen. Observasjonsprotokoller er spesielt egnet for å kartlegge undervisningspraksiser av større mengder videodata på en systematisk måte gjennom koding. Dette styrker mulighetene for å sammenligne undervisning på tvers av fag, klasser og kontekster (Klette, 2023; Klette m.fl., 2017; Praetorius m.fl., 2014). EDUCATE har utviklet en *protokoll for observasjon av algoritmisk tenkning og programmering* gjengitt i Boks 3. Vi fulgte faste prosedyrer for koding av videodata i EDUCATE.

Boks 6 EDUCATEs koding av videodata

EDUCATE-teamet har utarbeidet følgende rutiner for koding av videomaterialet i denne rapporten.

- Hver time som kodes legges inn i analyseprogrammet *InterAct,* med bruk av EDUCATEs mal.
- Hver undervisningstime deles inn i 15-minutters segmenter. Dersom det er igjen mindre enn 15 minutter igjen på slutten av timen, følger vi denne inndelingen: mindre enn 7,5 minutter legges inn i foregående segment, mens mer enn 7,5 minutter utgjør et nytt segment. Det vil si at siste segment i en time kan være mellom 7,5 minutter og 22,5 minutter langt.
- Hvert segment kodes ved først å se hele segmentet (15 minutter) og deretter kode segmentet før vi går til neste segment.
- Det gis én skår for hver av de fire kategoriene innenfor algoritmisk tenkning og programmering for hvert segment.
- Vi velger 20 % av timene i hvert fag på hvert trinn til dobbelkoding. Vi velger hovedsakelig 1. time fra ulike lærere for å få størst mulig spredning på timene som dobbelkodes. Målet er å oppnå minst 80 % enighet mellom de som koder samme materiale.

For å komme nært på undervisningen, ble alle videofilmede timer i EDUCATE delt inn i 15-minutters segmenter, se Tabell 3.4.

Tabell 3.4 Antall matematikktimer og segmenter analysert

Trinn	Timer	Segmenter
8	33	109
10	25	92
Vg1	56	142
Totalt	**117**	**343**

Kodingsprosessen besto først av å vurdere om et segment inneholder algoritmisk tenkning eller programmering. Om vi ikke observerte undervisning som inneholdt slike praksiser, ble segmentet kodet med skår (1): Ingen observasjon av algoritmisk tenkning i undervisningen. Hvis vi derimot observerte at segmentet inneholdt algoritmisk tenkning, spurte vi i hvilken grad innholdet stemte overens med følgende skår: (2) Tilstedeværelse av algoritmisk tenkning i undervisningen, (3) Eksplisitt arbeid med algoritmisk tenkning, eller (4) Aktualisert arbeid med algoritmisk tenkning. Vi kodet for fire kategorier: *lærerens undervisning* og elevenes arbeid med *algoritmisk problemløsning*, *algoritmiske arbeidsmåter* og *programmering*. Se Del 2 for en detaljert beskrivelse av protokollen og skårene.

3.6 Kvalitativ næranalyse av den videofilmede undervisningen

Etter å ha kodet alle segmenter med observasjonsprotokollen, gjorde vi en kvalitativ næranalyse for å beskrive hva som kjennetegnet undervisningen som ga de ulike skårene. Kodene fra observasjonsprotokollen forelå i et Excel-dokument, og vi benyttet Pivot-tabeller for å få oversikt over skårene på hvert segment. Først valgte vi ut alle undervisningstimene med minst ett segment på skår 3 eller 4 på minst én av de fire kategoriene, for å kunne beskrive eksplisitte og aktualiserte eksempler på algoritmisk tenkning og programmering i undervisningen.

Vi så gjennom videoene av alle disse undervisningstimene og tok korte notater i fem kolonner i Excel-tabellen, én kolonne for en generell beskrivelse av timen og de andre kolonnene for notater som var relevante for hver av de fire kategoriene. Mange av notatene tok utgangspunkt i de teoretiske begrepene fra protokollen (slik som *fikle*), men vi noterte også annet som vi så i dataene (slik som *elevene får ikke kjørt koden*). Vi valgte også ut segmenter med skår 2 og tok liknende notater, og så nøye på de delene av undervisningen som var relevant for algoritmisk tenkning. Eksempler på notater er vist i Tabell 3..

Tabell 3.5 Eksempel på notater for næranalyse av den filmede matematikkundervisningen

Variabel	**Eksempelnotat**
Beskrivelse av timen	Ferdiglaget tutorial på trinket.io. Turtle-geometri, bare følge gitte instruksjoner.
Lærerens undervisning	Nei
Algoritmisk problemløsning	Ingen
Algoritmiske arbeidsmåter	Utstrakt fikling for å få koden til å kjøre, noe samarbeid og feilsøking, men mange gir opp pga. får ikke kjørt koden.
Programmering	Tutorial som viser Python-syntaks hele timen. Ingen virkelige programmeringsoppgaver å snakke om.

Etter å ha tatt notater fra alle disse timene benyttet vi filtre til å sammenstille notatene på forskjellige måter. For eksempel sammenliknet vi notatene på alle segmentene med skår 4 for algoritmisk problemløsning for å finne hva som var typisk for disse, deretter sammenlignet vi med segmentene med skår 3. Vi gjorde tilsvarende for alle fire kategorier, og sammenlignet både innad og på tvers av trinn. Dette ga mulighet for å sammenligne (a) ungdomstrinn med videregående skole, (b) yrkesfag med studieforberedende program, og (c) undervisning på 8. og 10. trinn blant de samme lærerne vi fulgte fra 2021–22 til 2023–24. Til slutt valgte vi klasseromsepisoder som illustrerte funnene og transkriberte dem som utdrag i denne rapporten.

3.7 Kvantitativ og kvalitativ analyse av lærerlogger og -intervjuer

Lærerloggene ble levert inn elektronisk. Vi regnet ut hvordan svarene fordelte seg på de ulike matematikkfagene, fordelt på logg 1 og 2. Alle intervjuene ble transkribert med autotekst.uio.no, en tjeneste for sikker automatisk transkribering tilbudt av Universitetet i Oslo. De automatiske transkripsjonene ble samtidig renset for personidentifiserende informasjon, som navn på skoler, lærere og elever, og ble erstattet med koder og pseudonymer. Til slutt ble transkripsjonene kvalitetssikret gjennom å høre gjennom intervjuet og sammenligne med transkripsjonene for å sjekke at de var nøyaktige.

Vi analyserte lærerintervjuene med spesiell vekt på følgende tre punkter:

1. Hvordan forsto lærerne begrepet *algoritmisk tenkning*?
2. Hvordan underviste lærerne i algoritmisk tenkning og programmering?
3. Hvilke utfordringer og muligheter så lærerne med innføringen av algoritmisk tenkning og programmering i LK20?

Transkripsjonene av lærerintervjuene ble analysert med en programvare for kvalitativ analyse, HyperRESEARCH (Researchware Inc., 2023). Da vi analyserte transkripsjonene søkte vi etter ord som *algoritm* og *program* for å finne alle steder i transkripsjonene der ord som *algoritme*, *algoritmisk*, *programmering* eller *program* ble nevnt. Vi kodet ord, setninger og større tekstutdrag i transkripsjonene med forskjellige koder, som for eksempel: *forstår algoritmisk tenkning som problemløsningsmetode*, *underviser mindre programmering enn planlagt* eller *uklart hva elevene skal lære*. Disse utdragene henvendte seg gjerne til et av de tre punktene over, men vi satte også andre koder når vi syntes lærerne sa noe interessant som ikke var dekket av disse punktene. Til slutt lagde vi overordnede temaer ved å slå sammen koder og gi dem beskrivende navn. Vi lagde temaer som var beskrivende for intervjuene og som besvarte forskningsspørsmålene. For å være systematiske, presise, og avdekke flest mulig tolkninger av lærernes utsagn, gjennomførte både Stover og Hatlevik analysen og diskusjonen om mønstre, og bidro dermed til økt reliabilitet og validitet i analysen.

3.8 Forskningsetiske hensyn

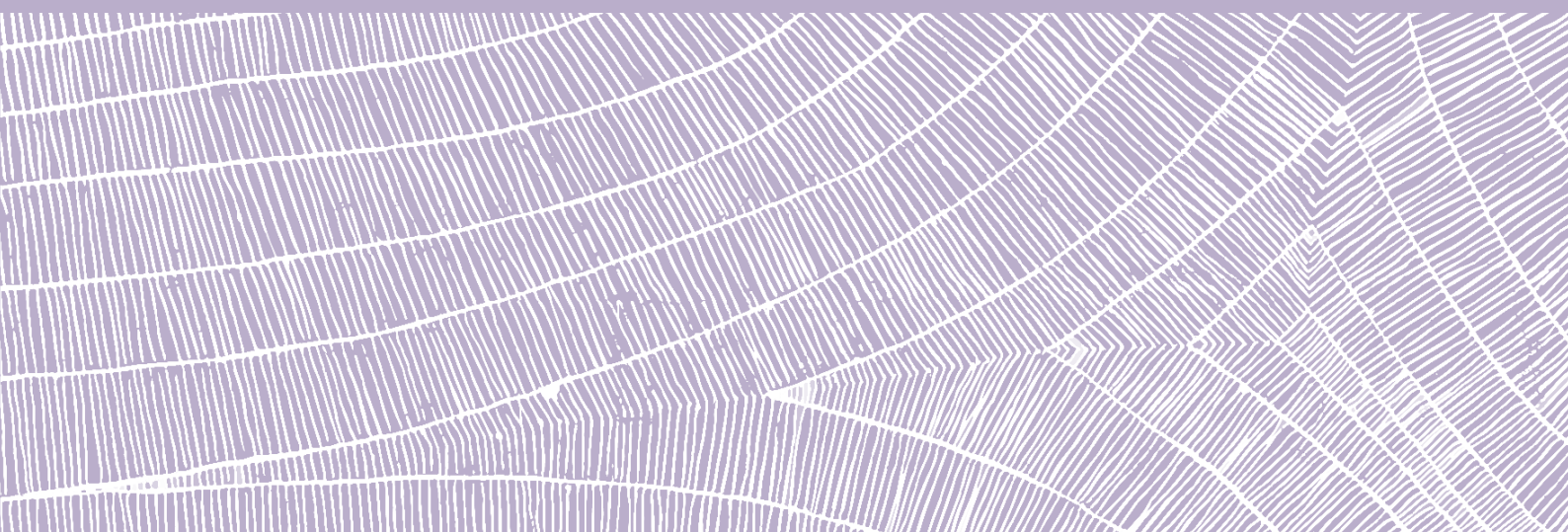
Datainnsamling, oppbevaring, tilgang og sletting av data i prosjektet er godkjent av NSD/Sikt (prosjekt 330104). Forskningsetiske hensyn i denne rapporten følger de overordnede retningslinjene som er tatt i EDUCATE-prosjektet (Brevik m.fl., 2023a). Disse er også i tråd med de forskningsetiske rettlingslinjene fra NESH (2021), inkludert frivillig, informert, utvetydig og dokumentert samtykke, anonymisering og pseudonymisering, hensynet til direkte og indirekte berørte personer, respekt for deltakernes selvforståelse samt prosjektets forhold til oppdragsgiver. Mer utfyllende beskrivelser av EDUCATEs forskningsetiske prosedyrer kan leses i Rapport 1 (Brevik m.fl., 2023a).

3.9 Oppsummering

I denne delen har vi gjort rede for vårt mixed methods forskningsdesign (Brevik m.fl., 2023a). Det observerte perspektivet på algoritmisk tenkning og programmering som vi får gjennom videodata, utfylles med lærernes selvrapporterte forståelser og oppfatninger av egen undervisning. Videre har vi i denne delen forklart fremgangsmåtene for analyser av video-, logg og intervjudata. Til slutt har vi henvist til forskningsetiske hensyn knyttet til prosjektet.

DEL 4

Algoritmisk tenkning og programmering i undervisningen



4 Algoritmisk tenkning og programmering i undervisningen

I denne delen presenterer vi funn fra algoritmisk tenkning og programmering i matematikkfagene på 8. trinn, 10. trinn og Vg1. I det følgende presenterer vi først et overblikk over kodingen med EDUCATEs observasjonsprotokoll på tvers av de 117 timene og 343 segmentene vi filmet (4.1). Disse analysene gir et overblikk over hvor ofte algoritmisk tenkning og programmering undervises og med hvilke egenskaper. Deretter beskriver vi hva som kjennetegner lærerens undervisning (4.2), og elevenes arbeid med algoritmisk problemløsning (4.3), algoritmiske arbeidsmåter (4.4) og programmering (4.5). Til slutt oppsummerer vi mønstre som er fremtredende når det jobbes med algoritmisk tenkning og programmering i matematikklasserommet (4.6).

Vi velger å presentere forholdsvis mange eksempler fordi det er få beskrivelser av hvordan algoritmisk tenkning og programmering ser ut i norske læreres klasserom. Vi håper eksemplene kan være nyttige for interesserte lærere og lærerutdannere. Vi har fokusert på eksempler med skår 3 eller 4 ved bruk av EDUCATEs observasjonsprotokoll og som enten er spesielt gode eller spesielt representative for våre data.

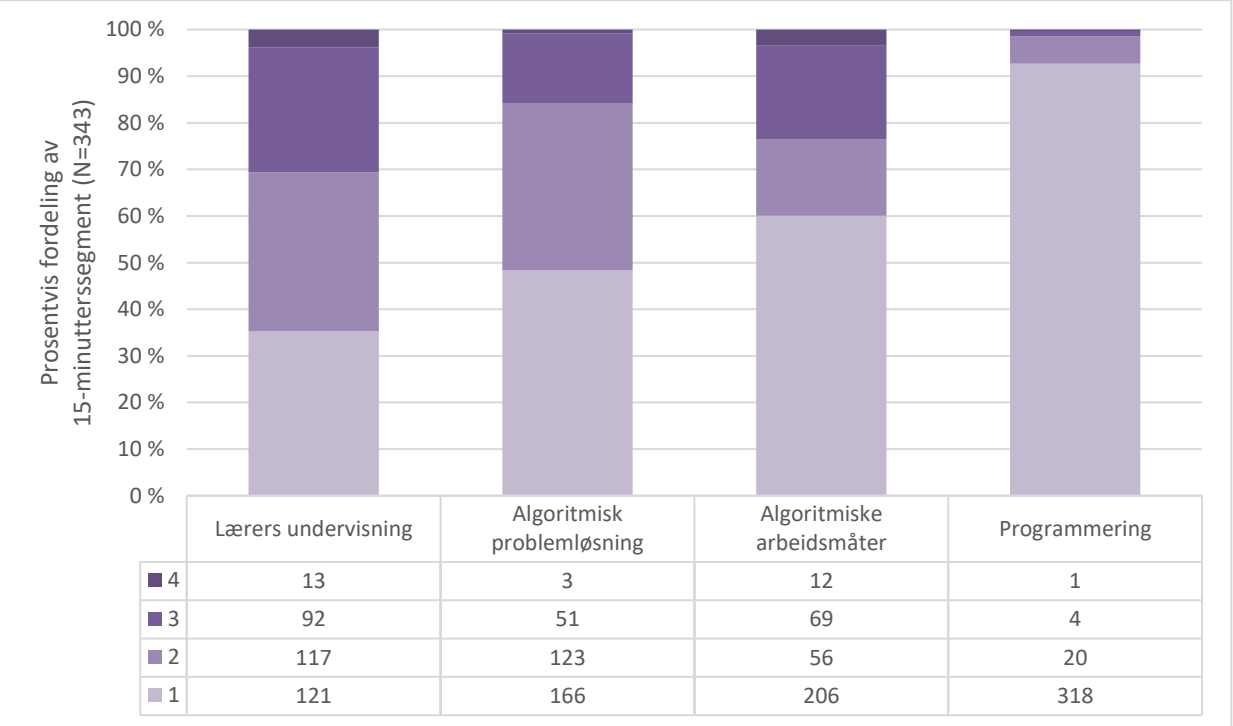
4.1 Resultater fra skåringen med observasjonsprotokollen

Hvert undervisningssegment ble kodet for fire kategorier: Lærerens undervisning og tilrettelegging for elevenes algoritmiske tenkning, samt elevenes arbeid med algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering. Figur 4.1 viser resultatene av kodingen med observasjonsmanualen for alle de 343 segmentene i matematikk på 8. trinn, 10. trinn og Vg1.

Figur 4.1 viser en prosentvis fordeling for hver av de fire kategoriene med en skår på 1–4. Når vi observerer algoritmisk tenkning i fagenes undervisning, får dette skår 2–4. Der vi ikke observerer algoritmisk tenkning får dette skår 1. Vi presenterer her matematikkfagene samlet, og vil i de påfølgende delene presentere separate tall for de ulike trinnene (8. trinn, 10. trinn og Vg1) og fag på Vg1 (1P, 1P-Y, 1T).

Først ser vi at i *lærernes undervisning* legger de til rette for å jobbe med algoritmisk tenkning i to tredeler av segmentene (65 %). Disse segmentene fordelte seg omtrent noe mer på skår 2 (34%), der lærerne la til rette for at elevene lærte en kjent algoritme for å løse en gitt type oppgave, syntaksen til et programmeringsspråk eller enkel bruk av algoritmiske arbeidsmåter, enn på skår 3 (27%), der lærerne la til rette for enkel algoritmisk problemløsning, forståelse av programmeringskonsepter, eller algoritmiske arbeidsmåter. Lærerne la sjelden til rette for løsning av problemer som krevde avansert algoritmisk problemløsning eller omfattende programmeringsarbeid (skår 4; 4%).

Figur 4.1 Prosentvis fordeling av skår for algoritmisk tenkning og programmering i 29 klasser



	Lærers undervisning	Algoritmisk problemløsning	Algoritmiske arbeidsmåter	Programmering
4	13	3	12	1
3	92	51	69	4
2	117	123	56	20
1	121	166	206	318

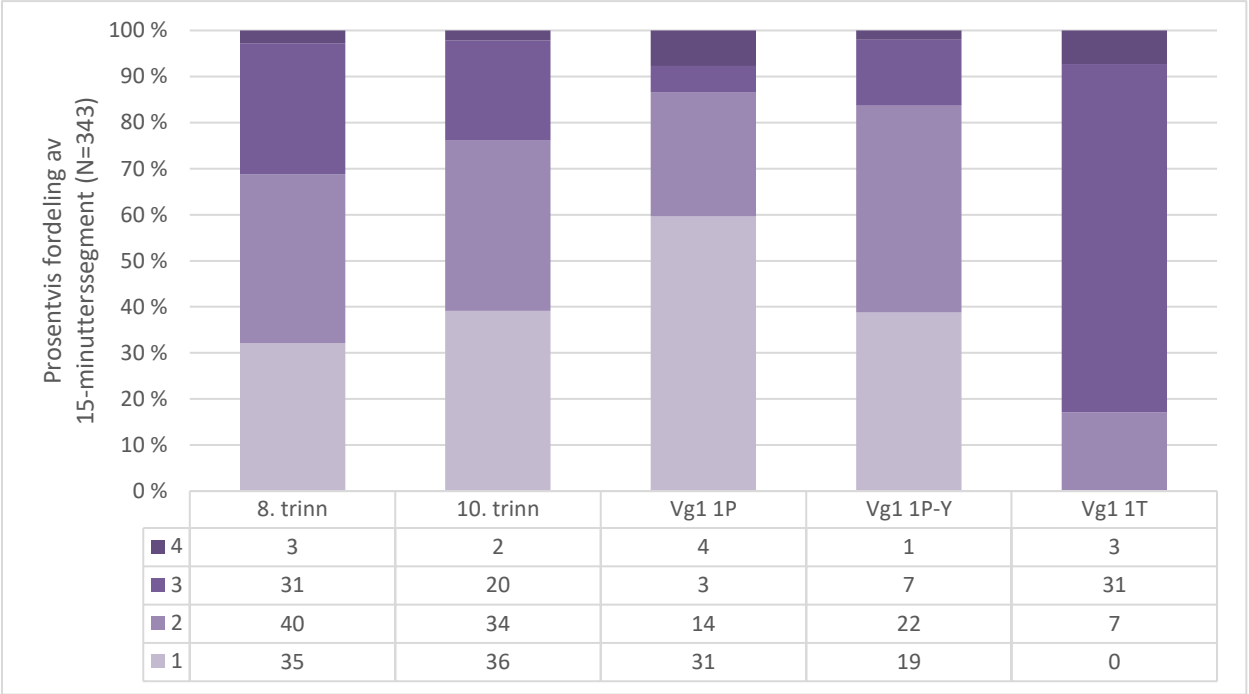
Videre ser vi at elevene jobbet med *algoritmisk problemløsning* i omtrent halvparten av segmentene (52 %). I litt over en tredel (36%) av disse segmentene observerte vi at elevene jobbet med en algoritme for å løse en gitt type oppgave (skår 2). I 15% av segmentene jobbet de med enkel algoritmisk problemløsning (skår 3), og i under 1 % av segmentene jobbet de med avansert algoritmisk tenkning (skår 4). Lærerne jobbet noe sjeldnere med *algoritmiske arbeidsmåter* (40%). Vi observerte i rundt en femtedel av segmentene at elevene fiklet alene eller sammen med andre (skår 2; 16 %) eller at de fiklet, skapte eller feilsøkte (skår 3; 20 %), og i 4 % av segmentene benyttet de algoritmiske arbeidsmåtene i utstrakt grad (skår 4). Vi ser relativt lite *programmering* i de segmentene vi filmet (7 %). Resultatene viser tydelig at programmering ikke undervises mye i de matematikktimene vi har observert. Når det forekom, handlet undervisningen om at elevene skulle lære seg syntaksen til et programmeringsspråk (skår 2; 6 %), uten at de løste programmeringsoppgaver (skår 3–4; 1 %). Nedenfor går vi nærmere inn på hver av de fire kategoriene.

4.2 Hva kjennetegnet læreres undervisning?

Lærerens undervisning kan skåres på bakgrunn av to ting, enten at læreren underviser om algoritmisk tenkning og programmering, eller at læreren legger til rette for at elevene kan jobbe med det. Hovedtrekket var at de filmede timene var kjennetegnet av at læreren ikke underviste om algoritmisk tenkning og programmering i matematikktimene, men i stedet la til rette for elevenes arbeid med algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering. For eksempel benyttet lærerne digitale veiledere for å forklare elevene programmeringskonsepter. De følgende avsnittene er en gjennomgang av våre data, men de er ikke egnet til å generalisere til trinn og fag mer generelt. Til dette er det for liten utvalgsstørrelse og for stor samvariasjon mellom enhetene i utvalget (lærere, klasser og timer).

Figur 4.2 viser hvordan lærerens tilrettelegging fordelte seg på skår 1–4 på ulike trinn og fag. På ungdomstrinnet la læreren la til rette for slikt arbeid noe oftere på 8. trinn (68 %) enn på 10. trinn (61%), selv om det var liten forskjell. På begge trinn prioriterte læreren å vise eller legge til rette for at elevene lærte en kjent algoritme for å løse en gitt type oppgave, syntaksen til et programmerings-språk eller enkel bruk av algoritmiske arbeidsmåter (skår 2; 37 %). Lærerne la også til rette for at elevene kunne jobbe med enkel algoritmisk problemløsning, forståelse av programmeringskonsepter, eller algoritmiske arbeidsmåter (skår 3), noe oftere på 8. trinn (28 %) enn på 10. trinn (23 %). I noen segmenter la lærerne til rette for elevenes løsning av problemer som krever avansert algoritmisk problemløsning eller omfattende programmeringsarbeid og bruk av algoritmiske arbeidsmåter (skår 4; 2–3%).

Figur 4.2 Prosentvis fordeling av skår for lærernes undervisning i 29 klasser



	8. trinn	10. trinn	Vg1 1P	Vg1 1P-Y	Vg1 1T
4	3	2	4	1	3
3	31	20	3	7	31
2	40	34	14	22	7
1	35	36	31	19	0

På Vg1 observerte vi en tydelige forskjeller mellom de tre matematikkfagene når det gjaldt lærernes tilrettelegging for algoritmisk tenkning og programmering. I 1T observerte vi det i alle timer (100 %), mens det forekom i 61 % av segmentene på 1P-Y og i 40 % av timene på 1P.

Lærerens undervisning i faget 1T (studiespesialiserende program) skilte seg ut ved at de i 75 % av segmentene la til rette for at elevene kunne jobbe med enkel algoritmisk problemløsning, forståelse av programmeringskonsepter, eller algoritmiske arbeidsmåter (skår 3). I de resterende segmentene la de til rette for at elevene lærte en kjent algoritme for å løse en gitt type oppgave, syntaksen til et programmeringsspråk eller enkel bruk av algoritmiske arbeidsmåter (skår 2; 18 %) og for løsning av problemer som krever avansert algoritmisk problemløsning eller omfattende programmeringsarbeid og legger til rette for bruk av algoritmiske arbeidsmåter (skår 4; 7 %).

Lærerens undervisning i fagene 1P (studiespesialiserende program) og 1P-Y (yrkesfaglige program) lignet mer på det lærerne gjorde på ungdomstrinnet. De la til rette for at elevene lærte en kjent algoritme for å løse en gitt type oppgave, syntaksen til et programmerings-språk eller enkel bruk av algoritmiske arbeidsmåter (skår 2), noe oftere i 1P-Y (45%) enn i 1P (27 %). Deretter la de til rette for at elevene kunne jobbe med enkel algoritmisk problemløsning, forståelse av programmeringskonsepter, eller algoritmiske arbeidsmåter (skår 3; 6–14%), og i noen få segmenter la de til rette for elevenes løsning av problemer som krever avansert algoritmisk problemløsning eller omfattende programmeringsarbeid og legger til rette for bruk av algoritmiske arbeidsmåter (skår 4; 2–8%).

Vi kommer mer inn på lærerens rolle i delene nedenfor, der vi kommer inn på hva som kjennetegnet elevenes arbeid med algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering.

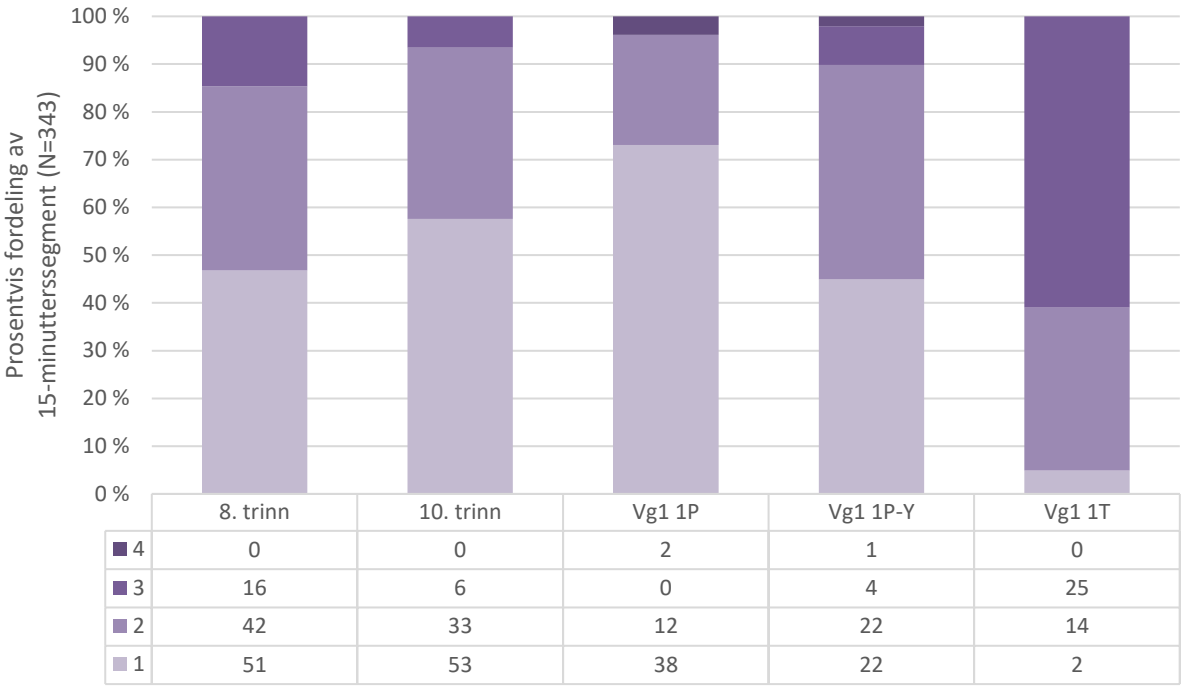
4.3 Hva kjennetegnet elevenes arbeid med algoritmisk problemløsning?

Elevene arbeidet med algoritmisk problemløsning i halvparten av timene vi observerte, i 178 av 343 segmenter (52 %). I de resterende segmentene jobbet de med andre temaer (skår 1). Det ble jobbet litt mer med algoritmisk problemløsning på Vg1 (56 %) enn på ungdomstrinnet (47 %). Se Figur 4.3.

Elevene trente primært på å utføre ulike kjente algoritmer (skår 2; 36 %) og dette så vi på alle trinn og fag. Noen ganger var det del av det å lære noe mer prinsipielt, slik som i en klasse på 8. trinn som jobbet med å forstå hva uttrykk som $2x + 3y$ kan bety ved å spille et terningspill der den ene terningen sto for verdien av x og den andre verdien av y. Der jobbet elevene altså med å forstå hvordan man tolker slike uttrykk, ved å øve på å regne verdien av uttrykkene for forskjellige verdier av x og y. Andre ganger var dette ren repetisjon fra tidligere arbeid, for eksempel i en klasse på Vg1 (1P-Y) der elevene i en halv time øvde på omgjøring mellom enhetene cm, dm, m og km.

Den største forskjellen mellom trinnene gjaldt elevenes arbeid med enkel algoritmisk problemløsning (skår 3; 15 %). På ungdomstrinnet forekom dette relativt sjelden (11 %), men likevel dobbelt så ofte på 8. trinn som på 10. trinn. På Vg1 forekom det oftere i 1P-Y (8 %) og i 1T (61 %), mens vi ikke observerte det i det hele tatt i 1P. På ungdomstrinnet og i 1T jobbet elevene aldri med avansert algoritmisk problemløsning (skår 4), mens vi observerte det i noen grad på 1P (4 %) og i en femtiendedel av segmentene i 1P-Y (2 %). Vi leser ikke mye inn i forskjellen mellom 1T og de andre fagene, fordi vi tilfeldigvis observerte kun timer i 1T der det ble jobbet med figurtall. Som vi viser senere i rapporten, var dette den vanligste konteksten å skåre høyt på i algoritmisk tenkning.

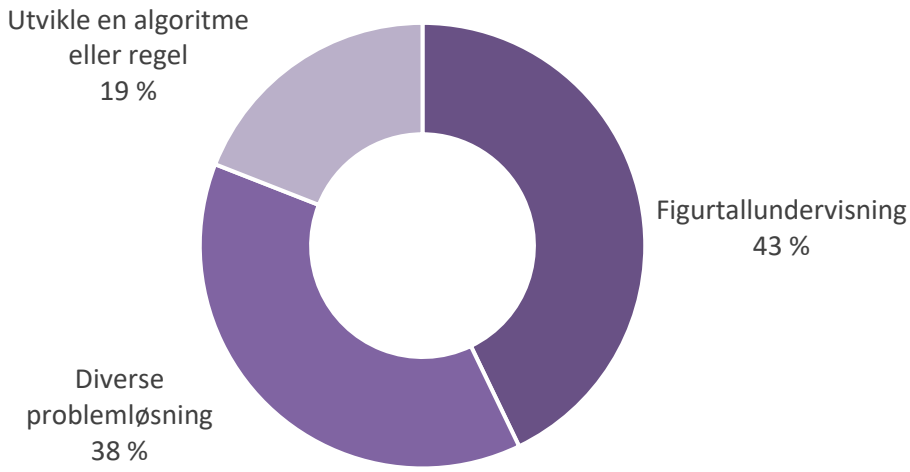
Figur 4.3 Prosentvis fordeling av skår for algoritmisk problemløsning i 29 klasser



	8. trinn	10. trinn	Vg1 1P	Vg1 1P-Y	Vg1 1T
4	0	0	2	1	0
3	16	6	0	4	25
2	42	33	12	22	14
1	51	53	38	22	2

I den kvalitative næranalysen (se del 3.6) delte vi elevenes arbeid med algoritmisk problemløsning i tre tematiske kategorier: *figurtallundervisning, utvikle en algoritme eller regel* og *diverse problemløsning*. Figur 4.4 viser en fordeling av hvor ofte de tre temaene forekom i timene med skår 3 og 4. Nedenfor går vi nærmere inn på de tre temaene, med eksempler fra ulike timer.

Figur 4.4 Andel timer for tre tematiske kategorier av *algoritmisk problemløsning*



Algoritmisk problemløsning: Figurtallundervisning

Nesten halvparten av timene (43 %) som inneholdt algoritmisk problemløsning handlet om *figurtallundervisning.* Dette var overraskende, siden figurtall ikke er nevnt i noen kompetansemål. Vi observerte likevel elever som jobbet med figurtalloppgaver i alle matematikkfagene, både på ungdomstrinnet (8. og 10. trinn) og på Vg1 (1T, 1P og 1P-Y). På 8. trinn trente elevene på figurtall da lærerne innførte algebra, og på 10. trinn jobbet de med det i forbindelse med funksjoner og regresjon. På Vg1 fremsto det nærmest som et eget tema i 1P og 1T, og i 1P-Y dukket figurtall opp som en typisk problemløsningsoppgave.

Vi observerte lite progresjon i figurtallundervisningen mellom de tre trinnene. For eksempel ble en av oppgavene som elevene jobbet med under innføring av algebra på 8. trinn også jobbet med i 1T og 1P på Vg1, bare litt raskere. Dette er en oppgave som omtales som «Korset» blant lærere (se Petersen, 2002). og så skulle elevene prøve å løse dette ved hjelp av regresjon med et lineært eller kvadratisk uttrykk og se om noen av dem passet til de oppgitte verdiene.

Enkelte lærere presenterte en strukturert algoritme for å jobbe med figurtalloppgaver. Disse segmentene inneholdt lite dekomponering, mønstergjenkjenning og logisk tenkning, og elevene fulgte primært en gitt algoritme (skår 2). Denne algoritmen var gjerne at elever skulle fylle ut en rubrikk som den i Figur 4.5 fra venstre mot høyre og til slutt bytte ut «tallet som økte» med n. En annen algoritme var å gjøre en regresjonsanalyse i GeoGebra. Den innebar å fylle inn verdiene som var oppgitt i tabellen (for oppgaven i Figur 4.5 ville dette vært: $f(1) = 6, f(2) = 10$ og $f(3) = 14$) og så prøve regresjon med lineære eller kvadratiske uttrykk for å se om regresjonslinja passet til de oppgitte verdiene.

Figur 4.5 Figurtalloppgave: en rubrikk elever fylte ut i timen

Figur				
Nummer	1	2	3	n
Figurnummer	6	10	14	
Regnemåte	$2 + 4$	$2 + 4 \cdot 2$	$2 + 4 \cdot 3$	$2 + 4 \cdot n$

Stort sett handlet figurtallundervisningen om enkel algoritmisk problemløsning (skår 3). Elevene fikk presentert mønstre som vokste, og de ble stilt spørsmål som: «a) hvor mange brikker trengs for å lage figur 4?, b) eller figur 100?, c) Hva er den første figuren som trenger mer enn 50 knapper?» Elevene skulle dermed se etter mønstre i hvordan figuren vokste, dekomponere mønsteret i sine bestanddeler og summere dem. Stort sett var mønstrene av samme kompleksitet som i «Korset» i Figur 4.5. Mønsteret vokste med et gitt antall prikker hver gang, så det var en lineær sammenheng mellom figurens nummer og antall brikker. Dette er tilstrekkelig for enkel algoritmisk problemløsning (skår 3).

En av timene med figurtall fikk skår 4. I Rasmus sin klasse observerte vi elevene besvare en samarbeidsprøve. Elevene jobbet stort sett alene, bortsett fra i tre ti-minutters intervaller i løpet av en dobbelttime der de kunne snakke med hverandre. På prøven var det en figurtalloppgave med fyrstikker. Elevene beskrev mønsterets utvikling rekursivt, altså ved å henvise til hvor mange fyrstikker den forrige figuren inneholdt. Med en rekursiv formulering av veksten klarer man ikke å svare på typiske figurtall-spørsmål, så elevene måtte analysere utviklingen dypere, noe som krevde avansert algoritmisk problemløsning (skår 4). Her forklarer en elev mønsteret hun så:

> Elev: Neste figur har alltid like mange som forrige figur pluss to mer enn forrige gang.
> Rasmus: Hva? Forklar det.
> Elev: Altså første figur har 1, og neste har 1 pluss 2, altså 3. Og neste der igjen har pluss 4, altså 7. Så pluss 6. Så pluss 8. Så hver figur har liksom forrige figur pluss to mer hver gang.

Hvis antallet fyrstikker i Figur n kalles a_n kan vi beskrive mønsteret eleven så slik, og så videre:

$$a_1 = 1$$
$$a_2 = a_1 + 2$$
$$a_3 = a_2 + 4$$

Her måtte eleven gå noen skritt videre enn i de andre figurtalloppgavene og fant at elevens mønster kunne uttrykkes som $a_n = a_{n-1} + 2(n-1)$, men det hjalp dem ikke videre. Aha-opplevelsen kom da eleven begynte å regne på antall fyrstikker manuelt, slik at eleven kunne besvare noen av oppgavene. Da så elevene at figurtallene, 1 3 7 13 21 31 43, var én mer en rektangeltallene de hadde jobbet med tidligere, noe som førte til hektisk arbeid videre. Det er uklart om elevene kom frem til et direkte uttrykk for figurtall a_n – det krever en del videre arbeid. Arbeidet med å bearbeide den rekursive beskrivelsen til et direkte uttrykk (som de ikke fikk til), bruke den rekursive beskrivelsen som en algoritme til å regne de første figurtallene, kjenne igjen mønsteret og jobbe videre med det, forsvarte kategorien «avansert algoritmisk problemløsning» (skår 4).

Algoritmisk problemløsning: Diverse problemløsning

Over en tredel av timene (38 %) som inneholdt algoritmisk problemløsning handlet om *diverse problemløsning*. Dette var problemløsningsoppgaver som lærerne brukte mens de jobbet med konkrete temaer (for eksempel: likningsløsning), som lærerne ga i egne problemløsningstimer der elevene jobbet seg gjennom ark med «matematikknøtter», eller som lærerne startet eller avsluttet timen med. Vi presenterer flere av disse oppgavene nedenfor.

I sin klasse på 8. trinn, startet Olivia en matematikktime med *Kjeksoppgaven* (Boks 7). Dette er en brøkoppgave der hele familien spiser en gitt brøkdel kjeks, inntil det bare er seks kjeks igjen. Spørsmålet er hvem som spiste flest kjeks. Elevene dekomponerte oppgaven og jobbet stegvis. Først fant de ut at bestemor hadde spist 6 kjeks, slik at Ina måtte ha etterlatt 12 kjeks, og slik fortsatte de. Det ble gjort mange feil underveis, slik som at Ina måtte ha spist 3 kjeks, siden en firedel av 12 er 3, men de oppdaget feilene da de regnet seg gjennom til slutt for å se om alle tallene stemte og kom frem til at det var seks kjeks igjen. Denne dekomponeringen, stegvise jobbingen og logiske tenkningen beskriver enkel algoritmisk problemløsning (skår 3).

Boks 7 Kjeksoppgaven (8. trinn)

Det står en boks med kjeks på et bord. I løpet av natten spiser Bestefar en seksdel av kjeksene. Så spiser Ole en femdel av det som er igjen. Tante spiser en tredel av de som er igjen. Ina spiser en firedel av resten. Når morgenen kommer spiser Bestemor halvparten av det som var igjen. Da er det seks kjeks igjen.

Hvem spiste flest kjeks?

Karen underviste i algebra til en klasse på 8. trinn. Hun ga dem *Smileyoppgaven* (Boks 8), som er en oppgave fra Matematikksenterets «MatteLIST»-ressurser (Boks 8). Karen presiserte at det var viktig at elevene ikke bare skrev ned svaret, men at de også måtte skrive ned hvert resonneringstrinn de gjorde. Elevene jobbet i rundt 20 minutter før noen ble ferdige, og de som ble ferdige fikk nye oppgaver av samme type. Etter 30 minutter skulle elevene sette seg sammen med andre elever for å forklare hvordan de hadde løst oppgaven. Denne oppgaven må dekomponeres i flere trinn, og elevene må se etter mønstre og tenke logisk. For eksempel må man innse at hvis 🤓 · 🥲 = 🥲så gjelder 🥲 = 0 eller 🤓 = 1. Denne oppgaven krever også enkel algoritmisk problemløsning (skår 3).

Boks 8 Smileyoppgaven (8. trinn)

Denne oppgaven var hentet fra Matematikksenterets «MatteLIST»-ressurser. I Smileyoppgaven var det 12 likninger av typen 😅 · 😈 = 😎 der hvert symbol sto for et av heltallene fra 0 til 12. Se lenken for detaljer om oppgaven (https://www.mattelist.no/176).

Ine underviste en yrkesfagsklasse i 1P-Y. Hele timen var satt av til problemløsning på vertikale tavler der klassen skulle jobbe i grupper og konkurrere mot hverandre. Gruppen med flest løste oppgaver vant. Ine viste problemløsningsoppgaver på fremviseren og omtrent hvert tiende minutt bladde Ine videre i lysbildene sine slik at klassen fikk et nytt problem å prøve seg på. Underveis førte hun poeng. Mange av problemene krevde algoritmisk problemløsning, men kun én oppgave krevde avansert algoritmisk problemløsning (skår 4), se *Lyspæreoppgave* (Boks 9).

Dette er en klassisk matematikknøtt der elevene må telle hvor ofte hver av 50 lyspærer blir slått av og på av 50 forskjellige personer. De fleste elevene startet med å prøve å holde oversikt over hver enkelt pære, ettersom personene slo dem av og på, men elevene mistet raskt oversikten og lagde ulike systemer.

Boks 9 Lyspæreoppgaven (Vg1)

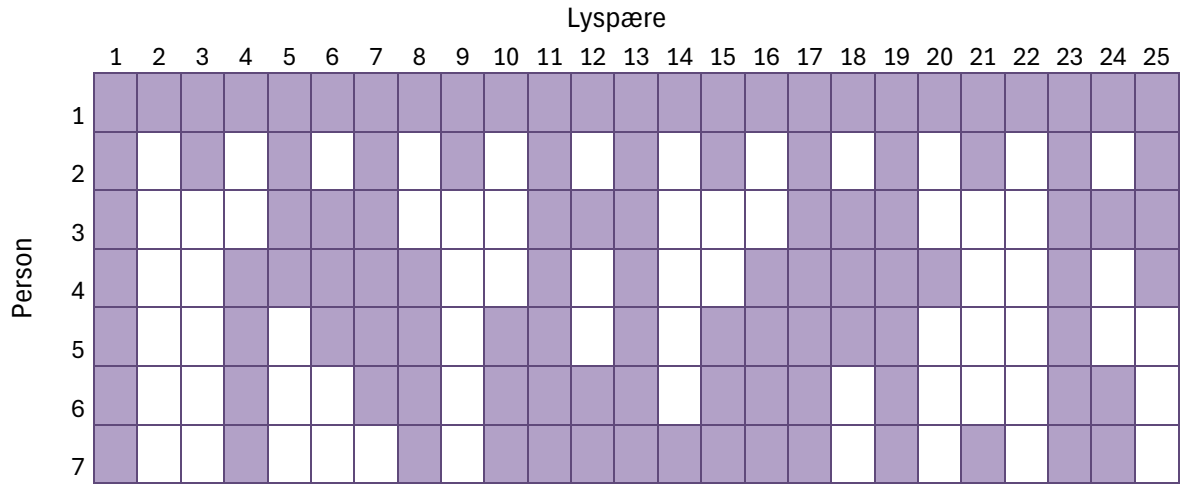
50 personer skal gå gjennom en tunnel. I tunnelen er det 50 lyspærer og alle har sin egen knapp. Når man trykker på knappen, skrur man på eller av pære, avhengig av om den er på eller av fra før av. Alle pærene er avslått i begynnelsen.

- Den første personen trykker på alle knappene. Alle lyspærene lyser.
- Den andre personen trykker på knapp nummer 2, 4, 6, osv. Det skrur disse pærene av.
- Slik fortsetter det; hver person trykker på alle knappene som har et nummer som er delelig med personens nummer.

Hvilke pærer lyser når alle har gått gjennom tunnelen?

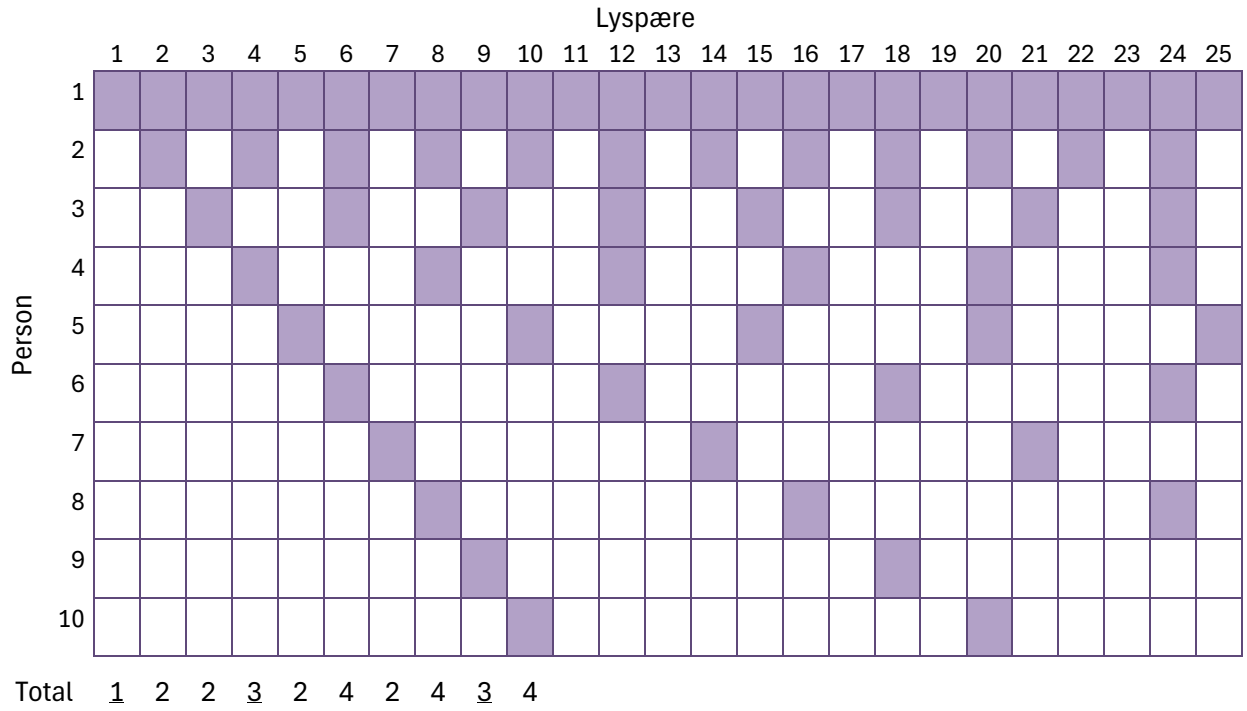
De fleste elevene startet med å holde oversikt over hvilke pærer som var av og på etter at hver person gikk forbi. I starten gjorde de det usystematisk, men de fant at ut at de måtte føre det systematisk i en tabell. For flere av elevgruppene, så det omtrent ut som i Figur 4.6. I figuren vil lilla farge i celle (m, n) bety at pære m er slått på når person n har passert. For eksempel kan vi i kolonne 6 se at den sjette lyspæren er påslått etter person 1 (lilla), avslått etter person 2 (hvit) og slått på igjen etter person 3 (lilla). Deretter holder den seg påslått (lilla) frem til person 6, og deretter forblir den avslått (hvit). Etter hvert fant elevene ut at de mistet oversikten, det var for vanskelig å holde rede på hver enkelt lyspære og lett å gjøre en liten feil. Enkelte grupper ga opp tabeller og prøvde å holde oversikt på andre måter, mens andre grupper forkastet den typen tabell og heller gikk over til å holde rede på hvor mange ganger en lyspæres knapp ble trykket på, slik som i Figur 4.7.

Figur 4.6 Lyspæreoppgaven: system 1



I Figur 4.7 betyr lilla farge i en celle (m, n) at knappen til lyspære m blir trykket på av person n. Fargene i kolonne 6 betyr at den sjette knappen blir trykket på av person 1, 2, 3 og 6, fordi cellene er lilla. Nederst er det en rad for «Total», der elevene summerte opp hvor mange ganger lyspærene ble slått på og av, altså hvor mange lilla ruter det var i kolonnen. Siden lyspære 6 ble slått på og av fire ganger ender den opp som «av». Kun lyspære 1, 4 og 9 ble slått på og av et odde antall ganger, så kun de blir stående påslått, etter at de 10 første personene er forbi.

Figur 4.7 Lyspæreoppgaven: system 2



Elevene som benyttet tabeller (Figur 4.6 og 4.7) brukte en riktig fremgangsmåte, men den var for komplisert å utføre for hånd. Noen kom frem til det riktige svaret, nemlig at lyspære 1, 4, 9, 25, 36 og 49 forble påslått etter at alle hadde passert, mens andre elever fikk andre svar på grunn av småfeil. Flere av gruppene så, ut fra et slikt tellesystem, at det handlet om hvor mange heltallsdivisorer tallene har. Siden tallet 6 har heltallsdivisorene 1, 2, 3 og 6 blir den knappen slått av og på 4 ganger. Elevene forsto da at oppgaven kunne løses ved å finne ut hvilke tall som har odde antall heltallsdivisorer. Flere av elevene kom etter noen få minutter frem til at det måtte være kvadrattallene, mens andre ikke klarte å løse oppgaven i løpet av de 20 minuttene de fikk til å løse oppgaven. Vi observerte ingen gode begrunnelser fra elevene for hvorfor kvadrattallene har odde antall divisorer, og læreren etterspurte heller ikke.

Lyspæreoppgaven illustrerer flere av nøkkelbegrepene i den algoritmiske tenkeren (se Del 2), som utgjør algoritmisk problemløsning. De fleste elevene *dekomponerte* problemet og utformet en *algoritme* som kunne latt seg *automatisere* da de tegnet opp tabellene sine. De *evaluerte* at metodene deres var ineffektive og så etter et *mønster* i tabellen. De tenkte *logisk* at kun de personene hvis nummer delte lyspærenummeret slo av og på bryteren og at oppgaven derfor kunne løses ved å telle heltallsdivisorer.

Elevene *abstraherte* bort nesten alt ved problemet – lyspærer, personer, knapper og tunneler – og igjen sto kun antallet heltallsdivisorer. I arbeidet med denne oppgaven jobbet elevene med avansert algoritmisk problemløsning (skår 4). Undervisningen skåret samtidig høyt på de andre kategoriene, ved at elevene benyttet seg av *algoritmiske arbeidsmåter* og en elevgruppe forsøkte også å *programmere* en løsning (se del 4.5).

Algoritmisk problemløsning: Utvikle algoritme eller regel

Omtrent en femtedel av timene (19 %) som inneholdt algoritmisk problemløsning handlet om *å utvikle en algoritme eller regel.* Disse timene var på samme skole og elevene jobbet med temaet når de skulle lære seg potensregning på 8. trinn. Læreren startet timene med «Dagens problem», som elevene jobbet med alene eller med sidemannen, og disse dreide seg om å utvikle den egenskapen ved potenser som de skulle jobbe med den timen. Oppgaven inkluderte å utforme en regneregel for å finne fortegnet til en potens med negativt grunntall, en regneregel for å multiplisere og dividere potenser med samme grunntall og å finne ut hva «opphøyd i nullte» betyr. Nedenfor viser vi hvordan elevene fant en regneregel for å multiplisere potenser med samme grunntall i oppgaven *Gange potenser* (Boks 10).

Boks 10 Gange potenser (8. trinn)

1. Forenkle til én potens hvis det er mulig:
   a) $3^4 \cdot 3^6$ b) $4^2 \cdot 4^7 \cdot 4^4$ c) $2^3 \cdot 3^2$
   d) Når kan du legge sammen eksponentene, og når må du regne ut potensene hver for seg?

2. Divisjon er det motsatte av multiplikasjon. Undersøke disse regnestykkene og lag en regel for hvordan du kan dividere to potenser med samme grunntall:
   a) $10^8 : 10^3$ b) $3^7 : 3^2$ c) $8^3 : 8^1$

I sin klasse på 8. trinn ga Hannah oppgaven *Gange potenser*. Elevene hadde tidligere jobbet med hva en potens betyr og regnet verdien av noen uttrykk med potenser. Spørsmålsstillingen «Forenkle til en potens hvis det er mulig» var likevel ny. Elevene var i villrede og måtte hjelpes en del med oppgave 1a). Hannah gikk rundt og oppfordret elevene til å skrive 3^4 og 3^6 som et «stooooort regnestykke» mens hun holdt hendene langt fra hverandre for å vise hvor stort det ble. Mange elever ga svaret: 59 049, fordi de hadde regnet ut verdien av uttrykket på kalkulatoren. Etter 20 minutter oppsummerte de oppgaven på tavla. Det ble tydelig at ikke alle elevene hadde forstått formålet med oppgaven, men at andre hadde det:

Hanna: Var det utfordrende? Mange av dere trengte litt hjelp, så begynte mange å se det. Så, først «tre i fjerde ganger tre i sjette», er det mulig å forkorte til én potens?
Elev 1: Tre i tiende
Hanna: Hvordan kom du frem til det?
Elev 1: Fordi du hjalp oss. Fordi fire pluss seks er ti.
Hanna: Hvorfor kan du bare legge sammen eksponentene når grunntallet er det samme?
Elev 1: Når du ganger med dem kan du det. Hmm, jeg vet egentlig ikke?
Elev 2: Hvis du legger ut «tre ganger tre ganger tre ganger tre», hele greia.

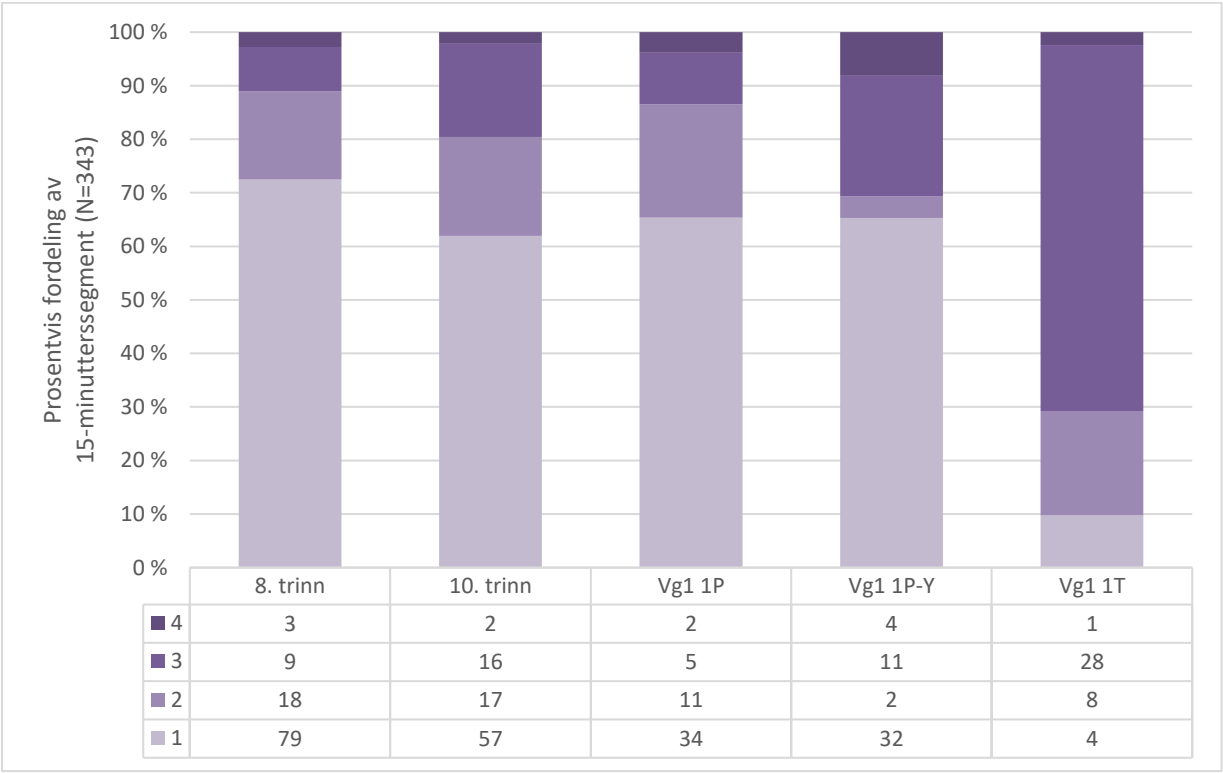
Hanna: [Skriver ut $3 \cdot 3 \cdot 3 \cdot 3$ på tavla]. Enig i at det er tre i fjerde. Tre i sjette?
Elev 2: Det er bare ti tretall som skal ganges sammen.
Hanna: Er alle med på det?
(Diskusjonen fortsetter)

Det er tydelig at Elev 1 hadde lært seg en algoritme for å gange sammen potenser med likt grunntall uten å forstå hvorfor. Derimot hadde Elev 2 tydelig en forklaring på hvorfor algoritmen fungerte. Ut fra denne samtalen var det ikke tydelig om det er elevene som har bedrevet algoritmisk problemløsning, som er et krav for å bli kodet som «algoritmisk problemløsning», eller om det var Hanna som hadde gitt forklaringen til elevene og at elevene gjengir forklaringen. Fra våre observasjoner i resten av timen ble det tydelig at flere elever diskuterte oppgaven, brøt ned problemet og forklarte selv hvordan algoritmen fungerte. Dette kodes som at «[elevene] analyserer en enkel algoritme for å finne ut […] hvordan den fungerer» (skår 3).

4.4 Hva kjennetegnet elevenes arbeid med algoritmiske arbeidsmåter?

Elevene arbeidet med algoritmiske arbeidsmåter i under halvparten av timene vi observerte, i 137 av 343 segmenter (40 %). For koden *algoritmiske arbeidsmåter* fikk en femtedel av segmentene skår 3, og de var fordelt på omtrent en tredel av timene. Omtrent én av tretti segmenter fikk skår 4 (3 %) og de var fordelt på en tyvendedel av timene, se Figur 4.8.

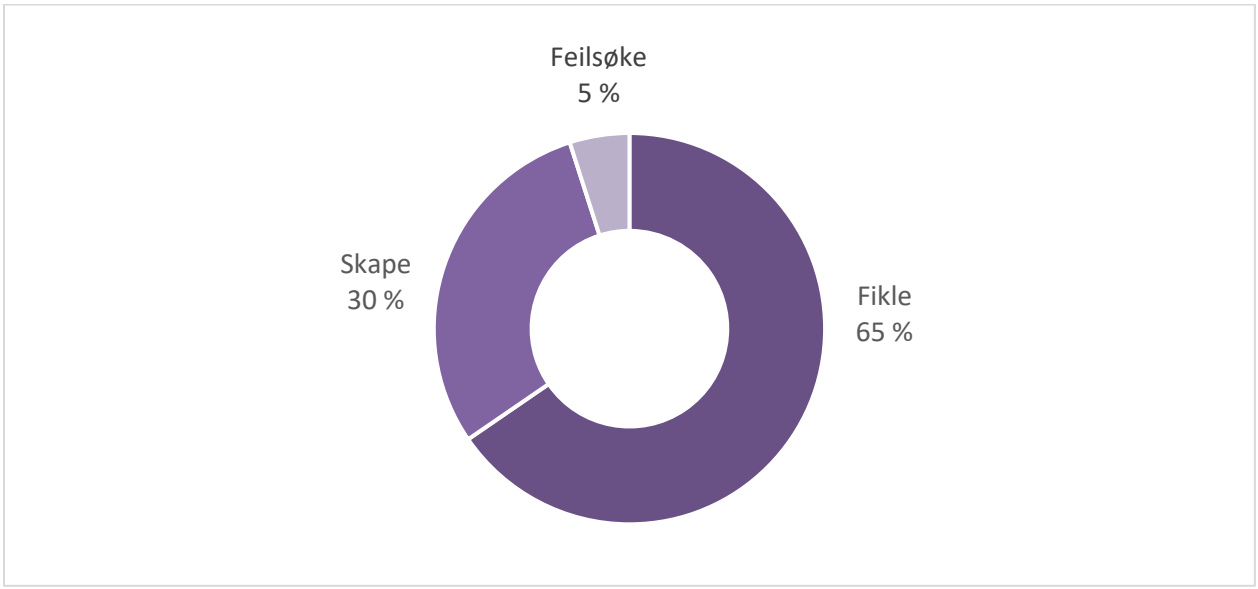
Figur 4.8 Prosentvis fordeling av skår for algoritmiske arbeidsmåter i 29 klasser



	8. trinn	10. trinn	Vg1 1P	Vg1 1P-Y	Vg1 1T
4	3	2	2	4	1
3	9	16	5	11	28
2	18	17	11	2	8
1	79	57	34	32	4

I den kvalitative næranalysen (se del 3.6) delte vi elevenes arbeid med algoritmiske arbeidsmåter i tre tematiske kategorier: *fikle, skape* og *feilsøke*. Figur 4.9 viser en fordeling av hvor ofte de tre arbeidsmåtene forekom i timene med skår 3 eller 4. Nedenfor går vi nærmere inn på de tre temaene, med eksempler fra ulike timer. De to siste algoritmiske arbeidsmåtene, *samarbeid* og *holde ut*, var det vanskelig å undersøke via klasseromsobservasjoner. Hvis elevene ikke samarbeider, hører vi ikke hva de jobber med og da vet vi ikke om de driver med algoritmiske arbeidsmåter. Derfor samarbeider elevene på nesten alle segmenter som er kodet for algoritmiske arbeidsmåter (skår 2, 3 og 4). *Holde ut* var også vanskelig å kode. Det kommer gjerne tydelig frem når elever gir opp den oppgaven de jobber med, men det var alltid noen elever som holdt ut og det er tilstrekkelig ut fra observasjonsmanualen. Likevel viste alle segmentene som skåret 4 i utpreget grad elever som holdt ut selv når de ikke fikk til oppgaven.

Figur 4.9 Andel timer for tre tematiske kategorier av *algoritmiske arbeidsmåter*



Algoritmisk arbeidsmåte: Fikle

Omtrent to tredeler av timene (65 %) som inneholdt algoritmiske arbeidsmåter handlet om *å fikle.* Vi minner om at *fikle* er en oversettelse av det engelske *tinker*, og betyr en iterativ leken utprøving for å komme videre med en oppgave eller prosjekt. Hvis elevenes arbeid var preget av fikling, og gjerne kombinert med samarbeid eller utholdenhet, ble segmentet skåret 3. Vi har allerede beskrevet hvordan elevene fiklet i både *Smileyoppgaven* (Boks 8) og *Lyspæreoppgaven* (Boks 9), og de ble kodet til skår 4. Elevene var uredde for å starte på noe de ikke visste om ville føre frem, de prøvde, systematiserte det de fant, forkastet noen forsøk, prøvde noe nytt, og itererte videre inntil de kom frem til en løsning. Et godt eksempel på segmentene som fikk skår 3 er mange av segmentene med figurtall. Elevene prøvde seg frem med forskjellige uttrykk, testet dem ut og endret på dem hvis de var feil. Fikling var også sterkt til stede i segmentene som inneholdt skaping og feilsøking

Algoritmisk arbeidsmåte: Skape

En tredel av timene (30 %) som inneholdt algoritmiske arbeidsmåter handlet om *å skape* noe. Vi minner om at å *skape* betyr å lage et produkt, gjerne noe som skal brukes, vises frem, eller liknende og ikke bare være et svar på en oppgave læreren har gitt. Hvis timen inneholdt å skape et produkt, var det ofte minst ett segment som fikk skår 4 på algoritmiske arbeidsmåter. Disse segmentene viste at hele klassen var aktiv, alle fiklet med produktet sitt, de samarbeidet og holdt ut. Feilsøking var mindre fremtredende.

Et eksempel på dette, var oppgaven *Surpriseparty* (Boks 11), som fikk skår 4 i flere segmenter. Oppgaven ble gitt på 10. trinn og dreide seg om å lage en overraskelsesfest for en kollega på jobben. Vi har endret alle navn og stedsnavn i oppgaveteksten for å bevare anonymiteten til skolen, lærerne og elevene. I oppgaven var det fire forskjellige festlokaler, to forskjellige musikkalternativer og tre forskjellige måter å ordne mat og drikke på. Ikke alle alternativene kunne kombineres med hverandre, og de hadde ulike styrker og svakheter. Oppgaven ba elevene skaffe oversikt over alternativene, for lettere å ta et valg. I undervisningen ble det gjort tydelig fra læreren at målet med oppgaven var å finne det beste alternativet og argumentere for hvorfor det var best. Det beste alternativet kunne ikke være for dyrt for hver deltager, men det måtte selvsagt balanseres opp mot å lage en god fest. Elevene jobbet i grupper og brukte halvannen økt på å skape en oversikt over alternativene, velge det beste alternativet og finne en måte å presentere det for klassen. Deretter presenterte de fleste gruppene arbeidet sitt.

Boks 11 Surpriseparty (10. trinn)

Roar vil lage en fest for Lisbeth, og tenker at han skal invitere mellom 30 og 70 gjester, men er veldig usikker på hvor mange han skal invitere. Det skal være middag og det skal være musikk å danse til utover kvelden. Det er tenkt at gjestene skal være med å spleise på utgiftene. Lisbeth skal ikke betale noe selv. Roar betaler for seg selv og sponser festen med kr 10 000 i gave til Lisbeth.

Alternative lokaler:

- *Meteor* koster kr 19 000. De har servitører og DJ.
- Grendehuset koster kr 4 500. Max 43 personer.
- Kan låne gratis lokaler av jobben til Greta sin mann, men da må vi bruke den cateringavtalen de har: kr 900 pr person inkludert drikke.
- Velforeningens lokaler koster kr 9 000. Det er med lydutstyr. Der er de nøye på renhold, så der bør det vurderes å bruke kr 2 500 på vaskebyrå.

Musikkalternativer:

- Ove er DJ. Han skal få kr 4 000 og gratis inngang og mat.
- Leie av lydutstyr koster kr 3 500.

Mat og drikke:

- Lage viltgryte og dessert selv koster kr 150 per person. Drikke ikke inkludert.
- Catering koster kr 350 pr person.
- På *Meteor* koster maten kr 200 pr person, og vi må kjøpe middag der.

Hjelp Roar å få oversikt over ulike muligheter, slik at det blir lettere å ta et valg.

Arbeidet med oppgaven inneholdt *skaping*, fordi elevene skulle lage et produkt, et ferdig festforslag, og de brukte utstrakt *fikling* for å skape det. Elevene gikk frem på ulike måter. Noen startet med å lage budsjett for hele festpakker inkludert lokale, musikkalternativ, mat og drikke og regnet ut prisen det ville kostet. Et problem elevene støtte på var at de ikke visste hvor mange som kom på festen, og noen alternativer ble dyre hvis det kom 30 gjester og rimelige hvis det kom 70 gjester og omvendt. Elevene kom frem til at de kunne dele totalprisen på antall deltagere og dermed få en omvendt proporsjonal funksjon:

$$f(x) = \frac{\text{totalpris}}{\text{antall gjester}}$$

Denne ideen spredte seg i klasserommet, godt hjulpet av læreren som tydelig hadde som mål for oppgaven at elevene skulle benytte omvendt proporsjonale funksjoner. Elevene tegnet grafen til slike funksjoner for hvert av alternativene de foreslo, men noen elever på gruppa fant stadig andre alternativer som kanskje ville fungere bedre. I arbeidet med denne oppgaven var det utstrakt grad av *fikling*, *utholdenhet* og *samarbeid* i tillegg til *skapingen*. Dette var typisk for segmentene med skaping.

Flere segmenter der elevene skapte noe fikk skår 3. Et eksempel er en yrkesfagsklasse på Vg1 (FAG) som skulle lage en spenningsdeler. Elevene skulle designe en krets på et 12-volts batteri der én komponent skulle ha 6,5 V over seg. Deretter skulle de teste om svaret var riktig ved å koble kretsen og måle om spenningen var riktig over komponenten. Denne timen inneholdt også *skaping*, da elevene skulle tegne kretsen og koble den, men i mer begrenset grad enn i *Surpriseparty* (Boks 11). Det var mindre fikling, samarbeid og utprøving, men likevel nok til å få skår 3.

Algoritmisk arbeidsmåte: Feilsøke

Den algoritmiske arbeidsmåten *feilsøke* var lite til stede i de observerte timene (5 %). Det kan være fordi elevenes feilsøking ikke fanges av kameraene og mikrofonene, for eksempel hvis de gjør det alene og lydløst, men vi tenker at det burde være mulig oftere å høre elever snakke sammen når de finner måter for å sjekke om de har gjort oppgaven på riktig måte, leter etter hvor de kan ha gjort feil, og diskuterer hvordan de kan rette den opp. Det vi derimot så ofte var lærere som lette etter feil i elevenes arbeid og diskuterte hvordan de kunne rette det opp. Vi fant kun to timer som var preget av feilsøking.

Den første timen som var preget av feilsøking, var *Smileyoppgaven*. Der skulle elevene finne ut hvilket tall 13 forskjellige smileys sto for, ut fra tolv likninger av denne typen: 😅·😈=😎. Her fant elevene ut at de hadde gjort feil da tallene deres ikke gikk opp. De lette i fremgangsmåten sin etter hvilke steg som var usikre, og siden de hadde blitt oppfordret av læreren til å ta strukturerte notater om hvert resonneringssteg, hadde de gode notater å feilsøke i. I hele timen var feilsøking fremtredende.

Det er verdt å nevne at feilsøking ikke var fremtredende i timene med programmering. Tvert imot var programmeringstimene preget av at når elevene hadde skrevet feil i koden og ikke fikk det resultatet de forventet, ble de ofte sittende lenge og vente til læreren kom og hjalp dem med å finne feilen. Det var ett unntak, en hjemmeundervisningstime på 8. trinn våren 2022. Der viste læreren frem oppgaver som dreide seg om å finne feil i forskjellige kodesnutter, slik som i oppgaven *Finn feilene i koden* (se Boks 12). Der skrev elevene koden på sine egne maskiner, og enkelte kom frem til riktige korrigeringer av koden til slutt. Vi så ikke hvordan elevene gjorde det, for eksempel om de analyserte feilmeldingene fra Python-fortolkeren eller gjorde noe annet.

Boks 12 Finn feilene i koden (8. trinn)

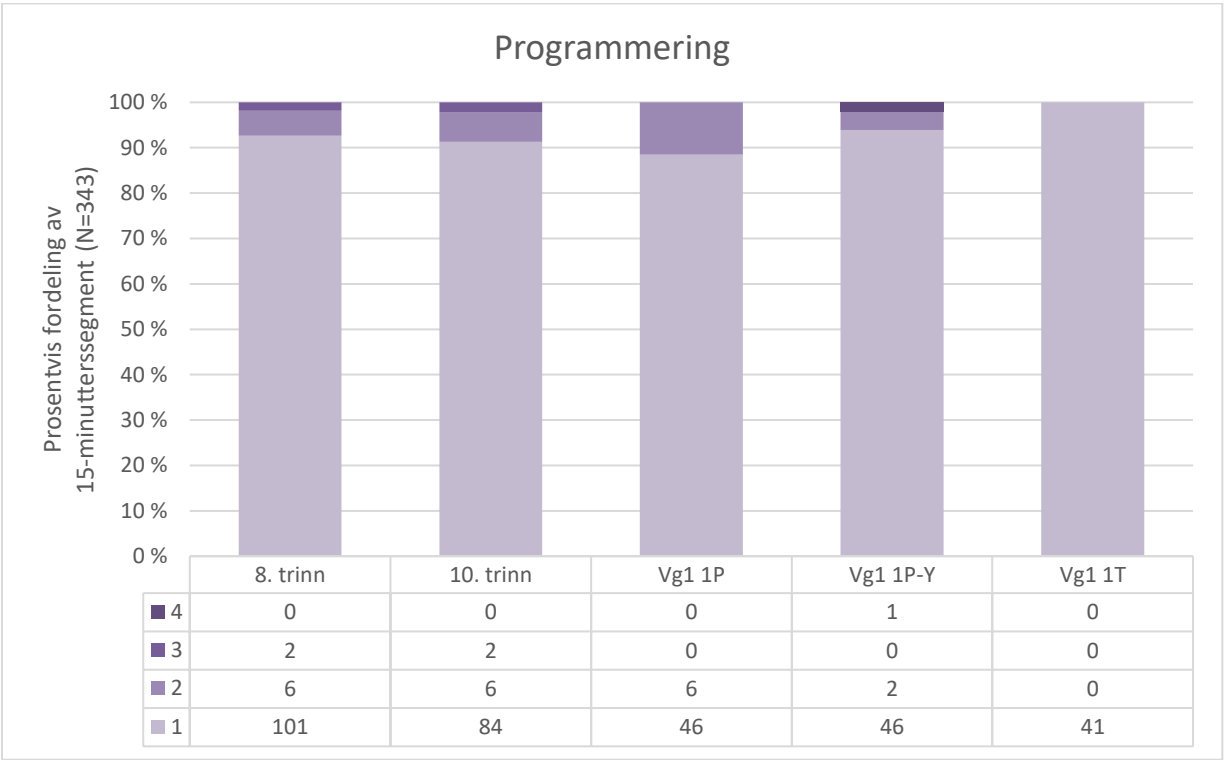
Denne koden virker ikke. Forklar hva som kan være grunnen og ordne på koden slik at den virker:

```
a = input('Hvor mange blyanter skal du kjøpe?')
print('Prisen blir ', p, kr)
12a = p
```

4.5 Hva kjennetegnet programmering?

Programmering forekom relativt sjelden i timene vi observerte, i 16 av 117 timer (14 %) og kun i 25 av 343 segmenter (7 %). Hovedfunnet er at vi i de fleste segmenter ikke observerte programmering (skår 1). Dette gjelder for alle trinn og fag, der observasjoner av programmering varierte fra 0 % på Vg1 i 1T, til 9 % på 10. trinn. Figur 4.10 viser hvordan programmering fordelte seg på skår 1–4, samt på ulike trinn og fag. Når vi observerte programmering på ungdomstrinnet, handlet det hovedsakelig om at en eller flere elever jobbet med oppgaver for å lære syntaksen til et programmeringsspråk (skår 2; 6 %) eller løste korte programmeringsoppgaver (skår 3; 2 %). Her var det ingen endring fra 8. trinn til 10. trinn. På Vg1 observerte vi liten forskjell mellom de tre matematikkfagene. På 1T var det ingen programmering (skår 1; 100 %). I 1P var det noen eksempler på at elever jobbet med oppgaver for å lære syntaksen til et programmeringsspråk (skår 2; 12 %). I 1P-Y, var det ett segment der elevene løste omfattende programmeringsoppgaver (skår 4; 2 %), i Lyspæreoppgaven (Boks 9).

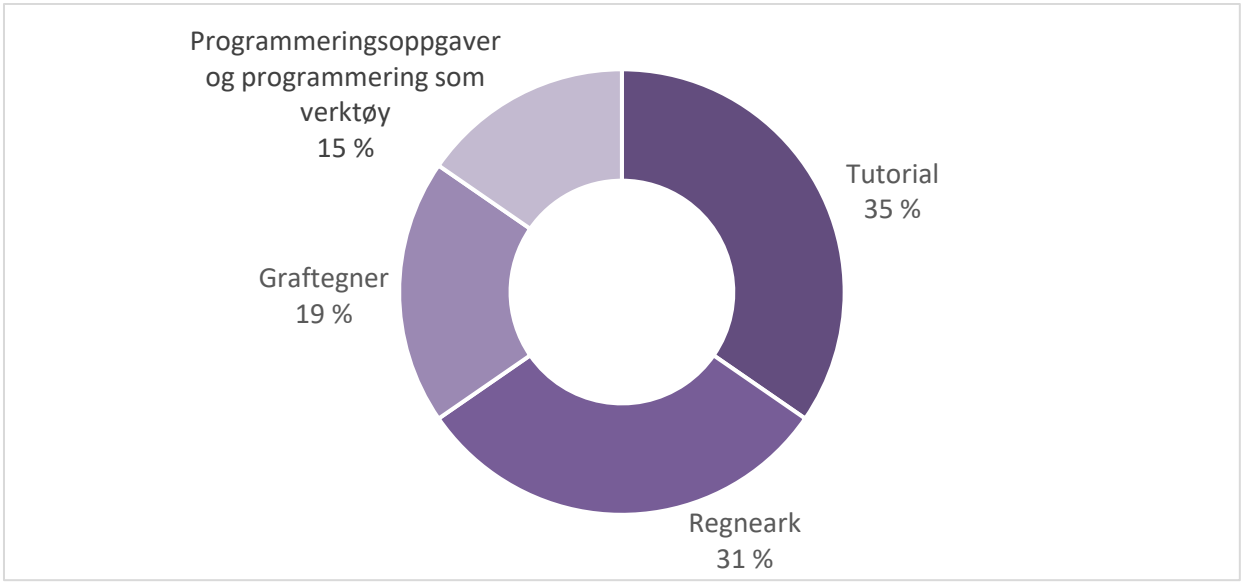
Figur 4.10 Prosentvis fordeling for programmering i 29 klasser



	8. trinn	10. trinn	Vg1 1P	Vg1 1P-Y	Vg1 1T
4	0	0	0	1	0
3	2	2	0	0	0
2	6	6	6	2	0
1	101	84	46	46	41

I den kvalitative næranalysen (se del 3.6) delte vi elevenes arbeid med programmering inn i fire forskjellige typer: *digital veileder (tutorial), regneark, graftegner* og *programmeringsoppgaver og programmering som verktøy*. Figur 4.11 viser en fordeling av hvor ofte de ulike typene programmering forekom i timene med programmering. Vi inkluderte en del formelbruk i regnearkverktøyet Excel og graftegneren GeoGebra som programmering. Hvis vi ekskluderer disse, gjenstår 6 timer (5 %) og 10 segmenter (3 %) fordelt på tre læreres undervisning i tre klasser som viste programmering i Python. Nedenfor går vi nærmere inn på de fire typene programmering, med eksempler fra ulike timer.

Figur 4.11 Andel timer for fire tematiske kategorier av *programmering*



Programmering ved hjelp av digital veileder

I halvparten av timene (6 av 12 timer) som inneholdt programmering fulgte elevene en *digital veileder*, også kalt en *tutorial.* Vi observerte dette i over en tredel av segmentene (35%) og det forekom på åttende og tiende trinn. Den digitale veilederen kombinerer korte forklaringer med at elevene får prøve å programmere selv. Digitale veiledere er laget for å være selvforklarende og er svært interaktive[9]. I matematikktimene var første del oftest en vanlig matematikktime, og så var det en overgang til programmering som en separat del av timen. I tre av timene jobbet elevene med digitale veiledere en halv time på slutten av timen, og i de resterende tre timene satte læreren av det siste kvarteret til å jobbe med en digital veileder. Boks 13 viser en oversikt over alle programmeringskonsepter og kommandoer vi observerte at elevene brukte i timene. Alle er innledende programmeringskonsepter.

[9] Et eksempel kan ses her https://learnpython.trinket.io/

Blant de programmeringskonseptene som nevnes i matematikkfagene i LK20, observerte vi arbeid med vilkår og variabler på 8. og 10. trinn. Dette er kompetansemål etter 5. trinn. Vi observerte også bruk av Pythons Turtle-pakke på 8. og 10. trinn, som kan gjenskape Paperts eksperimenter fra syttitallet (Papert, 1980), se Del 2. På Vg1 så vi vilkår bli brukt i regnearkprogramvaren Excel. Vi observerte ingen timer der elevene lærte seg løkker eller funksjoner.

De elevene som kom i gang med programmeringen, jobbet de engasjert. Vi observerte likevel en del ventetid, ved at elever ventet på hjelp fra læreren da koden deres ikke virket. For eksempel hadde noen elever glemt å lukke en parentes, `input("skriv et tall"`, noen hadde skrevet variabeltilordningen feil vei, `3 = a`, noen hadde skrevet stor bokstav, `Print("Hei")`, og noen hadde glemt en *e*, `from turtl import *`.

Boks 13 Programmeringskonsepter (8. og 10. trinn)

I Python:

- Variabeltilordning, `a = 3`, `navn = "Lisbeth"`
- Input fra tastatur, `navn = input("Skriv navnet ditt:")`
- Kommentarer, `# fra tomme til cm`
- Regneoperasjoner, `cm = tomme * 2.54`
- Anførselstegn rundt strenger
- Typesikkerhet med 'int', `antall = int(input("hvor mange blyanter skal du kjøpe?"))`
- Å laste inn pakker, `from turtle import *`
- Å gå forover og snu seg til høyre og venstre med `turtle`-biblioteket, `forward(100); right(90)`

I Excel:

- Vilkår, `hvis(summer(B2:B5) >= 2000; 0; 199)`

Vi observerte også at elever mistet tid på grunn av utfordringer med programvaren, noe som stort sett handlet om problemer med innlogging på nettstedene de brukte som Python-fortolker. Et annet eksempel som illustrerer hvor problematisk det kunne være å komme i gang, var i en time på 10. trinn. En elev fikk ikke til å kjøre koden sin, fordi, som læreren sa, «Lingdyspanelet er over playknappen». Eleven fikk altså ikke kjørt koden fordi dysleksiprogramvaren sto i veien for knappen som kjørte koden. Verken eleven selv, medelevene eller læreren klarte å finne en måte å kjøre koden på.

Elevene hadde sjelden spørsmål som ikke handlet om å få koden til å kjøre, men noen ytterst få ganger stilte elevene spørsmål om forståelsen av programmeringskonseptene eller hvordan konseptene relateres til matematikk-konsepter. Følgende eksempel er fra klassen til Karen på 8. trinn:

> Elev: Jeg skjønner ikke dette. Skal vi skrive dette eller opprette en variabel?
> Karen: Å skrive dette er å opprette en variabel, `a = 7` oppretter variabelen.
> Elev: Nei.
> Karen: Variabel er noe som kan bli hva som helst. I programmering setter man inn, akkurat som emojiene, de er også variabler [Karen refererer her til *Smileyoppgaven*, Boks 8]. Variabler er noe man kan lagre informasjon i. Det gjør vi her: `a = 7`. Deretter, `print(navn, a)`, så printer vi det som står i `navn` og det som står i `a`.
> Elev: Men hva er oppgaven?
> Karen: Skriv inn de fire linjene der. [peker på elevens skjerm]
> Elev: Ah.

Dette utdraget viser at eleven var forvirret om betydningen av likhetstegnet (=) og ordet *variabel*, som har en relatert, men ikke identisk, betydning i matematikk og i programmering. Av alle lærer–elev-samtalene i timene med digitale veiledere, er dette det utdraget med mest faglig diskusjon om programmeringskonsepter og med mest overføringsverdi til matematikk. Stort sett handlet diskusjoner mellom elev og lærer i matematikktimene om at elever hadde glemt et anførselstegn eller liknende.

Programmering ved hjelp av regneark

Den andre typen programmering vi observerte, handlet om bruk av regneark. Dette forekom i en tredel av segmentene (31%) med programmering og programmet Excel ble benyttet i alle tilfeller. Disse segmentene ble kodet som tilstedeværelse av algoritmisk tenkning (skår 2), fordi elevene jobbet med enkle funksjoner som minner om programmeringskonsepter. For eksempel vil det å benytte fyll-verktøyet i Excel kombinert med formler, tilsvare det å skrive en løkke som itererer over cellene og det å benytte en hvis-setning er det samme i Excel som i et mer spisset programmeringsspråk som Python.

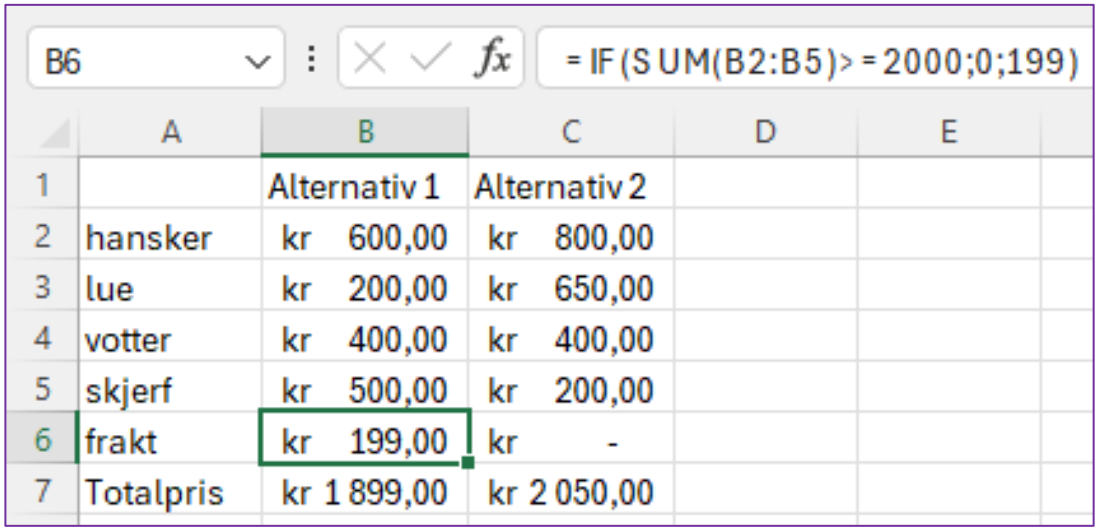
Da elevene programmerte i Excel skrev de inn formler med cellereferanser, for eksempel `=B2-C2`, og kopierte formlene til tilstøtende celler ved å benytte fyllhåndtaket, altså den svarte ruta nede til høyre i markerte celler, som ser slik ut:  . Elevene programmerte i Excel kun da tematikken dreide seg om personlig økonomi, og de brukte to forskjellige teknikker: *summer og avvik* og *hvis-setninger*. I Figur 4.12 har vi konstruert eksempler tilsvarende det vi observerte i filmene, men vi så ikke skjermene godt nok til å vite eksakt hva som sto i regnearket.

Første eksempel i Figur 4.12 viser *summer og avvik.* I en time på Vg1 (1P) analyserte elevene avviket mellom budsjett og regnskap. De skrev budsjettert beløp og faktisk beløp for ulike budsjettposter i to kolonner, regnet ut avviket og summerte totalt avvik til slutt. Andre eksempel i Figur 4.12 viser en *hvis-setning.* I en annen time på Vg1 (1P), lærte noen av elevene om hvis-setninger da de gjorde oppgaver i et Excel-kurs lærerne hadde laget. Hvis-setningen ble aktuell da oppgaven fortalte at man fikk gratis frakt hvis prisen på varene oversteg kr. 2 000. Læreren viste enkelte elever hvordan de kunne skrive inn dette i regnearket. Vi fikk inntrykk av at elevene gjorde det selv etterpå, derav skår 2. Disse to eksemplene dekker all bruk av Excel til å programmere i de observerte matematikktimene.

Figur 4.12 Funksjoner i Excel kodet som eksempler på programmering

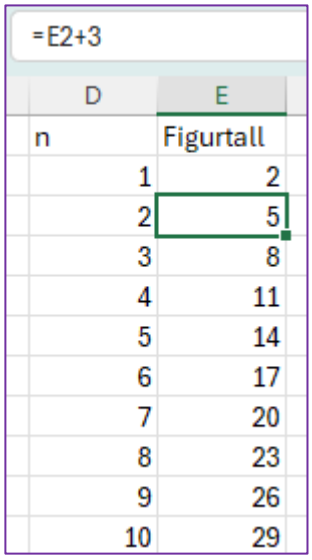
Avvik mellom budsjett og regnskap

Hvis-setninger der fraktprisen avhenger av om man har handlet for mer enn kr 2 000

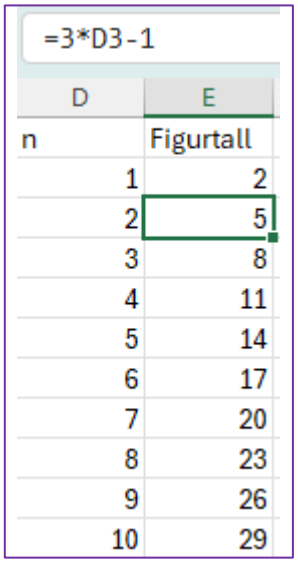


Vi observerte ingen eksempler på at lærerne oppfordret elevene til å benytte regneark i oppgaver som handlet om figurtall, selv om figurtall er en naturlig kontekst å benytte regneark i og elevene er kjent med verktøyet. I alle klasser vi observerte var det flere elever som kun så en rekursiv formulering av figurtallet, tilsvarende $a_n = a_{n-1} + 3$, og ikke en direkte formel, $a_n = 3n + 2$. Disse elevene hadde ingen måter å få til oppgavene på. I Excel kan man benytte den rekursive formuleringen, se lengst til venstre i Figur 4.13, eller en direkte formel, se i midten av Figur 4.13. Én oppgave dreide seg om summen av alle figurtallene opp til n. Denne oppgaven er også enkel å løse i Excel, se til høyre i Figur 4.13, men elevene brukte kalkulator og tastet inn sum med mange ledd.

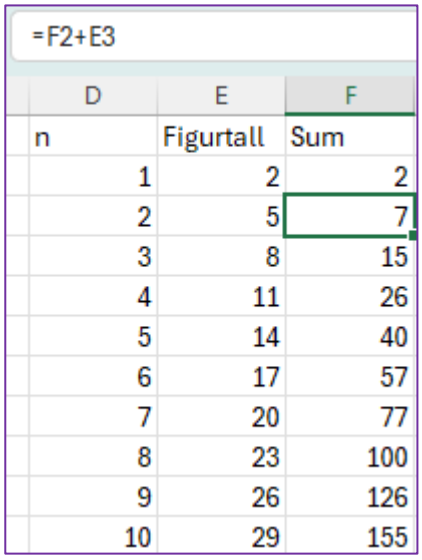
Figur 4.13 Konstruert eksempel på programmering med figurtall i Excel



Eksempel med en rekursiv formel



Eksempel med en direkte formel



Eksempel med en summekolonne

Programmering ved hjelp av graftegner

Den tredje typen programmering vi observerte handlet om bruk av graftegner, noe som forekom i en femtedel (19 %) av segmentene med programmering. Den eneste graftegneren som ble benyttet var GeoGebra, og den ble benyttet svært ofte. På alle trinn og hos de fleste lærere var graftegneren oppe minst én gang i løpet av filmingen. Stort sett ble GeoGebra benyttet til å tegne funksjoner og finne noen skjæringspunkter, og dette er ikke kodet som programmering. Vi kodet bruk av GeoGebra som programmering når elevene skrev inn funksjoner som benytter seg av en syntaks man finner igjen i programmeringsspråk. I alle segmenter som viste programmering ved hjelp av GeoGebra ble det kodet med skår 2, øving på syntaks. Vi observerte tre timer på tiende trinn, der læreren viste elevene hvordan de kunne innskrenke definisjonsområdet til en funksjon med en *hvis-setning*, se Boks 14.

Boks 14 Programmering i GeoGebra (10. trinn)

```
f(x) = hvis(x <= 2, 3x+2)
```

Programmering i oppgaver eller som verktøy

Den siste typen programmering vi observerte i disse timene, handlet om programmeringsoppgaver som dreide seg om noe mer enn kun å lære syntaks. Dette forekom i 3 timer og 4 segmenter av timene med programmering (15 % av segmentene). Dette var de eneste timene som forutsatte at elevene kunne noe programmering og skulle benytte det til noe.

En av timene var i Karens klasse på 8. trinn, som ble holdt som hjemmeundervisning under Covid-19. Vi filmet timen som ble gjennomført i Teams. Den første oppgaven som ble gitt var oppgaven *Finn feilen i koden*, se Boks 12. Den andre oppgaven var *Antall døgn i et sekund*, se Boks 15.

Boks 15 Antall sekunder i et døgn (8. trinn)

Denne oppgaven hadde tre deler: a) forklare hva følgende program gjør, b) finne ut av hva som blir skrevet til skjerm, og c) endre programmet slik at det printer ut antall sekunder i et døgn:

```
timer = 24
minutter = 60
døgn = minutter * timer
print(døgn)
```

Den siste oppgaven i Karens klasse handlet om å skrive en algoritme for hvordan man kan omregne en vilkårlig lengde som er oppgitt i tommer til cm, se *Omregning* i Boks 16. Basert på videoopptakene gjort i Teams fra hjemmeundervisningen, klarte elevene stort sett disse oppgavene. Elevenes utfordringer handlet om å omgjøre det de skrev inn i programmet (input), til et tall de kunne bruke i utregninger. Læreren løste dette ved å gjøre om inputen til et heltall, `int(input(«hvor mange tommer»))`.

Boks 16 Omregning (8. trinn)

Skriv en algoritme som tar inn et antall tommer fra brukeren og regner om til centimeter.

En annen time var i Sanders klasse på 10. trinn, der de jobbet med digitale veiledere. To av elevene fikk en oppgave med figurtall. De hadde funnet et direkte uttrykk for figurtallet: $f(n) = n(n+1)$, men skulle lage et program som lot dem regne ut figurtallet for en vilkårlig verdi av n. Elevene klarte ikke dette selv, men fikk en god del hjelp av læreren skrev de

```
nummer = int(input(«Rektangeltall nummer:»))
print(nummer * (nummer + 1))
```

I disse timene benyttet elevene programmering kun som en kalkulator, men i den siste timen ble programmering forsøkt benyttet som et verktøy. I *Lyspæreoppgaven* på Vg1 (1P-Y) (Boks 9) var det én gruppe på tre elever som forsøkte å programmere en løsning. Læreren ble glad for dette og oppfordret dem til å prøve. Elevene fikk skrevet 5–6 linjer før det stoppet opp. Flere elever samlet seg foran skjermen deres, ogtil slutt sto seks elever og diskuterte hvordan de kunne løse oppgaven. Elevene diskuterte i 10 minutter og hadde 10–12 linjer kode før læreren avbrøt arbeidet med oppgaven og gikk videre i undervisningen. Heller ikke disse interesserte elevene unngikk dataproblemer, og like før læreren avbrøt arbeidet sa eleven som satt ved tastaturet: «Ah, jeg har ikke installert Python på denne PCen», så de fikk aldri forsøkt å kjøre koden. Læreren oppfordret elevene til å «skrive det ut med programmering til neste gang».

I Boks 17 viser vi en måte å løse lyspæreoppgaven på ved bruk av programmering. Vi har konstruert denne løsningen for å vise programmeringsnivået elevene må ha for å løse en litt vanskelig matematikknøtt. Da blir det tydelig at nivået som undervises er langt lavere enn dette. Løsningen består av ni linjer kode, hvis man ikke teller med kommentarene, og koden benyttet kun programmeringskonsepter fra læreplanen LK20 i matematikk på 6. trinn, bortsett fra modulus-operatoren % som gir resten ved heltallsdivisjon.

Boks 17 Lyspæreoppgaven ved bruk av programmering

```
# For hver lyspære, tell hvor mange ganger den blir slått av og på
for bulb_nr in range(1, 51):
    button_pushes = 0

    for person_nr in range(1, bulb_nr + 1):
        if(bulb_nr % person_nr == 0):
            button_pushes += 1

    # Print om pæra er på eller av når alle personene
    # har gått forbi.
    if(button_pushes % 2 == 0):
        print("Lyspære", bulb_nr, "-")
    else:
        print("Lyspære", bulb_nr, "PÅ")
```

Koden gir følgende output:

```
Lyspære 1 PÅ
Lyspære 2 -
Lyspære 3 -
Lyspære 4 PÅ
Lyspære 5 -
Lyspære 6 -

...
```

4.6 Oppsummering

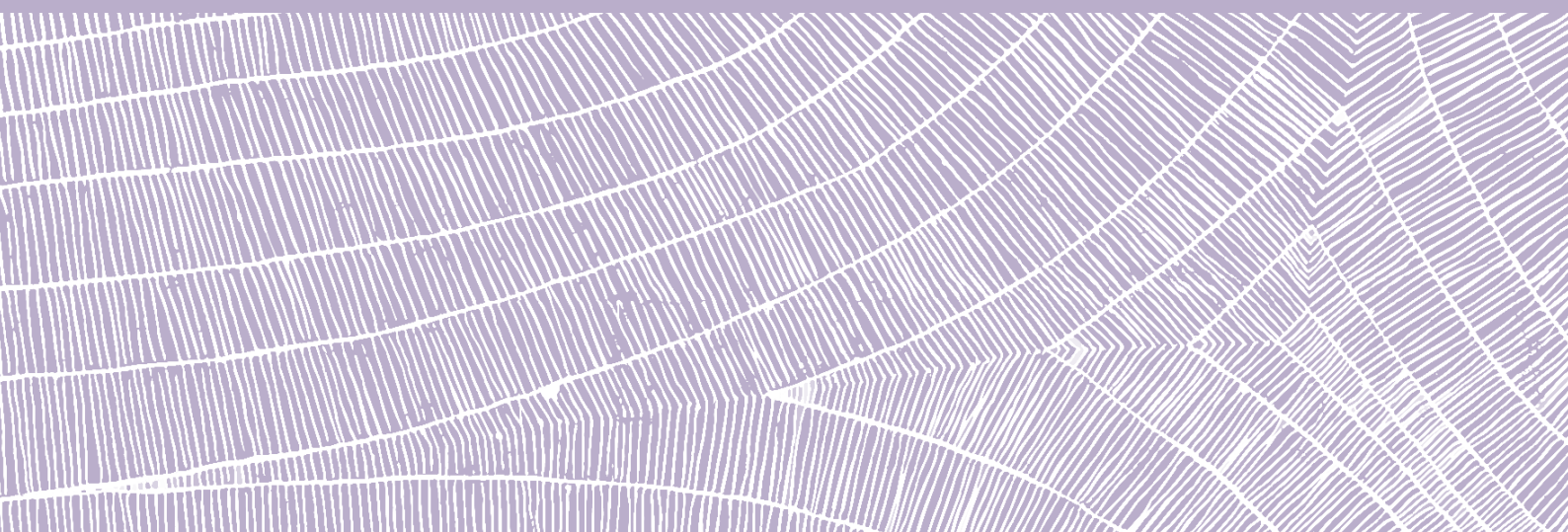
Vi finner i liten grad forskjeller mellom trinn og fag basert på skåringene. *Algoritmisk problemløsning* var til stede i halvparten av segmentene i den filmede undervisningen (52 %), mesteparten av tiden som skår 2, som betyr at elevene utfører kjente algoritmer eller arbeider med forståelsen av dem. En tredel av segmentene med algoritmisk problemløsning var på skår 3 og 4, som betyr at elevene jobber med problemløsning som benytter seg av dekomponering, mønstergjenkjenning og logikk, og kanskje lager en løsning som kan automatiseres. Den algoritmiske problemløsningen dreide seg oftest om figurtallsoppgaver, som vi observerte i alle matematikkfagene (8. og 10. trinn, 1T, 1P og 1P-Y), men også mange problemløsningsoppgaver krevde algoritmisk problemløsning. Det var også en klasse på 8. trinn som jobbet med algoritmisk problemløsning da de utviklet reglene for potensregning sammen.

Algoritmiske arbeidsmåter var til stede i litt under halvparten av undervisningen (40 %). I en fjerdedel av tiden på skår 3 og 4, som betyr at elevene i tillegg til fikling bedrev feilsøking eller skaping. Spesielt var det gode eksempler på algoritmiske arbeidsmåter i undervisningen der elevene skapte noe, fordi elevene samarbeidet, holdt ut lenge, var analytiske og feilsøkende, og fiklet i utstrakt grad.

Programmering forekom i 7 % av segmentene, eller i 3 % av timene hvis vi ikke teller med programmeringsliknende syntaks i Excel og GeoGebra. I Excel benyttet elevene vilkår og i GeoGebra skrev de inn kommandoer for å begrense området en funksjon skulle tegnes på. Resten av programmeringen var i Python, der mesteparten av programmeringen dreide seg om at elevene fulgte *digitale veiledere*.

DEL 5

Algoritmisk tenkning og programmering fra et lærerperspektiv



5 Algoritmisk tenkning og programmering fra et lærerperspektiv

I denne delen ser vi nærmere på hva lærerne på 8. trinn, 10. trinn og Vg1 mener algoritmisk tenkning og programmering innebærer i matematikk. Først ser vi på lærernes loggføring av algoritmisk tenkning umiddelbart før og etter den filmede undervisningen (5.1). Deretter presenterer vi lærernes perspektiver på algoritmisk tenkning og programmering fra individuelle intervjuer (5.2). Til slutt oppsummerer vi denne delen (5.3).

5.1 Lærere loggfører algoritmisk tenkning

I forkant av den filmede undervisningen ba vi lærerne loggføre hvorvidt de planla å inkludere algoritmisk tenkning i undervisningen (logg 1). I etterkant av undervisningen ba vi dem loggføre om de mente at de faktisk hadde jobbet med dette i timen (logg 2). Loggene er et viktig supplement til videoobservasjonene fra klasserommet, fordi de bidrar til å kontekstualisere den filmede undervisningen. Videre brukes de som utgangspunkt for lærerintervjuene.

Tabell 5.1 Lærerlogg 1 og 2 om algoritmisk tenkning før og etter timen

		Matematikktimer	Algoritmisk tenkning (AT)	
Trinn	Fag	Totalt antall timer filmet	Planlagt arbeid med AT (logg 1)	Faktisk arbeid med AT (logg 2)
8. trinn	Matematikk	36	11	25
10. trinn	Matematikk	25	9	18
Vg1 SF	Matematikk 1T	16	12	10
	Matematikk 1P	20	4	12
Vg1 YF	Matematikk 1P	12	5	8
	Matematikk 1P-Y	8	0	4
	Totalt	**117**	**41**	**77**
Gjennomsnitt av 117 timer		**100 %**	**35 %**	**66 %**

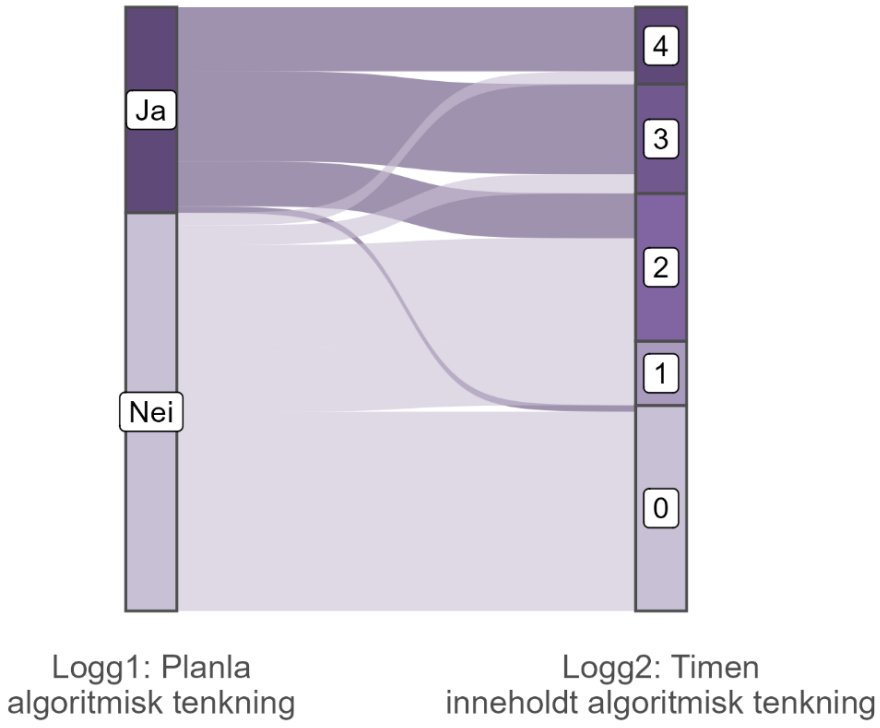
Tabell 5.1 viser at lærerne planla (logg 1) å inkludere algoritmisk tenkning i undervisningen i alle matematikkfagene på alle trinn, noe de etter undervisningen bekreftet de at de hadde gjort (logg 2). Loggene viser at lærerne enten hadde jobbet med algoritmisk tekning i de timene de hadde planlagt å gjøre det (1T) eller i flere timer (8. trinn, 10. trinn, 1P, 1P-Y). Til sammen rapporterte lærerne at de planla å jobbe med algoritmisk tenkning i 35 % av timene, men at de hadde endt opp med å gjøre det i 66 % av timene. I logg 2 ba vi lærerne om å indikere *i hvilken grad* de hadde jobbet med algoritmisk tenkning.

Tabell 5.2 Lærerlogg 2: grad av algoritmisk tenkning i matematikkundervisningen

Antall timer lærerne rapporterte å ha inkludert algoritmisk tenkning i fagene på 8. trinn, 10. trinn og Vg1			I hvilken grad de jobbet med algoritmisk tenkning (antall timer)			
			Litt	I noen grad	I stor grad	Mye
8. trinn	Matematikk	25	4	10	6	5
10. trinn	Matematikk	18	1	7	5	5
Vg1 SF	Matematikk 1T	10	0	4	6	0
	Matematikk 1P	12	4	4	2	2
Vg1 YF	Matematikk 1P	8	2	3	3	0
	Matematikk 1P-Y	4	2	2	0	0
Total respons		**77**	**13**	**30**	**22**	**12**
		(66 %)	**35 %**		**31 %**	

Tabell 5.2 viser at når lærerne arbeidet med algoritmisk tenkning i matematikkundervisningen, fordelte det seg jevnt på *litt* og *i noen grad* (35 %) samt *i stor grad* og *mye* (31 %). Dette tyder på at lærerne tenkte at algoritmisk tenkning var sentralt i en tredel av timene vi filmet. Figur 5.1 viser sammenhengen mellom på venstre side, de timene lærerne ikke hadde planlagt å jobbe med algoritmisk tenkning (0) og de timene som var planlagt for det (1) og på høyre side, i hvilken grad timene faktisk hadde inneholdt algoritmisk tenkning litt (1), i noen grad (2), i stor grad (3) og mye (4). Oppsummert viser dette at lærerne gjorde det de hadde planlagt i to tredeler av timene, mens de i den midterste tredelen hadde inkludert algoritmisk tenkning litt (1) eller i noen grad (2) selv om det ikke var planlagt (0).

Figur 5.1 Sammenheng mellom lærerlogg 1 og 2 om algoritmisk tenkning



I loggene førte lærerne inn hvilket tema de hadde jobbet med i de filmede timene. Det var relativt stor bredde i de temaene lærerne rapporterte at de hadde jobbet med i forbindelse med algoritmisk tekning (se Boks 18), temaer som samsvarte med det vi observerte (se Del 4):

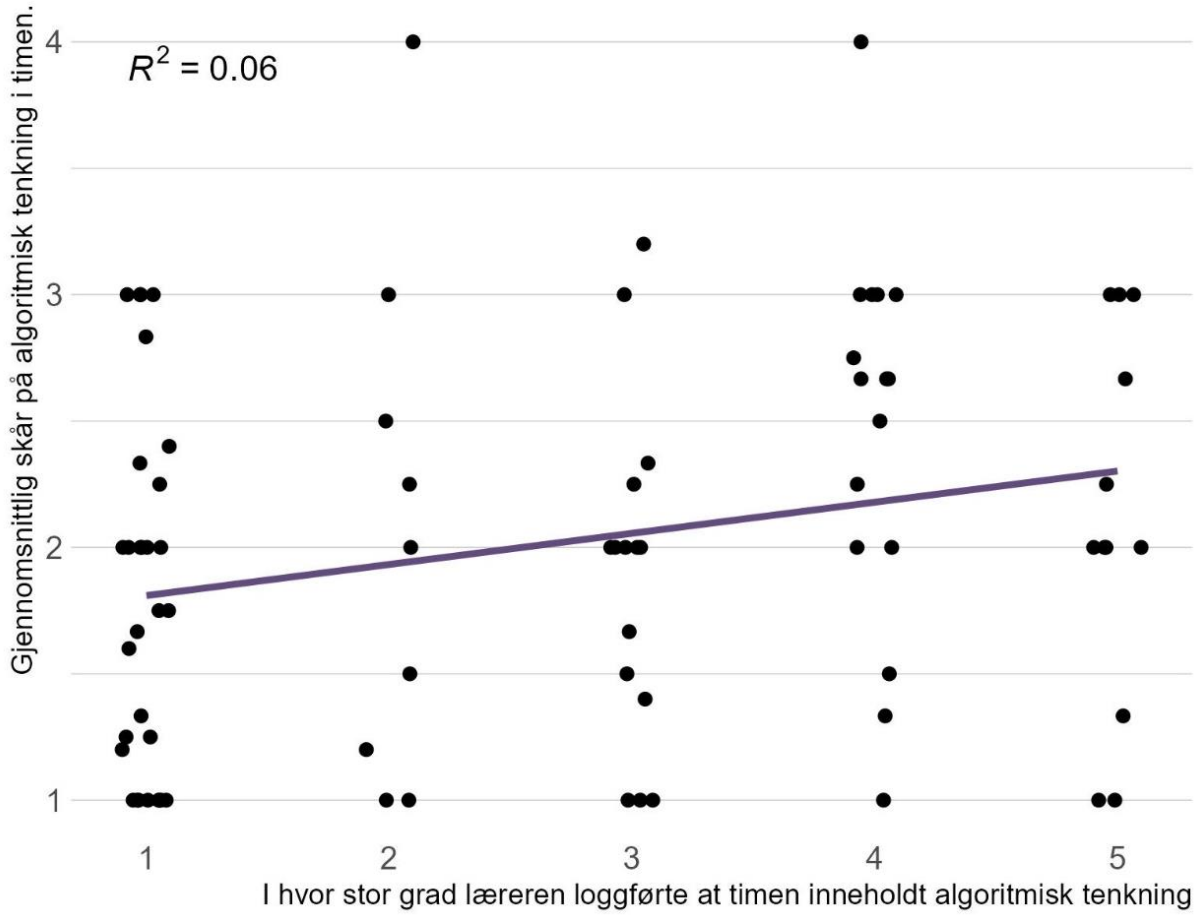
Boks 18 Temaer loggført for algoritmisk tenkning

De timene lærerne loggførte å ha jobbet med algoritmisk tenkning handlet om følgende temaer:

- 8. trinn: algebra, brøk, mønstre, potenser, problemløsning og Python
- 10. trinn: funksjoner, modellering og programmering
- Vg1 1T: figurtall, modellering og regresjon
- Vg1 1P: Excel og måleenheter
- Vg1 1P-Y: former, figurer, formler; mva. og økonomi; spenning og tolking av tall

Figur 5.2 viser at det var lite samsvar mellom logger som rapporterte at timen inneholdt algoritmisk tenkning og vår observasjon av algoritmisk tenkning ved hjelp av observasjonsmanualen. Det manglende samsvaret mellom lærernes logger og våre observasjoner tyder på at lærerne ikke forstår algoritmisk tenkning på samme måte som oss og læreplanen, noe som bekreftes i lærerintervjuene (Del 5).

Figur 5.2 Samsvar mellom lærerlogg og observasjoner av algoritmisk tenkning



5.2 Lærernes beskrivelser av algoritmisk tenkning og programmering i intervjuene

Boks 19 Temaer vi lagde basert på intervjuene

Disse oppfatningene ble uttrykt av de intervjuede matematikklærerne, uavhengig av trinn og fag (der antall ikke er oppgitt ble det nevnt av 1–4 lærere):

Lærernes forståelse av begrepet algoritmisk tenkning:

- Algoritmisk tenkning er programmering og det å lage oppskrifter (8 lærere)
- Algoritmisk tenkning er å forstå kjente algoritmer (6 lærere)
- Algoritmisk tenkning er å utføre algoritmer (5 lærere)
- Algoritmisk tenkning er en problemløsningsmetode (4 lærere)
- Vet ikke hva algoritmisk tenkning er (1 lærer)

Holdninger til programmering:

- Programmering er en viktig kompetanse i arbeidslivet
- Programmering gir muligheter i undervisningen
- Programmering kan gjøre matematikken mer motiverende

Hvor mye lærerne programmerte med klassen:

- Jeg har programmert mindre enn planlagt (11 lærere)
- Jeg skal programmere senere i skoleåret (7 lærere)
- Jeg har programmering som et eget tema på slutten av skoleåret
- Jeg ønsker å undervise programmering så lite som mulig
- Jeg er glad eksamen ble avlyst, så mangel på programmering ikke går ut over elevene

Utfordringer med programmering:

- Jeg vet ikke hva elevene skal kunne (6 lærere)
- Vi må prioritere andre kompetansemål og har ikke tid (6 lærere)
- Lærerne kan ikke det de skal fra tidligere skoleslag (5 lærere)

Hva lærerne syntes om egen programmeringskompetanse:

- Jeg kan ikke nok programmering (14 lærere)
- Jeg er ikke god, men kan nok til å prøve det ut i undervisningen (2 lærere)
- Jeg er god nok i programmering (1 lærer)

Hvor lærerne hadde lært programmering:

- Jeg har blitt tilbudt kurs
- Jeg har funnet ut av noe sammen med kollegaer
- Jeg har en kollega som fungerer som ressursperson på skolen
- Jeg lærer meg å programmere sammen med elevene

Intervjuene ble gjennomført etter filmingen, og totalt ble 17 lærere intervjuet: 13 lærere i 2021–22 (8. trinn og Vg1) og 4 lærere i 2023–24 (10. trinn). På Vg1 bidro intervjuene med et komparativt perspektiv ved at fire av lærerne på studiespesialiserende program underviste både 1T og 1P (Camilla, Gloria, Gustav, Rasmus) og ved at én lærer på yrkesfag underviste både 1P og 1P-Y (Emily). På ungdomstrinnet bidro intervjuene med et komparativt perspektiv over tid, ved at to av lærerne ble filmet og intervjuet både på 8. trinn og 10. trinn (Karen, Sander). Lærerne fikk åtte spørsmål, og selv om de dreide seg utelukkende om algoritmisk tenkning (se Boks 5), snakket lærerne hovedsakelig om programmering (Boks 19). I intervjuene handler derfor første del om algoritmisk tenkning, mens de andre delene dreier seg om programmering.

5.3 Lærernes forståelse av begrepet algoritmisk tenkning

Vi spurte alle lærerne om hvordan de oppfattet begrepet algoritmisk tenkning. Lærerne uttrykte fire forskjellige oppfatninger av begrepet, oppsummert i Figur 5.3.

Figur 5.3 Lærernes oppfatninger av algoritmisk tenkning

a. *Algoritmisk tenkning er en problemløsningsmetode (4 lærere)*

«Ja, jeg tenker at algoritmisk tenkning handler om å dele opp problemer. Eller dele opp oppgaver eller regnestykker sånn at ja, få mer oversikt, kunne se mønster i ting. Vi har jobbet med veldig mange strategier eller løsningsmetoder som elevene har jobbet med.» (Olivia, 8. trinn)

b. *Algoritmisk tenkning er programmering og det å lage oppskrifter (8 lærere)*

«Jeg har vel ikke brukt det så mye til nå med de jeg har i åttende nå, men jeg forbinder det litt med programmering, lage oppskrifter, så det tenker jeg vi kommer til ganske snart over jul. Ja, lære dem om det begrepet [algoritmisk tenkning], kanskje starte med at man må ikke gjøre det på en datamaskin, men å lære og lage oppskrifter.» (Hannah, 8. trinn)

c. *Algoritmisk tenkning er å utføre algoritmer (5 lærere)*

«Ja, altså for meg så er jo [algoritmisk tenkning] en regneregel eller et eller annet som på en måte funker for å komme frem til et svar som du lærer. […] Jeg tenker jo at vi jobber med det ganske mye når vi jobber med algebra, for der er det jo ganske mye algoritmer du må … Altså, for jeg tenker jo på algoritmer som konkrete regneregler.» (Petter, 10. trinn)

d. *Algoritmisk tenkning er å forstå kjente algoritmer (6 lærere)*

«Jeg legger i det … vi legger jo opp til at de skal på en måte lære seg til å se mønstre og hvorfor ting er som det er, og forklare hvorfor, for så å ende opp med type denne algoritmen selv.» (Sara, 8. trinn)

To av oppfatningene (a og b) samsvarer mer eller mindre med LK20. Fire av lærerne hadde en forståelse av at *algoritmisk tenkning er en problemløsningsmetode*, og to av disse lærerne nevnte dekomponering spesifikt, som i å «bryte ned problemer». Åtte av lærerne hadde en forståelse av at *algoritmisk tenkning er programmering og det å lage oppskrifter*. Selv om dette samsvarer med definisjonen av algoritmisk tenkning i litteraturen og læreplanen, er beskrivelsen vag og inkluderer ingen av begrepene fra UDIRs algoritmiske tenker (se Figur 2.2). De lærerne som hadde oppfatninger av algoritmisk tenkning som samsvarte med læreplanen likte begrepet og uttrykte at det bygget opp under fagets kjerne.

Da Olivia ble spurt om algoritmisk tenkning hadde gjort at matematikk undervisningen hadde endret seg, svarte hun:

> Ja, det tror jeg absolutt. Det henger nok litt sammen med utforsking da, men det å gjøre ting mindre komplekst og legge opp til situasjoner der elevene får jobbe med mindre deler av et større problem og sette ting sammen, og lage kanskje ofte mening sammen som klasse da, men man jobber individuelt med mindre deler av et problem, for eksempel. (Olivia, intervju, 8. trinn).

To av oppfatningene (c og d) var ikke i samsvar med LK20. Fem lærere hadde en forståelse av at *algoritmisk tenkning er å utføre algoritmer*, noe som bryter med tanken bak matematikklæreplanen. Seks av lærerne sa at *algoritmisk tenkning er å forstå kjente algoritmer*. Dette stemmer ikke med matematikklæreplanens begrep om *algoritmisk tenkning*, men det samsvarer med andre mål i læreplanen om å bygge forståelse og ikke bare følge regler og utføre prosedyrer. De lærerne som oppfattet algoritmisk tenkning som å følge algoritmer uttrykte at den motarbeidet fagets kjerne:

> Jeg opplever en algoritme som en, det er jo disse prosedyrene som jeg føler at har vært hemskoen til matematikk lenge da, at man jobber så mye med prosedyrer at man glemmer matematikken. Så jeg har vært veldig opptatt av å flytte det bort fra prosedyrer da, og heller få de til å formulere ting selv. (Petter, intervju, 10. trinn)

Generelt, da vi spurte lærerne om algoritmisk tenkning, nølte flere av dem eller lo usikkert og trakk på ordene før de forklarte hva begrepet betyr. Én lærer sa at han var usikker på hva som mentes med begrepet. I intervjuene kom lærerne nesten alltid inn på programmering ved spørsmål om algoritmisk tenkning. Selv om lærerne ble spurt om holdninger til og kunnskaper om algoritmisk tenkning, fikk vi få svar om dette og i stedet mange svar om programmering. Ingen av lærerne nevnte de algoritmiske arbeidsmåtene, selv ikke de fire lærerne som definerte algoritmisk tenkning i tråd med læreplanen. Det virker altså som om begrepet *algoritmisk tenkning* ikke har satt seg hos norske lærere, og i hvert fall ikke begrepet *algoritmiske arbeidsmåter*.

De neste delkapitlene handler derfor om programmering som lærerne hadde mer å fortelle om enn algoritmisk tenkning.

5.4 Lærernes holdninger til programmering

Lærerne uttrykte positive holdninger til at programmering var kommet inn i skolen, og uavhengig av trinn og fag, mente alle at programmering hører hjemme i matematikkfaget. Flere lærere vektla at programmering var en viktig kompetanse i arbeidslivet:

> Jeg tenker jo at det er kjempemulighet og at når ting blir såpass digitale og det er så fokus på det i samfunnet så er det jo kjempeviktig at de lærer det, det er jo så mange jobber innen det [programmering], så det er kjempehjelpemiddel så det er knallbra at det [programmering] har kommet inn. (Ine, intervju, Vg1)

Andre lærere mente at programmering ga flere muligheter i undervisningen, og flere påpekte at elevene syntes programmering gjorde matematikkfaget mer motiverende:

> Det jeg håper da, er at programmering på sikt kan gjøre folk mer interessert i matematikk, at de sier «du kommer deg ingen vei i programmeringsverdenen uten matematikk». (Sander, intervju, 10. trinn)

> Hvis man får trygghet på at det finnes mange løsninger, men at det finnes noe mer effektive og man kan sette opp et system, hvis man får den tryggheten i klasserommet, så tror jeg det kan være ganske kult å jobbe mot. [Man] får en slags norm, at når man har jobbet en stund, la oss si at man jobber med figurtall da, at det er en etablert sannhet i vårt klasserom at vi skal komme frem til en generell formel der, hvis det blir akseptert da, at man skal jobbe for å effektivisere det, så er det mange muligheter. Har ikke klart for meg hvordan de mulighetene skal dukke opp ennå da. Noen tema, yes; andre, ikke helt. (Lukas, intervju, Vg1)

Hos mange lærere var de positive holdningene blandet med negative tanker om hvordan programmering var kommet inn i læreplanen og hva slags opplæring de hadde blitt tilbudt. Dette beskrives nedenfor.

5.5 Hvor mye lærerne programmerte med klassen

Elleve av lærerne sa de ikke hadde hatt programmering med klassen sin, og for de fleste ble det mindre enn planlagt. Sju lærere sa at de skulle programmere senere i skoleåret, mens to lærere kommenterte at de hadde glemt å jobbe med det. Noen fortalte også at de ikke underviste programmering sammen med annen matematikk, men som et eget tema:

> Det [programmering] har jeg tenkt å jobbe med etter hvert. (Hannah, intervju, 8. trinn)

> Ja, så det vil jo da, da vil programmering bli en ting som går på siden, nettopp fordi at jeg kom så sent i gang med det i den klassen her. […] Jeg bare skyver det foran meg, som sikkert mange andre lærere også gjør, også krysser fingra for at vi kanskje finner en løsning. At han som kan dette her faktisk kan ta jobben og gjøre det best mulig for elevene, så bytte litt klasser og sånn, det hadde vært det beste. Gjøre hverandre gode, da. (Sander, intervju, 10. trinn)

> Vi putter jo ikke programmering inn i alle temaer, liksom. Og det blir på en måte mer enn sånn «nå skal vi jobbe med programmering». (Petter, intervju, 10. trinn)

> Nei, vi har hatt lite om det foreløpig. Ehm, og så har vi tenkt å bruke tiden frem mot sommeren på det, egentlig, for da har vi på en måte vært gjennom mange av temaene og da kan du bruke de temaene inn i programmeringen. Så vi har på en måte vridd det om sånn, så foreløpig så har vi gjort lite der, ja. (Ine, intervju, Vg1)

> Altså, det er, ehm [...], nei altså, utfordringen er det at det er en ting at jeg rett og slett bare har glemt det litt på en måte. (Rasmus, intervju, Vg1)

Lærerne pekte også på konkrete opplegg de hadde gjennomført. På ungdomstrinnet nevnte lærerne et introduksjonsopplegg fra *Lær Kidsa Koding*[10] og programmeringsopplegget til nettlæreverket Kikora. I videregående skole nevnte lærerne å ha hatt en introduksjon til Python i 1T. Videre snakket to av lærerne om sammenhenger mellom programmeringen de hadde gjort og matematikkfaget, men forklaringene var noe vage:

> Ja, men det er jo igjen, at man blir jo bevisst på, for at den turtle-en, når du sier at den skal gå til høyre og du velger for eksempel 60 grader så går den 120, så tenker du, hvorfor gjør den det? Går for eksempel komplementær vinkel. Men hvilken del av vinkelen er det han ser som høyre og venstre? Skjønner du? Det vet jeg ikke. Da sier jeg til dem det må du bare prøve og teste. Er det ikke 60, er det 120. Er jo bare å bytte liksom, hvis du skjønner. Så jeg er der at jeg selv må prøve å feile litt på hva den gjør. (Sander, intervju, 10. trinn)

> For som jeg sa til dem, at det er også f.eks. å modellere med data, det er jo fantastisk bra. For eksempel, vi hadde en sånn oppgave her om dagen i matte som, med modellering, det var noe sånn som du trengte egentlig ikke selv å skrive. Du måtte oversette en algoritme som var skrevet. Og så sier du at dette er gjort, det er bare å skrive inn i Python. Og så får du alle svarene med en gang. Men hvor mange som tenker på det? Det er ikke så mange ennå. (Karen, intervju, 10. trinn)

På bakgrunn av egen kompetanse og hvor lite de har endt opp med å undervise var lærerne glade for at eksamen var avlyst i 2022:

> Sånn sett så er jeg litt glad for at eksamen er avlyst [i 2022], så nå kan jeg på en måte prøve med disse og se hvordan det går uten at jeg føler at en ikke har gitt elevene den læringen de skal ha da på en måte. (Ine, intervju, Vg1)

[10] www.kidsakoder.no

Skoleåret 2023–24 fortalte 10. trinns lærere som vi intervjuet at de ønsker å undervise så lite programmering som mulig, og definerte dette ut fra hva som var kommet på eksamen:

> Ja, vi venter litt og ser, ja. Så gjør vi liksom et minimum opp mot eksamen, fordi vi vet at det kommer til å komme på eksamen. Men vi har liksom ikke ... Ja, vi gjør det som et bare minimum. […] Og hvis de skal lære det skikkelig, så må vi bruke mye tid på det. Og siden vi ikke vet helt ennå, så vil vi ikke bruke de ... Hvis vi skulle lære noe skikkelig, så må vi bruke ti timer på det. Og de ti timene har vi ikke puttet inn noe sted. Så vi er veldig på et minimum der, og det gjør at vi ... Ja, jeg vil si at vi sitter veldig på gjerdet og venter, og ser hva som egentlig kreves. (Ludvig, intervju, 10. trinn)

Sander forteller også at han var glad det ennå bare var gitt flytdiagrammer og tolking av blokkprogrammering til eksamen.

> Så det er litt dumt, og det er jo ikke deres [elevenes] skyld, det er jo min skyld, på en måte. For jeg burde jo ha tatt meg selv i nakken og blitt bedre på dette her tidligere, på et vis, hvis du skjønner. Så der får man bare skylde seg selv. Men vi kommer i mål, altså. Jeg vet at vi lander med det som kreves til at vi skal kunne fikse til eksamen, tror jeg. Det har vært blokkprogrammering så langt. Krysser fingrene for at det er der fortsatt, én sesong til. Tolke enten en sånn flytskjema eller en blokkprogrammering. […] Men tipper at det, som en del av repetisjonen frem mot en eksamen, at man kan se på det [programmering]: «Oi, her har vi delelighet da kan vi se på hvordan Python kan løse delelighet. Og her har vi terningkast. Skal vi simulere terningkast, så her har vi et program. Og husk, ta godt vare på dette her, ha det liggende på skrivebordet, slik at dere kan laste det opp og vise hvor flinke dere er i programmering når den tid [eksamen] kommer.» (Sander, intervju, 10. trinn)

Kun én lærer, Sander på 10. trinn, sa at han nå jobbet jevnlig med programmering:

> Ja nå gjør jeg det hver uke, hver fredag, første halvtime. Da er det i hvert fall programmering. Så det vil man kunne se frem til sommeren (Sander, 10. trinn).

5.6 Utfordringer med programmering

Lærerne uttrykte flere utfordringer med å inkludere programmering i matematikkundervisningen. Tre utfordringer skilte seg ut. For det første sa lærerne at de ikke visste hva elevene skulle kunne. For det andre fortalte de at elevene ikke kunne det de skulle fra tidligere og til slutt at lærerne manglet tid til å bruke på programmering. Seks lærere skjønte ikke hva elevene skulle kunne:

> Grunnen til det, jeg er jo egentlig [at jeg er] veldig glad i programmering, jeg har også gjerne brukt det mye. Men inntil videre, så vet vi ikke hvor mye elevene må kunne av det. (Ludvig, intervju, 10. trinn)

En av lærerne sammenliknet situasjonen med innføringen av graftegner (GeoGebra) ved revideringen av LK06 i 2012:

> For jeg lærte jo aldri GeoGebra på lærerutdanninga. Og så ble det innført når jeg begynte å jobbe. Og da synes jeg de første årene, synes jeg det var veldig vanskelig å undervise i GeoGebra. Fordi det var ikke helt klart hva er det du forventer at elevene skal. Og så var jeg på ett kurs, og da var det en som bare var ganske tydelig. (...) Han sa bare, lærer du elevene dette, dette og dette, så går det helt fint til eksamen. Og fra da så føler jeg meg veldig god på GeoGebra. (Petter, intervju, 10. trinn)

Fem lærere kommenterte at elevene ikke kunne det de skulle fra tidligere skolegang, og at situasjonen kanskje ville bedre seg når elevene begynte å kunne noe fra før av:

> Og det jeg sier, at det tror jeg at vi trenger litt mer tid på oss. Da disse elevene får litt mer i utgangspunktet. For jeg spurte dem i åttende klasse om det var noen som har vært borti blokkprogrammering. Da var det kanskje 3 av 25. Da betyr det at det jeg skulle bygge på, den var ikke der. Da må vi ta det et annet sted. (Karen, intervju, 10. trinn)

Og seks lærere kommenterte at tid var en utfordring:

> I åttende nå, vi fant ut i matte at brøk det må vi ta grundig. Det skal man jo egentlig ikke ta grundig i åttende, fordi det skal de kunne fra før, men har et halvt år gått og vi har holdt på med tall og tallregning. Jeg la en plan og tenkte sånn i uke 47/48, da begynner vi med algebra, men det blir januar før vi begynner med det. Så ja, det med tid det er alltid en knapp ressurs. (Hannah, intervju, 8. trinn)

5.7 Lærernes egen programmeringskompetanse

I intervjuene ble lærerne synlig engasjerte da de fikk spørsmål om sin egen kompetanse, og 14 av 17 lærere fra alle trinn og fag mente de at de ikke kunne nok programmering.

> Jeg er som veldig mange andre mattelærere at vi trenger mer kursing på programmering. Og kanskje ikke bare kursing, men egentlig også ... Altså... når man setter i gang en sånn ting i en læreplan, så burde jo nesten alle mattelærere gått et år til på skole for å lære det. Altså det er jo ... det er jo helt hårreisende. Skulle man liksom ... ja, innføre et nytt ... skulle norskfaget ha også dansk da? Det er jo veldig rart hvis alle norsklærere også skulle begynne å undervise dansk uten å kunne noen ting om det. Så skal de bare lære seg selv det, liksom? Ja, eller liksom noe enda rarere, kinesisk liksom. Det er jo helt utrolig at vi har gått med på det! (Sander, intervju, 10. trinn)

Mangelen på kompetanse førte også til at flere lærere ikke så mulighetene med programmering i undervisningen:

> Nei ... men jeg er ikke god på det liksom! (Petter, intervju, 10. trinn)

Tre av de 17 lærerne følte seg klare til å undervise i programmering, men to av dem sa at de kunne nok til å prøve det i undervisningen, selv om de ikke oppfattet at de hadde god programmeringskompetanse:

> Ja, men jeg er ikke redd for å hive meg utpå. Jeg synes jo det er spennende og gøy å prøve og feile. Det syns jeg er helt fint så derfor har jeg prøvd programmering litt i det små på en måte, og så har jeg liksom tatt disse kursene og nå tenker jeg, ok, nå da kan jeg liksom hjelpe de mer (ler) nå etter at jeg har tatt det, så nå føler jeg liksom jeg er klar. (Ine, intervju, Vg1)

Selv de lærerne som hadde hatt programmeringskurs på universitetet og som hadde realfaglig bakgrunn nevnte at de hadde elever som var flinkere enn dem selv, eller at de prøvde å være ett skritt foran elevene til enhver tid:

> Jeg tenker sånn, det [programmering] er jo kjempeviktig. Nå har vi meldt oss på sånn programmeringskurs og jeg kan, har gjort litt Python på universitetet og sånn, men det er jo ting som jeg tenker at, jeg ville aldri stått og undervist i noe som jeg ikke følte meg kompetent i. Men jeg er heller ikke redd for å spørre elever om jeg føler at de kan mer enn meg. I hvert fall i programmering, tenker jeg sånn at det er en kjemperessurs i klasserommet å ha sånne, mange steder har de sånne superbrukere av elever som kan hjelpe til, og det tenker jeg er bare kjempefint. (Olivia, intervju, 8. trinn)

Flere lærerne uttrykte også frustrasjon over at de har måttet lære seg programmeringen selv. Mange nevnte at de var blitt tilbudt kurs, men ingen syntes kursene var tilstrekkelige:

> Så har jeg jo tatt noe programmering for ikke så lenge siden, så i fjor underviste jeg i det i valgfag og føler at jeg har bygd meg opp litt kompetanse der. Digital kompetanse ellers, det er det man har lært seg mest. Har på en måte bare måttet lære seg alle disse tingene som har kommet. (Hannah, intervju, 8. trinn)

Da kursene var utilstrekkelige, måtte lærerne finne ut av programmeringen sammen med kollegaer. Petter (intervju, 10. trinn) forklarer at de får tilbud om kurs utenfor jobben på frivillig basis og at de må selv sette seg inn i programmering. De fleste nevnte en kollega som kunne mer programmering, og fungerte som ressursperson. Lukas (intervju, Vg1) fortalte om «en kollega på jobb som har masse kompetanse innenfor programmering, som har holdt en del kurs» og Sander uttrykker godt hvor sårbart det er å måtte stole på kollega til programmeringshjelp:

> Og nå forsvinner jo den flinkeste av oss, han flytter jo hjemover til sommeren. Så da er det jo, håper jeg at de får ansatt noen da, som kan dette da, godt. Vi har noen til å ta over stafettpinnen. Det er greit og viktig at det er noen, kall det primus motorer, som skal holde en gammel lærer i ørene på dette her, for ellers så, ja. (Sander, intervju, 10. trinn)

Det er ikke kun kollegaer som fungerer som ressurspersoner. Tre lærere nevnte også at de utforsket og lærte seg programmering sammen med elevene. Karen (intervju, 10. trinn), sa «jeg lærer jo sammen med dem [elevene], på en måte, [er det] den veien å gå. Vi må finne ut av det sammen, på en måte. Det er det jeg føler da».

5.8 Sammenligning på tvers av matematikkfag og over tid

Fire av lærerne på studiespesialiserende program underviste både 1T og 1P (Camilla, Gloria, Gustav, Rasmus) og én lærer på yrkesfag underviste både 1P og 1P-Y (Emily). Lærerne nevnte ingen forskjeller mellom fagene, men de ble heller ikke spurt om det. På ungdomstrinnet fulgte to av lærerne klassene vi filmet fra 8. trinn skoleåret 2021–22 til 10. trinn skoleåret 2023–24 (Karen, Sander), og de ble intervjuet begge skoleårene.

Sander uttrykte på 8. trinn at algoritmisk tenkning var som «å følge oppskrifter», mens han på 10. trinn mente det handlet om å tenke «slik programmerere tenker» og å «se algoritmer fra forskjellige vinkler». På 8. trinn sa han at det var greit at programmering var kommet inn i matematikkfaget, fordi det var nyttig i arbeidslivet, mens han på 10. trinn la til at programmering kan være gøy og kan gi motivasjon for matematikkfaget. Han uttrykte også en utvikling fra 8. trinn, der han mente at han var litt «bakpå» og verken hadde kommet i gang med programmering i undervisningen, eller hadde lyst til å lære det selv, men at han hadde vært på kurs. På 10. trinn sa han derimot at «nå er jeg klar» til å undervise. Han fortalte at han hadde lært mye av en yngre kollega og av selv å gjennomføre de digitale veilederne i nettressursen Kikora som elevene jobbet med. Samtidig sa han at blant annet «def-kommandoer» var utfordrende, som er Pythons kommando for å lage funksjoner. Dette er et kompetansemål for programmering etter 6. trinn. Karen fortalte på 8. trinn at hun godt visste hva algoritmisk tenkning var, at hun kunne programmering fra studiet sitt og at hun var positiv til at programmering var kommet inn på 8. trinn. På 10. trinn betonet hun tydeligere at programmering var gøy og at hun hadde lært seg mer programmering ved å undervise det. Selv om hun ikke nevnte noen utfordringer med programmeringen på 8. trinn, sa hun på 10. trinn at det var et problem at elevene ikke kunne det de skulle fra tidligere skoleslag og at hun manglet tid.

5.9 Oppsummering

Oppsummert har vi i denne delen sett at lærerne loggførte at de inkluderte algoritmisk tenkning i alle matematikkfagene på alle trinn, men loggene samsvarte ikke med når vi observerte algoritmisk tenkning i undervisningen (se Del 4). Forklaringen fant vi i intervjuene, der de fleste lærerne definerte algoritmisk tenkning ulikt oss og læreplanen, og mange definerte det som å utføre algoritmer eller å forstå kjente algoritmer. Lærerne hadde få synspunkter om algoritmisk tenkning, men mange om programmering. De var positive til programmering, og trodde det kunne føre til motivasjon, gi nye muligheter i undervisningen og gjøre matematikk mer yrkesrelevant. Lærerne hadde programmert mindre med klassen enn de syntes de burde. Noen hadde ikke startet med programmering, enten fordi de hadde lagt det som eget tema på slutten av året eller fordi de prøvde å programmere så lite som mulig.

I og med at det kun var to av lærerne som fulgte de filmede klassene fra 8. trinn (2021–22) til 10. trinn (2023–24), hadde vi lite data om utviklingen i lærernes syn på algoritmisk tenkning og programmering over tid. Likevel er det optimistisk at Sander mente han hadde klart å tilegne seg mer programmeringskunnskap via undervisningen og at han hadde kommet frem til at programmering kunne være gøy. Lærerne syntes det var utfordrende at de ikke visste hva elevene skulle kunne, at de ikke hadde nok undervisningstid til programmering og at elevene ikke kunne det de skulle fra tidligere skoleslag. De fleste lærerne syntes heller ikke de kunne programmere selv. Mange hadde vært på programmeringskurs, som de syntes var utilstrekkelige, og nevnte de at de lærte mer av kollegaer eller sammen med elevene.

DEL 6

Diskusjon og implikasjoner

6 Diskusjon og implikasjoner

I denne avsluttende delen trekker vi fram de tre hovedfunnene i EDUCATE Rapport 5 og diskuterer dem i lys av implikasjoner for lærere, skoleledelse, skoleeier og myndigheter. *Formålet med denne rapporten har vært å svare på UDIRs oppdrag om å bidra til økt kunnskap om implementeringen av algoritmisk tenkning og programmering på 8. trinn, 10. trinn og Vg1, og gi indikasjoner på hva som kjennetegner slike klasseromspraksiser i de obligatoriske matematikkfagene, både innenfor og på tvers av trinn.*

EDUCATE-prosjektet benytter et komplekst forskningsdesign som integrerer flere metoder for å få fram ulike perspektiver på arbeid med algoritmisk tenkning og programmering i matematikkfagene. Kjernen i evalueringen er filming av naturlig forekommende undervisning i klasserom, der vi eksplisitt ber lærerne om å gjennomføre ordinær undervisning uten endringer i undervisningsopplegg eller noen form for tilpasning (se Brevik m.fl., 2023a). Det betyr at lærerne som deltok, ikke er bedt om å undervise spesifikt om eller legge til rette for algoritmisk tenkning eller programmering i matematikkundervisningen. I tillegg har vi spurt lærerne om deres perspektiver på algoritmisk tenkning generelt, i læreplanverket LK20 og i sin undervisning, gjennom logger de har ført umiddelbart før og etter undervisningen, samt lærerintervjuer som er gjennomført etter at filmingen er avsluttet. Til denne rapporten har vi brukt data fra undervisning i obligatoriske matematikkfag på 8. trinn, 10. trinn og Vg1, blant 17 lærere i til sammen 29 klasserom. Fagene på Vg1 er 1T, 1P og 1P-Y.

6.1 Tre hovedfunn om algoritmisk tenkning og programmering

Vi har identifisert tre hovedfunn som bygger på integrerte analyser av videofilmet undervisning, Lærerlogger og lærerintervjuer i de obligatoriske matematikkfagene på 8. trinn, 10. trinn og Vg1. De praksisene vi ser i fagene kjennetegnes av tre hovedfunn: For det første ser vi at algoritmisk tenkning forekommer i matematikkundervisningen, men at lærerne synes konseptet er uklart og definerer det på ulike måter (se Hovedfunn 1). For det andre forekom programmering sjelden i undervisningen, men lærerne hadde positive holdninger til programmering, selv om de manglet programmeringskompetanse (se Hovedfunn 2). For det tredje ser vi en uklar progresjon i algoritmisk tenkning og programmering mellom matematikkfag og på tvers av trinn (se Hovedfunn 3).

Hovedfunn 1: Algoritmisk tenkning forekommer i matematikkundervisningen, men lærerne synes konseptet er uklart og definerer det på ulike måter

Den videofilmede undervisningen i de obligatoriske matematikkfagene på 8. trinn, 10. trinn og Vg1 viser at algoritmisk tenkning forekom i halvparten av timene. Det vi omtaler som algoritmisk tenkning er delt inn i algoritmisk problemløsning og algoritmiske arbeidsmåter.

I matematikkundervisningen besto den *algoritmiske problemløsningen* oftest av figurtallundervisning og problemløsningsoppgaver, og i noen klasserom utviklet eller forklarte elevene en algoritme eller regneregel. Dette arbeidet var hovedsakelig preget av dekomponering og resonnering, og i svært liten grad av at elever faktisk lagde en algoritme som løste et problem. De *algoritmiske arbeidsmåtene* besto oftest av fikling, altså leken utprøving for å komme videre med en oppgave, og skaping, gjerne mens elevene samarbeidet. Derimot jobbet de lite med feilsøking, kanskje fordi feilsøking ofte gjøres individuelt og kan derfor være utfordrende å fange på lyd og film.

Lærerne planla å jobbe med algoritmisk tenkning i en tredel av timene. Samtidig var det lite samsvar mellom det lærerne loggførte etter undervisningen og det vi observerte i de videofilmede timene. Lærerne uttrykte en usikkerhet når det gjaldt forståelsen av begrepet algoritmisk tenkning. Fire av de 17 lærerne vi intervjuet forbant algoritmisk tenkning med en problemløsningsmetode, i likhet med det som står i LK20. Samtidig presenterte 12 lærere andre definisjoner og én lærer sa han ikke visste hva begrepet innebar. Ingen lærere nevnte algoritmiske arbeidsmåter som en del av algoritmisk tenkning. Her bør skoleeier og skoleledelsen ta ansvar for å utarbeide felles forståelse for algoritmisk tenkning i tråd med læreplanen. Se forslag i EDUCATEs algoritmiske agenda.

Et hovedspørsmål i denne rapporten er om innføringen av algoritmisk tenkning i LK20 har ført til ønskede praksisendringer. Ut fra våre observasjoner og intervjuer med lærere ser det ut som innføringen har ført til begrensede endringer i undervisningen. Algoritmisk tenkning forekom i undervisningen ved at elevene jobber med problemløsningsoppgaver som krever de algoritmiske problemløsningsmetodene *å dekomponere* og *å se etter mønstre*. De benytter også de algoritmiske arbeidsmåtene *å fikle* og i noen grad *å skape*. Likevel visste få av lærerne hva algoritmisk tenkning var, og mange forbandt det kun med programmering. Dette samsvarer med andre studier som har vist at verken lærere eller elever forbinder de algoritmiske arbeidsmåtene med algoritmisk tenkning (Thune, 2023). Våre funn tyder derfor på at innføringen av begrepet al*goritmisk tenkning* i LK20 har ført til begrensede praksisendringer, og at de praksisendringene vi ser skyldes at programmering nå nevnes i kompetansemålene snarere enn at algoritmisk tenkning nevnes i kjerneelementet om problemløsning.

Det er grunn til å vurdere om den manglende praksisendringen i forbindelse med algoritmisk tenkning handler om at begrepet tilfører lite nytt til allerede etablerte praksiser. De fleste begrepene som brukes i LK20 for å beskrive algoritmisk tenkning er dekket i andre deler av læreplanen. Mange av begrepene i Figur 2.2 kjenner vi igjen fra andre steder i LK20 (se Tabell 2.1). Tabell 2.1 viser hvordan begreper om algoritmisk tenkning er koblet til overordnet del (Kunnskapsdepartementet, 2017) og til læreplanen i matematikk (Kunnskapsdepartementet, 2020).

Tabell 2.1 viser hvordan algoritmisk problemløsning er fremhevet i kjerneelementene i LK20 og hvordan de algoritmiske arbeidsmåtene er fremhevet i kjerneelementene eller i overordnet del. Dette kan tyde på at de hyppige observasjonene våre av algoritmisk tenkning i undervisningen kan være forårsaket av at lærerne vektlegger overordnet del eller kjerneelementene i matematikkfagene snarere enn algoritmisk tenkning. Dette samsvarer med at vi observerte mer arbeid med problemløsning og arbeidsmåter som er nevnt andre steder i læreplanen (*se etter mønstre*, *fikle*, *samarbeide*), enn det som er mer spesifikt knyttet til algoritmisk tenkning (*logikk*, *algoritmer* og *feilsøking*).

Dersom LK20s definisjoner av algoritmisk tenkning hadde vært mer informatikknære, slik som i rammeverket til Weintrop m.fl. (2016), se Tabell 2.2, kunne vi muligens ha sett større endringer i undervisningspraksisen i matematikkfagene. Da ville algoritmisk tenkning inkludert å *omforme problemer for algoritmiske løsninger* og *utvikle modulære beregningsbaserte løsninger og abstraksjoner*, noe som vi ikke finner igjen andre steder i læreplanen. Det hadde muligens ført til at algoritmisk tenkning fremsto som noe nytt og gitt praksisfeltet konkrete mål å jobbe mot. En positiv vinkling av at læreplanens definisjon av algoritmisk tenkning i stor grad overlapper med andre deler av læreplanen, er at den kan støtte opp om resten av læreplanen. For eksempel var timene der vi så *skaping* svært avanserte ifølge EDUCATEs protokoll for observasjon av algoritmisk tenkning (se Boks 3), og de er i stor grad de samme timene som har blitt fremhevet i matematikkfaget i EDUCATEs Rapport 3 om utforsking

(Brevik m.fl., 2024) og Rapport 4 om digital kompetanse (Gudmundsdottir m.fl., 2024). At LK20s definisjon av algoritmisk tenkning er fra den samme konstruktivistiske tradisjonen som skapte *Scratch* og *Makerspaces,* kan føre til at matematikkfaget blir mer skapende og utforskende og bygger opp under vyene fra overordnet del om skaperglede og utforskertrang.

Hovedfunn 2: Programmering forekom sjelden i undervisningen og lærerne sa de manglet programmeringskompetanse, men de hadde likevel positive holdninger til programmering

Lærerne underviste i programmering i 3 % av tiden, eller i 7 % av tiden hvis vi regner med bruk av vilkår i regneark og graftegner. Programmeringsundervisningen var hovedsakelig at elevene lærte seg programmering ved å følge digitale veiledere (*tutorials*) mens læreren hjalp elevene. Elevene benyttet noen ganger programmeringsliknende syntaks (spesielt vilkår) når de arbeidet i regneark og graftegner, og i noen få tilfeller løste de programmeringsoppgaver eller benyttet programmering som verktøy. Av de fire konseptene elevene skal kunne om programmering etter 6. trinn (*variable*, *vilkår*, *løkker* og *funksjoner*) identifiserte vi kun arbeid med variable og vilkår i undervisningen.

I intervjuene fortalte lærerne at de programmerte mindre enn de hadde planlagt, og de fleste sa at de underviste programmering som et eget tema, løsrevet fra matematikken. Dette underbygges av våre observasjoner av timene vi filmet. Det er viktig å påpeke at andelen programmering i de filmede matematikktimene trolig underestimerer andelen programmering i disse klassene. Flere lærere nevnte i intervjuene at de ofte underviste om programmering som et eget tema rett før sommerferien. EDUCATE-teamet filmet undervisning andre tider på året, for ikke å forstyrre skolene i eksamensforberedelser. Vi kan derfor ha gått glipp av programmeringsundervisningen på tampen av året som lærerne fortalte om i intervjuene.

Lærerne var positive til at programmering hadde kommet inn i skolen, både fordi de anså det som en viktig kompetanse i arbeidslivet og et tema som kunne bidra til elevenes motivasjon i faget. Derimot var de kritiske til egen programmeringskompetanse. De fleste lærerne mente de ikke kunne å programmere godt nok, mens noen sa de hadde akkurat nok kompetanse til å *prøve* programmering i undervisningen. Med tanke på lærernes eksplisitte uttrykk for manglende programmeringskompetanse, slik det fremkommer både i denne og andre studier, fremstår det som viktig å tilby kompetanseheving i programmering for lærere. Se forslag i EDUCATEs digitale agenda til slutt i dette sammendraget og utdypet i Del 6.

Analysene våre tyder på at innføringen av programmering i matematikkfaget ikke er kommet langt. Lærerne underviser lite om programmering, har selv begrenset programmeringskompetanse og vet ikke hva elevene skal kunne. Når programmering blir undervist, er det ofte som et separat tema frikoblet fra resten av faget. Noen av utfordringene vil løse seg med tiden. Når elevene har hatt mer programmering på lavere årstrinn trenger ikke lærerne å starte med blanke ark på Vg1. I tillegg, når det foreligger flere eksamener, vil lærerne ha bedre innsyn i hva som kreves av elevene på de ulike trinnene. Spørsmålet er hvilke av utfordringene som vil løses av seg selv når LK20 er mer etablert, og hvilke som vil trenge betydelig innsats av lærere, skoleledere, skoleeiere og myndigheter for å løse.

Én av lærerne som ble intervjuet sammenliknet statusen til programmering med statusen GeoGebra hadde for ti år siden. Da matematikkfagets læreplan ble revidert i 2013 og eksamensordningen ble revidert i 2015, kom graftegnere, dynamisk geometriprogramvare og CAS (computer algebra systems) inn i matematikkfaget. Verktøyene ble undervist på slutten av skoleåret som et eget tema, lærerne

måtte lære seg dem selv eller delta på kurs, og de var usikre på hva som var forventet av elevene og hvilke funksjoner elevene måtte kunne til eksamen. Nå, ti år senere, ser vi i de filmede timene at GeoGebra er godt integrert i undervisningen. Lærere underviser i GeoGebra sammen med matematikktemaene der de finner det naturlig, og vet hva elevene skal kunne. Spørsmålet er om vi kan forvente at det samme vil skje med algoritmisk tenkning og programmering.

Det er flere argumenter imot at programmeringen skal løse seg på samme måte som for GeoGebra. Programvaren GeoGebra er laget for å benyttes i matematikkundervisning og er forholdsvis brukervennlig. Programvaren er visuell med et velkjent pek-og-klikk-grensesnitt og det er innlysende hvordan den kan benyttes i geometri-, funksjons- og algebraundervisning. Med pek-og-klikk kan man prøve seg frem og finne ut av funksjonaliteten uten hjelp. Scratch og annen blokkprogrammering som benyttes mye på barnetrinnet, har også et pek-og-klikk-grensesnitt og kan derfor læres ved å prøve seg frem. Python og annen tekstprogrammering har ikke et pek-og-klikk-grensesnitt, og lærere og elever må overvinne barrieren det er å skulle instruere en datamaskin ved å skrive tekst. Det er ingen menyer som viser alle mulighetene som finnes i programvaren, og det er ikke mulig å lære seg programmering ved enkel prøving og feiling. Dersom tekstbasert programmering ikke kan læres uten vesentlig tidsbruk, må man spørre seg hvor mye tid lærere trenger til å lære dette.

På universitetene tilbys innføringsemner i tekstbasert programmering for lærere, for eksempel «IT1001 informasjonsteknologi, grunnkurs» ved NTNU[11]. Der tilbys «en introduksjon til prosedyreorientert programmering, med Python som språk, samt erfaring med å gjennomføre et lite programmeringsprosjekt med bruk for undervisning i realfag i ungdomsskole og videregående». Emnet gir 7,5 studiepoeng, som tilsvarer 225 timers arbeid. Dersom lærere tilbys en halv dag til jobbing med emnet hver uke (for eksempel tre timer etter lunsj, slik noen lærere rapporterer), vil det ta 78 uker før lærerne får dekket emnet, altså nesten to arbeidsår. Det er mulig at kurset tilbyr mer programmering enn det matematikklærere trenger for å undervise på ungdomstrinnet og i videregående skole. Det fremstår likevel som sannsynlig at det å tilby lærere kurs i vanlig arbeidstid er utilstrekkelig. Lærerne vi intervjuet støttet dette synspunktet.

Undervisning i programmering kan berike matematikkfaget og de matematiske temaene kan berike programmering. Vi observerte i liten grad at slike gjensidige styrker ble vektlagt i den filmede undervisningen. Vi ønsker derfor å trekke frem figurtall som et tema der programmering kunne vært benyttet i økt grad. Å programmere løsninger på figurtalloppgaver kan utvikle matematikkundervisningen. For det første gir programmering elevene en grunn til å finne et generelt uttrykk for figurtallet (se utdraget i Del 2 fra Kilhamn m.fl., 2022). For det andre gir programmering en mulighet til å løse figurtallsoppgaver med rekursive formler. Elever som ikke fant et generelt uttrykk, men som så en rekursiv sammenheng («det øker med 1 mer hver gang»), har ingen måte å løse mange av oppgavene på. Med programmering gis de en mulighet. Elevene kan deretter motiveres til å finne generelle uttrykk ved å se at det fører til færre linjer med kode.

Å programmere løsninger på figurtallsoppgaver kan også løfte programmeringen. Hvis elever lærer seg å programmere løsninger på figurtallsoppgaver, vil det gi elevene en kontekst der programmering er

[11] https://www.ntnu.no/studier/emner/IT1001

nyttig. Figurtall fremstår også som en kontekst som egner seg til å lære programmering, siden løsningene blir relativt korte og benytter seg av få programmeringskonsepter. Ofte vil det holde å initiere en variabel og benytte den i en løkke.

For å få progresjon, kan lærerne enkelt lage mer omfattende oppgaver der elevene kan øve på vilkår («når blir figurtallet høyere enn …») eller løpende summer («når har figurene totalt benyttet 1000 brikker»?). Elevene vil dermed også få mengdetrening på relativt like oppgaver over tid.

For å finne koblinger mellom programmering og matematikk kan det være en svakhet at vi kun har filmet 8. og 10. trinn og Vg1. Årsaken er at kompetansemålene om programmering i LK20 i disse fagene ikke er koblet spesifikt til matematiske temaer, men sier at elevene skal lære å «utforske hvordan algoritmer kan skapes, testes og forbedres med programmering», «utforske matematiske egenskaper og sammenhenger ved å bruke programmering» og «formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsningsstrategier, digitale verktøy og programmering». På 6., 7. og 9. trinn og i Matematikk S2 og R2 er kompetansemålene spesifikt knyttet til matematiske temaer, henholdsvis geometri, statistikk, sannsynlighet, rekursive sammenhenger og bestemte integraler. Det er derfor grunn til å tro at programmering og matematikk hadde vært tettere knyttet sammen på disse trinnene.

Det er ikke overraskende at lærerne svarte at de manglet programmeringskompetanse, i og med at det har vært rapportert tidligere (for eksempel Kravik m.fl., 2022). Våre data fra våren 2024 bekrefter dette. EDUCATE mener det er viktig at lærerne selv opplever at de har nok programmeringskompetanse til å undervise kompetansemålene i matematikk i LK20. Flere lærere nevnte kollegaer som hadde mer kunnskap enn dem selv og som hjalp dem med undervisningsopplegg knyttet til programmering. Samtidig var lærerne motiverte til å utvikle programmeringskompetansen sin. De opplevde at kursene de hadde deltatt på hadde vært nyttige, selv om de mente de var utilstrekkelige. Flere lærere sa også at de var lettet over at eksamen for elevene på 10. trinn og vg1 fortsatt var på et overkommelig nivå, slik at mangelen på programmeringsundervisning ikke ble urettferdig for elevene.

Det springende punktet for programmeringsundervisningen i matematikk ser derfor ut til å være hvordan lærerne skal heve egen programmeringskompetanse. Lærerne som deltok i EDUCATE-prosjektet var mer kritiske til egen kompetanse enn lærerne i studien til Turgut m.fl. (2024), som samlet svar fra 365 matematikklærere (se Figur 2.5). Samtidig er det viktig å presisere at matematikklærerne som deltok i de to prosjektene ikke nødvendigvis var representative for matematikklærere i hele Norge, slik at forskjellen mellom det lærerne rapporterte kan knyttes til systematiske forskjeller i utvalgene. Med tanke på lærernes eksplisitte uttrykk for manglende kompetanse, fremstår programmering som et område det er spesielt viktig å tilby kompetanseheving i. Se forslag i EDUCATEs algoritmiske agenda.

Hovedfunn 3: Uklar progresjon i algoritmisk tenkning og programmering mellom fag og trinn

Det tredje hovedfunnet i denne rapporten peker på en uklar progresjon i algoritmisk tenkning og programmering mellom trinn og mellom ulike matematikkfag. Verken den videofilmede undervisningen eller lærerintervjuene viste klare forskjeller mellom matematikkfag og trinn. For eksempel ble det undervist om figurtall på alle trinn og fag, og til dels samme oppgaver på 8. trinn som i matematikk 1T. Vi så også de samme programmeringskonseptene bli undervist til samme klasse på både 8. og 10. trinn. Også i intervjuene med de to lærerne som fulgte klassene sine fra 8. til 10. trinn, ga lærerne relativt like svar om algoritmisk tenkning og programmering på tvers av trinnene, selv om de rapporterte at de hadde undervist noe mer programmering skoleåret 2023–24 enn de gjorde i 2021–22. Den manglende progresjonen kan være forårsaket av at våre data er fra de første årene etter innføringen av LK20, og at elevene derfor ikke kan det LK20 forventer fra tidligere trinn. Da er det naturlig at vi, for eksempel, ser den samme programmeringsundervisningen på 8. trinn som på Vg1. Likevel, i våre data er det en mangel på progresjon mellom trinnene.

Denne mangelen på progresjon har vi sett på tvers av våre datakilder. Verken den videofilmede undervisningen eller lærerintervjuene viste klare forskjeller mellom matematikkfag og trinn, selv om lærerne rapporterte at de hadde undervist noe mer programmering skoleåret 2023–24 enn i skoleåret 2021–22. For eksempel observerte undervisning om figurtall i matematikktimer på alle trinn og fag, til og med i samme klasse på både 8. og 10. trinn, der elevene jobbet med tilnærmet like oppgaver og metoder og som begge årene lærte om variabeltilordning og hvordan man kunne skrive til skjerm. For å adressere denne utfordringen bør det legges vekt på tydelig progresjon mellom trinn og skoleslag, for å bidra til et mer helhetlig og integrert læringsløp knyttet til algoritmisk tenkning og programmering. Lærere kan også dra nytte av samarbeid på tvers av trinn for å sikre konkret progresjon.

6.2 EDUCATEs agenda for algoritmisk tenkning og programmering

Lærere

- Lærerne bør forsøke å **jobbe med programmering når elevene løser figurtallsoppgaver**, i tillegg til å jobbe med penn og papir. Vi baserer dette på at figurtallsoppgaver gikk igjen på samme måte på alle trinnene vi filmet. Å benytte figurtallsoppgaver som en kontekst for programmering vil både kunne gi jevnlig øving på programmering og en progresjon i figurtallundervisningen. Figurtallsoppgaver er godt egnet for programmeringsløsninger fordi de støtter opp om målet for figurtallsoppgaver (å uttrykke vekst algebraisk), gir elever som kun ser rekursive formuleringer av veksten en mulighet til å løse oppgavene, gir en kontekst der elevene ser at programmering kan være nyttig, og kan differensieres til å også benytte vanskeligere programmeringskonsepter. Programmerings-konseptene man øver ved figurtall kan overføres relativt direkte til andre områder som også jobbes med i skolematematikken, som modellering av økonomisk vekst (renter, inflasjon) og populasjoner, men de ble ikke jobbet med i timene vi filmet.

Skoleledelse

- EDUCATE oppmuntrer skoleledelsen til å ta ansvar for å **kartlegge og utarbeide felles forståelse** for algoritmisk tenkning blant matematikklærerne ved skolen. Vi baserer dette på lærernes usikkerhet om begrepet algoritmisk tenkning; kun et fåtall definerte begrepet i tråd med læreplanen og det var liten sammenheng mellom når lærerne loggførte at timen inneholdt algoritmisk tenkning og når vi observerte det. Det er viktig at matematikklærere ved samme skole har en felles forståelse i tråd med læreplanen, av hva algoritmisk tenkning innebærer i undervisningen. Dette sikrer at elevene får en konsistent opplæring uavhengig av hvilke lærere de har, og at de opplever progresjon i faget. Når lærerne jobber mot de samme målene og bruker felles terminologi og sammenfallende metoder, kan de bedre støtte hverandre og fremme en kultur for samarbeid og deling av gode praksiser.
- EDUCATE ser et tydelig behov for at skoleledelsen **tilrettelegger for kompetanseheving** blant matematikklærere, slik at de får mulighet til å utvikle sin kompetanse, spesielt når det gjelder programmering. Vi baserer dette på at det ikke er nok at myndigheter og skoleeiere tilbyr kompetanseheving, skoleledelsen må også tilrettelegge for dette, for eksempel ved å unngå undervisning på bestemte dager for lærere som skal på kurs eller å lage klare rammer for vikarer. Å gi lærerne mulighet til å utvikle kompetansen sin gjennom kurstilbud eller tid til kollegasamarbeid vil bidra til undervisning i tråd med LK20.

Myndigheter og skoleeiere

- EDUCATE ser behov for at myndigheter **kartlegger programmeringsundervisning ved lærerutdanningene** og går i dialog med lærerutdanningene om kompetanseheving i programmering og programmeringsdidaktikk blant lærerstudenter.
- EDUCATE ser sterkt behov for at myndigheter og skoleeiere **utarbeider undersøkelser og kartlegger læreres programmeringskompetanse**. Vi baserer dette på våre intervjuer med lærere og tidligere forskning som viser at matematikklærere sier de ikke kan programmere godt nok. Vi mener funnet om at matematikklærere mangler programmeringskompetanse er så alvorlig og så godt underbygget at myndigheter bør monitorere situasjonen inntil de fleste matematikklærere kan lese og lage kode med variabler, vilkår, løkker og funksjoner. En slik kartlegging vil bidra med innsikt i hva som mangler av programmeringskompetanse og innenfor hvilke områder. EDUCATEs forslag er å gjennomføre en spørreundersøkelse blant et representativt utvalg lærere annethvert år. Her vil man kunne samarbeide med Universitetet i Bergen og Nasjonal forkunnskapsprøve i programmering om å låne oppgaver som måler programmeringskompetanse (Bolland m.fl., 2024). Undersøkelsene kan med fordel også gi matematikklærere mulighet til å rapportere om de opplever å være rustet til å undervise om programmering på ungdomstrinnet og i videregående skole. Vi vet mindre om kompetansen på barnetrinnet, men ønsker å poengtere at kompetansemålene med programmering på 6. og 7. trinn er avanserte.

- EDUCATE oppfordrer myndighetene til å **øke satsingen på kompetanseheving** for matematikklærere, i tillegg til å gjennomgå eksisterende kompetansehevingstiltak innenfor programmering og programmeringsdidaktikk og vurdere om revidering av disse er nødvendig. Det krever mye arbeid å lære seg å programmere, mer enn man kan forvente av matematikklærere med den tiden de har til rådighet i en fulltids lærerjobb. Matematikklærere bør tilbys kompetanseheving tilsvarende 7,5 studiepoeng, rundt 200 timer, i et didaktisk rettet programmeringskurs for de matematikkfagene læreren underviser i. De 200 timene bør inkludere seminarer, workshoper, lekser, innleveringsoppgaver, utprøving av ferdige programmeringsopplegg, design og utprøving av egne programmeringsopplegg samt egenarbeid. Alt arbeidet bør kunne gjøres i vanlig arbeidstid ved reduksjon i mengden undervisning. Alternativt kan det tilbys kortere sommerkurs til lærere i algoritmisk tenkning og programmering, som lærerne bør få lønn eller stipend for å gjennomføre.
- EDUCATE anbefaler myndighetene å **klargjøre hva som forventes av programmeringsundervisningen**. Vi baserer dette på at lærerne var usikre på om det var forventet at elevene skulle programmere med tekst og hvor mye tekstprogrammering som var nødvendig å kunne. I tillegg fant EDUCATE at det var vanskelig å finne dokumenter som spesifiserte noe om programmering. Læreplanen spesifiserer ikke om elevene skal programmere med tekst. Basert på læreplanen kunne elevene benyttet blokkprogrammering, analog programmering, eller programmering i Excel, gjennom grunnskolen og videregående, inkludert Matematikk R1 og R2. Eksamensveiledningen på 10. trinn spesifiserer at elevene kan møte oppgaver der kode er vist med tekst og at de *kan* løse oppgaver med blokk- eller tekstprogrammering. I eksamensveiledningen til 1T står det at hvis programkode representeres i en oppgave, vil den være skrevet i Python. Likevel spesifiserer verken læreplanen eller eksamensveiledningene hvilke kommandoer elevene bør kunne. Læreplanen og eksamensveiledningene bør samkjøres. Dersom det er forventet at elevene skal forstå tekstkode til eksamen på ungdomstrinnet og Python-kode til eksamen i 1T, bør det spesifiseres i kompetansemålene at elevene skal lære å forstå tekstkode og Python. Eksamensveiledningen kan med fordel ha en liste med kommandoer som vil bli brukt til å representere programkode til eksamen.
- EDUCATE anbefaler en **tydeliggjøring i læreplanverket og veiledninger**. Dette innebærer for eksempel at Utdanningsdirektoratet klargjør: (a) om elever på barnetrinnet skal programmere med tekst, (b) om elever på ungdomstrinnet skal programmere med tekst og om de skal lære Python, (c) hvilke programmeringskonsepter som kan representeres med tekst i eksamen til ungdomsskolen, (d) hvilke Python-kommandoer elevene er forventet å forstå til eksamen i 1T.

- EDUCATE oppfordrer myndighetene til å **følge nøye med på elevenes kompetanse i programmering**. Vi baserer dette på at flere lærere fortalte at de ikke hadde lagt inn noe programmering det året vi intervjuet dem. Ved å følge med på elevenes kompetanse kan man få innsikt i hva de får muligheten til å lære. Dette kan for eksempel gjøres ved å fortsette gjennomføringen av ICILS, den internasjonale *Information and Computer Literacy Study*. Da er det viktig å sørge for å inkludere deltakelse i modulen om algoritmisk tenkning. ICILS avholdes hvert femte år og gir i utgangspunktet kun kunnskap om norske 9. trinnelevers ferdigheter i algoritmisk tenkning, ikke i programmering. Likevel er oppgavene om algoritmisk tenkning er meget programmeringsrelevante og vil derfor kunne gi pekepinn om norske elevers programmeringskompetanse. At Norge ikke deltok i ICILS 2018, der algoritmisk tenkning var målt for første gang, fremstår som uheldig med tanke på sammenlikning mellom LK06 og LK20. Det betyr at vi har begrenset kunnskap om hvordan norske elevers kompetanse i algoritmisk tenkning var før LK20. En annen kilde til informasjon om elevers programmeringskompetanse vil være enkeltoppgaver fra skriftlig eksamen. Ved å følge med på hvor mange poeng elever får på oppgaver som krever programmering, vil man kunne få en viss informasjon om hva de kan. I og med at eksamen ikke er laget for å måle utvikling, må slike resultater tolkes i kontekst.

6.3 Konklusjon

Avslutningsvis ønsker vi å understreke at de praksisene vi ser i matematikkundervisningen på 8. trinn, 10. trinn og Vg1, viser at algoritmisk problemløsning, algoritmiske arbeidsmåter og programmering brukes i alle de obligatoriske matematikkfagene, men i variert omfang.

Lærernes arbeid med å innføre algoritmisk tenkning og programmering etter LK20 er likevel i startfasen. Lærerne sliter med å forklare hva algoritmisk tenkning er, og de mener selv at de mangler programmeringskompetanse. Algoritmisk tenkning er til stede i matematikkundervisningen, men mest i de generelle delene av begrepet, som overlapper med generell utforsking og problemløsning. Lærerne er likevel positive til innføringen av algoritmisk tenkning og programmering i læreplanen, de blir tilbudt kurs for å lære seg å programmere, og de får hjelp av ressurspersoner ved egen skole. Et usikkerhetsmoment er om lærerne vil lærer seg å programmere godt nok med de tiltakene som allerede er satt i gang. Programmering er krevende å lære seg godt som del av en fulltids lærerjobb. Ambisjonene i LK20 er at programmeringen skal benyttes til utforsking og skaperglede i matematikkfagene på alle trinn. Dette krever programmeringsferdigheter på et annet nivå enn lærerne kan forventes å tilegne seg ved egen utprøving i ledige stunder i arbeidstiden.

7 Referanser

Babbage, C. (1864). *Passages from the Life of a Philosopher*. Longman, Green, Longman, Roberts, & Grenn. https://www.gutenberg.org/files/57532/57532-h/57532-h.htm

Ball, D. L., & Hill, H. C. (2009). Measuring teacher quality in practice. I D. H. Gitomer (Red.), *Measurement issues and assessment for teaching quality* (s. 80–98). SAGE.

Bill & Melinda Gates Foundation. (2012). *Gathering feedback for teaching: Combining high quality observations with student surveys and achievement gains*. The Bill & Melinda Gates Foundation.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is Involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Beecher, K. (2017). *Computational Thinking: A beginner's guide to problem-solving and programming* (Illustrated edition). BCS, The Chartered Institute for IT.

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M. A., Jasuté, E., Malagoli, C., Masiulionytė-Dagienė, V., & Stupurienė, G. (2022). *Reviewing computational thinking in compulsory education: State of play and practices from computing education*. Publications Office of the European Union. https://publications.jrc.ec.europa.eu/repository/handle/JRC128347

Bolland, S. S., Haraldsrud, A., Jensen, S. M., Strömbäck, F., Styve, A., Tøssebro, E., Valseth, E., & Strømme, T. J. F. (2024). *Nordic Prior Knowledge Test in Programming 2024*.

Boom, K. D., Bower, M., Siemon, J., & Arguel, A. (2022). Relationships between computational thinking and the quality of computer programs. *Education and Information Technologies*, *27*(6), 8289–8310. https://doi.org/10.1007/s10639-022-10921-z

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Paper Presented at the Annual Meeting of the American Educational Research Association*.

Brevik, L. M., Gudmundsdottir, G. B., Doetjes, G., & Barreng, R. L. S. (2023). *Å observere fagfornyelsen i klasserommet. Observasjonsprotokoller for livsmestring, utorsking og digital kompetanse* [Rapport 1 fra forsknings- og evalueringsprosjektet EDUCATE ved Institutt for lærerutdanning og skoleforskning]. Universitetet i Oslo. https://www.uv.uio.no/ils/forskning/prosjekter/educate/rapporter/educate-rapport-1-2023.pdf

Brevik, L. M., Gudmundsdottir, G. B., Aashamar, P. N., Barreng, R. L. S., Dodou, K., Doetjes, G., Hartvigsen, K. M., Hatlevik, O. E., Mathé, N. E. H., Roe, A., Siljan, H., Stovner, R. B., & Suhr, M. L. (2024). *Å jobbe utforskende på Vg1 og vg2. Den enkelte lærers undervisning har mer å si enn fagenes egenart*. [Rapport 3 fra forsknings- og evalueringsprosjektet EDUCATE ved Institutt for lærerutdanning og skoleforskning]. Universitetet i Oslo.

Brevik, L. M., & Mathé, N. E. H. (2021). Mixed methods som forskningsdesign. I E. Andersson-Bakken & C. P. Dalland (Red.), *Metoder i klasseromsforskning: Forskningsdesign, datainnsamling og analyse* (s. 47–70). Universitetsforlaget.

Bråting, K., & Kilhamn, C. (2021). Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning*, *23*(2), 170–185. https://doi.org/10.1080/10986065.2020.1779012

Buchholz, K., & Løvseth, L. (2024). *Programmering i matematikk. En kvalitativ studie om læreres kunnskaper, holdninger og utfordringer* [Masteroppgave, Oslo Metropolitan University]. https://oda.oslomet.no/oda-xmlui/handle/11250/3162504

Computing at School. (2024). *Friends of CAS*. Computing At School. https://www.computingatschool.org.uk/about-us/friends-of-cas

Csapó, G., Csernoch, M., & Abari, K. (2020). Sprego: Case study on the effectiveness of teaching spreadsheet management with schema construction. *Education and Information Technologies*, *25*(3), 1585–1605. https://doi.org/10.1007/s10639-019-10024-2

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational Thinking: A guide for teachers*. Computing At School. https://www.computingatschool.org.uk/media/kscbloob/computationalthinking.pdf

Den nasjonale forskningsetiske komité for samfunnsvitenskap og humaniora [NESH]. (2021). *Forskningsetiske retningslinjer for samfunnsvitenskap og humaniora*. https://www.forskningsetikk.no/globalassets/dokumenter/4-publikasjoner-som-pdf/forskningsetiske-retningslinjer-for-samfunnsvitenskap-og-humaniora.pdf

Gudmundsdottir, G. B., Brevik, L. M., Aashamar, P. N., Barreng, R. L. S., Dodou, K., Doetjes, G., Hartvigsen, K. M., Hatlevik, O. E., Isaksen, A. R., Magnusson, C. G., Mathé, N. E. H., Roe, A., & Skarpaas, K. G. (2024). *Å gi rom for variasjon og valgfrihet, mens vi venter på digital dømmekraft* [Rapport 4 fra forsknings- og evalueringsprosjektet EDUCATE ved Institutt for lærerutdanning og skoleforskning]. Universitetet i Oslo.

Haseski, H., Ilic, U., & Tugtekin, U. (2018). Defining a new 21st century skill – computational thinking: Concepts and trends. *International Education Studies*, *11*(4), Artikkel 4. https://doi.org/10.5539/ies.v11n4p29

Jones, I., & Pratt, D. (2006). Connecting the equals sign. *International Journal of Computers for Mathematical Learning*, *11*(3), 301–325. https://doi.org/10.1007/s10758-006-9107-6

Løken, T., Mushtaq, A., Eichler, K. P., & Bímová, D. (2023). Teachers' perspectives on programming through emerging technologies in mathematics education. I P. Drijvers, C. Csapodi, H. Palmér, K. Gosztonyi, & E. Kónya (Red.), *Thirteenth Congress of the European Society for Research in Mathematics Education (CERME13)* (Bd. TWG15, Nummer 23). Alfréd Rényi Institute of Mathematics. https://hal.science/hal-04412067

Kallia, M., van Borkulo, S. P., Drijvers, P., Barendsen, E., & Tolboom, J. (2021). Characterising computational thinking in mathematics education: A literature-informed Delphi study. *Research in Mathematics Education*, *23*(2), 159–187. https://doi.org/10.1080/14794802.2020.1852104

Kilhamn, C. (2022). Tinkering in algebra—the case of John. *Twelfth Congress of the European Society for Research in Mathematics Education (CERME12)*, *TWG03*(08). https://hal.science/hal-03745133

Kilhamn, C., Bråting, K., Helenius, O., & Mason, J. (2022). Variables in early algebra: Exploring didactic potentials in programming activities. *ZDM – Mathematics Education*, *54*(6), 1273–1288. https://doi.org/10.1007/s11858-022-01384-0

Klette, K. (2009). Challenges in strategies for complexity reduction in video studies. Experiences from the PISA+ study. I T. Janik, & T. Seidel (Red.), The power of video studies in investigating teaching and learning in the classroom (s. 61–83). Waxmann Publishing.

Klette, K. (2023). Towards programmatic research when studying classroom teaching and learning. I F. Ligozat, K. Klette, & J. Almqvist (Red.), *Didactics in a changing world: European perspectives on teaching, learning and the curriculum* (s. 137–158). Springer. https://doi.org/10.1007/978-3-031-20810-2_9

Klette, K., Blikstad-Balas, M., & Roe, A. (2017). Linking instruction and student achievement. A research design for a new generation of classroom studies. *Acta Didactica Norge*, *11*(3). https://doi.org/10.5617/adno.4729

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323–343. https://doi.org/10.2307/2318994

Kravik, R., Berg, T. K., & Siddiq, F. (2022). Teachers' understanding of programming and computational thinking in primary education – A critical need for professional development. *Acta Didactica Norden*, *16*(4), Artikkel 4. https://doi.org/10.5617/adno.9194

Kunnskapsdepartementet. (2020a). *Læreplan i matematikk 1.–10. Trinn (MAT01-05)*. https://www.udir.no/lk20/mat01-05
Kunnskapsdepartementet. (2020b). *Læreplan i matematikk fellesfag VG1 praktisk (MAT08-01)*. https://www.udir.no/lk20/mat01-05
Kunnskapsdepartementet. (2020c). *Læreplan i matematikk fellesfag Vg1 teoretisk (MAT09-01)*. https://www.udir.no/lk20/mat01-05
Lahn, L. C., & Klette, K. (2023). Reactivity beyond contamination. An integrative literature review of video studies in educational research. *International Journal of Research & Method in Education*, *46*(1), 1–18. https://doi.org/10.1080/1743727X.2022.2094356
Leinhardt, G. (1987). Development of an expert explanation: An analysis of a sequence of subtraction lessons. *Cognition and Instruction*, *4*(4), 225–282. https://doi.org/10.1207/s1532690xci0404_2
Lv, L., Zhong, B., & Liu, X. (2023). A literature review on the empirical studies of the integration of mathematics and computational thinking. *Education and Information Technologies*, *28*(7), 8171–8193. https://doi.org/10.1007/s10639-022-11518-2
Maxwell, J. A. (2021). Why qualitative methods are necessary for generalization. *Qualitative Psychology*, *8*(1), 111–118. https://doi.org/10.1037/qup0000173
Munthe, M. (2022). Programmering i matematikklasserommet – Utfordringer elevene møter. *Acta Didactica Norden*, *16*(4), Artikkel 4. https://doi.org/10.5617/adno.9173
Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
Petersen, V. B. (2002). Figurtall–en kilde til kreativitet. *Tangenten – tidsskrift for matematikkundervisning*, *1*, 3–7.
Praetorius, A. K., Pauli, C., Reusser, K., Rakoczy, K., & Klieme, E. (2014). One lesson is all you need? Stability of instructional quality across lessons. *Learning and Instruction*, *31*, 2–12. https://doi.org/10.1016/j.learninstruc.2013.12.002
Researchware. (2023). *HyperRESEARCH 4.5.3* [Programvare]. http://www.researchware.com/
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779
Resnick, M., & Robinson, K. (2013). *Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play*. Mit Pr.
Resnick, M., & Rosenbaum, E. (2013). Designing for tinkerability. I M. Honey (Red.), *Design, make, play: Growing the next generation of STEM innovators*. Routledge. https://doi.org/10.4324/9780203108352
Rohatgi, A., Hatlevik, O. E., Gudmundsdottir, G. B., Erstad, O. A., & Björnsson, J. K. (2024). *ICILS 2023: Digital kompetanse og algoritmisk tenkning hos norske niendeklassinger*. Cappelen Damm Akademisk. https://doi.org/10.23865/noasp.219
Sandal, B. (2024). *Matematikklæreres oppfatninger og utfordringer ved integrering av programmering i matematikkundervisningen i ungdomsskolen* [Masteroppgave, Universitetet i Innlandet]. https://brage.inn.no/inn-xmlui/handle/11250/3138984
Sanford, J. (2018). Introducing Computational Thinking Through Spreadsheets. I M. S. Khine (Red.), *Computational thinking in the STEM disciplines: Foundations and research highlights* (s. 99–124). Springer. https://doi.org/10.1007/978-3-319-93566-9_6
Tashakkori, A., Johnson, R. B., & Teddlie, C. (2020). *Foundations of mixed methods research: Integrating quantitative and qualitative approaches in the social and behavioral sciences*. SAGE.
Thune, E. H. (2023). *Status for innføring av algoritmisk tenking og programmering i matematikk* [Masteroppgave, Universitetet i Bergen]. https://bora.uib.no/bora-xmlui/handle/11250/3072541

Turgut, M., Kohanová, I., & Gjøvik, Ø. (2024). Developing survey-based measures of mathematics teachers' pedagogical technology knowledge: A focus on computational thinking and programming tools. *Research in Mathematics Education*, 1–24. https://doi.org/10.1080/14794802.2024.2401482

Utdanningsdirektoratet. (2013). *Læreplan i Matematikk Fellesfag*.

Utdanningsdirektoratet. (2024). *Algoritmisk tenkning*. Digitalisering i barnehage og skole. https://www.udir.no/kvalitet-og-kompetanse/digitalisering/algoritmisk-tenkning/

van Borkulo, S. P., Chytas, C., Drijvers, P., Barendsen, E., & Tolboom, J. (2023). Spreadsheets in Secondary School Statistics Education: Using authentic data for computational thinking. *Digital Experiences in Mathematics Education*, *9*(3), 420–443. https://doi.org/10.1007/s40751-023-00126-5

Vinnervik, P., & Bungum, B. (2022). Computational thinking as part of compulsory education: How is it represented in Swedish and Norwegian curricula? *Nordic Studies in Science Education*, *18*(3), Artikkel 3. https://doi.org/10.5617/nordina.9296

Voldmo, I. (2022). *Undervisningskunnskap i programmering* [Master thesis, OsloMet - Storbyuniversitetet]. https://oda.oslomet.no/oda-xmlui/handle/11250/3032227

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

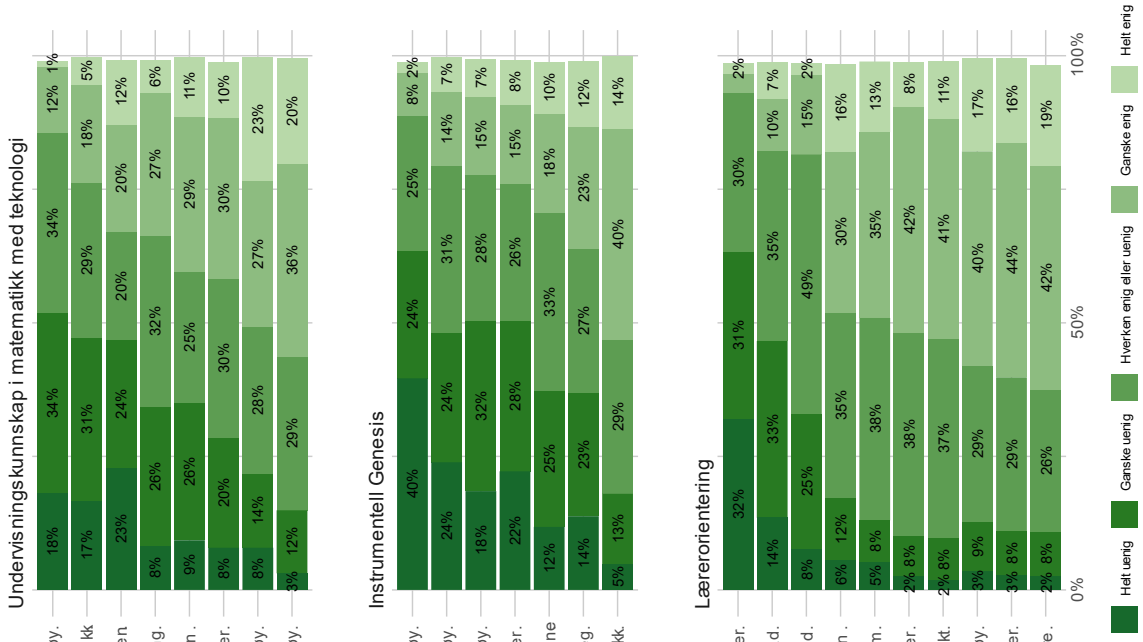
Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, *25*(2), Artikkel 2. https://doi.org/10.17471/2499-4324/922

Yin, R. K. (2009). *Case Study Research: Design and Methods*. SAGE.

8 Appendix

Svar på alle spørsmål fra Turgut m.fl. (2024), inndelt etter deres kategorier.



UNIVERSITETET
I OSLO

EDUCATE

EVALUERING AV FAGFORNYELSEN I FAG