Øving 1, algoritmer og datastrukturer

Oppgave

Oppgave 1–1, 1–2 og 1–3 sidene 18–19 i læreboka.[1] Det fins mange løsninger på problemet, med kompleksitetene $O(n^2)$, $O(n)$, $O(n^3)$, eller til og med $O(n \log n)$. Programmer en av løsningene. Skriv koden selv, dette er altfor smått til at det lønner seg å bruke kode fra nettet. Sjekk først at du får riktig svar ved å sammenligne med eksemplet i boka. Deretter skal du prøve ut kjøretiden ved å prøve forskjellige problemstørrelser. Hvilken kompleksitet har din algoritme? Sammenlign teoretisk kompleksitet med de praktiske målingene.

Presisering: Det skal gjøres ett kjøp og ett salg, som gir best mulig fortjeneste. Ikke flere kjøp og salg, det er et annet problem.

Krav til godkjenning

- Et program, som finner rett resultat. Det kan f.eks. skrives på skjermen. Legg ved utskrift.
- Analyse som forteller hvilken kompleksitet programmet ditt har.
- Tidsmålinger (med flere ulike n), som ser ut til å bekrefte analysen.

Noen tips

- Ikke fått tak i boka ennå? Det er tidlig i semesteret, så jeg har lagt ut boksidene på nett for dere.
- Metoder som bare leter opp de to dagene med absolutt lavest og høyest kurs *vil ikke virke*. Dette fordi dagene kan komme i feil rekkefølge. Man må kjøpe aksjene *før* man selger dem.
- For å teste med store tabeller, kan dere f.eks. bruke math.random for å fylle tabellen med tilfeldige kursforandringer. Pass på at det blir både positive og negative tall. (Ikke ta tiden på initialiseringen, math.random er en treg metode.)
- En vanlig feil på denne oppgaven, er å programmere som om tabellen beskriver *verdien* på aksjene fra dag til dag. Det gjør den ikke, den beskriver hvordan verdien *forandrer seg*. Men det er ikke vanskelig å lage en løkke som regner ut hva aksjeverdien blir på dagene. Det er bare å starte med en passende startverdi, og

[1] Bok: Hafting/Ljosland: *Algoritmer og datastrukturer*

legge til forandringene fra dag til dag. (Startverdi er ikke gitt, men påvirker heller ikke hva som lønner seg best.)

- I forelesningen så vi på triks for å får gode tidsmålinger selv om maskinklokka ikke er helt god.
- For små datasett gir som regel dårlige tidsmålinger uansett. Algoritmer som er $O(n^2)$ trenger ofte målinger med $n > 1000$. Lineære algoritmer kan fort trenge $n > 1000\,000$, fordi PCer er så raske. Og java-klokka har lav oppløsning.
- For å bekrefte en analyse, trengs målinger med flere ulike n. For en lineær algoritme, regner vi med at en tidobling av n, gir omtrent tidobling av kjøretid. Men for en kvadratisk algoritme, $O(n^2)$, regner vi med at kjøretiden hundredobles hvis n tidobles.

  Fortvil ikke, om tidsmålingene er litt usikre. F.eks er en 80-dobling av tidsforbruk klart nærmere en 100-dobling enn en 10-dobling. Tegn gjerne kurver, for å få bedre inntrykk av målingene.
- Ikke ta tiden på utskrift. Gjør ferdig tidsmålingen, før du skriver ut resultatet. Resultatet kan f.eks. lagres i en variabel. Men, skriv ut resultatet til slutt! Hvis ikke, kan kompilatoren optimalisere vekk hele beregningen, fordi resultatet ikke brukes. I så fall blir tidsmålingen urealistisk…