IN2010
Oblig 3

23. oktober 2020

Innlevering

Last opp filene dine på Devilry. Innleveringsfristen er fredag 6. november 2020, kl. 23:59.

Vi anbefaler så mange som mulig om å samarbeide i små grupper på *opp til tre*. Dere må selv opprette grupper i Devilry, og levere som en gruppe (altså, ikke last opp individuelt hvis dere jobber som en gruppe).

Filene som skal leveres er:

- Én PDF som skal hete `IN2010-oblig3.pdf`
- En Java med `main` som skal hete `Oblig3.java`
- Alle Java-filer du har brukt
- Alle `.csv`-filer du har brukt og referert til i `IN2010-oblig3.pdf`
- Alle `.out`-filer som du har klart å generere

Filene skal ikke zippes eller lignende.

Sortering

I denne oppgaven skal du implementere fire sorteringsalgoritmer!

Disse to sorteringsalgoritmene *må* implementeres:

- Insertion sort
- Quicksort

I tillegg skal du implementere to sorteringsalgoritmer, der:

- én er i $O(n^2)$
- én er i $O(n \log(n))$

Her står du helt fri til å velge sorteringsalgoritme selv, så lenge én er i $O(n \log(n))$ og den andre er i $O(n^2)$.

For den ekstra ivrige: Du kan implementere sorteringsalgoritmer som *ikke* er i pensum. Da må du selv sørge for at implementasjonen er forståelig for retteren, og kan ikke forvente hjelp fra gruppelærer i like stor grad.

For alle deloppgavene er input det samme. Det som skiller deloppgavene er output.

Input

Det består av n heltall. For hvert tall i er det slik at $0 \leq i \leq 2^{31} - 1$. Input skal leses fra fil, hvor filnavnet blir gitt som kommandolinjeargument (tilsvarende Oblig 2).

Eksempelinput:

```
80
91
7
33
50
70
13
321
12
```

Deloppgave 1: Korrekthet

For et gitt input skal programmet ditt skrive en fil *for hver* sorteringsalgoritme du implementerer. Hver av dem skal inneholde resultatet av sorteringen på samme format som input. Fordi alle algoritmene skal oppnå samme resultat, er det forventet at alle outputfilene er like.

Navngi hver outputfil etter navnet på inputfilen, etterfulgt av sorteringsalgoritmen som ble brukt til å sortere input. Hvis inputfilen heter `example` skal outputfilen hete `example_alg.out`, der `alg` byttes ut med den aktuelle sorteringsalgoritmen. Algoritmenavnet skal bare inneholde små bokstaver (`a`–`z`). For eksempel kan outputfilen for insertion sort få navnet `example_insertion.out`.

Gitt eksempelinputet ovenfor bør vi få følgende output.

Eksempeloutput:

```
7
12
13
33
50
70
80
91
321
```

Deloppgave 2: Sammenligninger, bytter og tid

Hver av sorteringsalgoritmene skal kunne telle *antall sammenligninger* og *antall bytter* som gjøres i løpet av en sortering. Det kan være lurt å skrive egne metoder for sammenligning og bytter, slik at det blir enkelt å telle. I tillegg skal den totale tiden brukt på sortering oppgis i *mikrosekunder*. Du kan ta tiden på din sorteringsalgoritme (i mikrosekunder) slik:

```
long t = System.nanoTime();

// din sorteringsalgoritme kjøres her

long time = (System.nanoTime()-t)/1000;
```

Du skal kun produsere én outputfil for en gitt inputfil, som en CSV-fil. Dersom inputfilen heter `example`, skal outputfilen hete `example_results.csv`.

Hvis inputfilen inneholder n heltall, skal du gjøre $n + 1$ sorteringer for hver sorteringsalgoritme du tester. Første sortering gjøres på det tomme arrayet, andre sortering gjøres på et array som inneholder det første elementet, tredje sortering gjøres på et array som inneholder de to første elementene, og så videre, helt opp til n.

Den første linjen i output skal se slik ut:

```
n, alg1_cmp, alg1_swaps, alg1_time, alg2_cmp, alg2_swaps, alg2_time
```

Det samme mønsteret skal følges for alle algoritmene du tester. De etterfølgende linjene skal bestå av kommaseparerte heltall.

- Første tall skal angi hvor mange elementer som ble sortert
- Andre tall skal angi hvor mange sammenligninger `alg1` brukte
- Tredje tall skal angi hvor mange bytter `alg1` gjorde
- Fjerde tall skal angi hvor lang tid `alg1` brukte (i mikrosekunder)
- Femte tall skal angi hvor mange sammenligninger `alg2` brukte
- ...

Eksempeloutput:

```
n, insertion_cmp, insertion_swaps, insertion_time, quick_cmp, quick_swaps, quick_time
0,             0,              0,              1,         0,          0,          1
1,             0,              0,              0,         0,          0,          1
2,             1,              0,              1,         1,          2,          3
3,             3,              2,              2,         3,          2,          2
4,             6,              4,              2,         8,          5,          4
5,             9,              6,              2,        12,          8,          5
6,            12,              8,              2,        16,          9,          5
7,            18,             13,              3,        21,         11,          5
8,            19,             13,              4,        26,         12,         10
9,            27,             20,              4,        26,         13,          7
```

Merk at mellomrom ignoreres.

Deloppgave 3: Eksperimentér

Her skal du kun bruke programmet fra deloppgave 2 for å gjøre noen refleksjoner. Vi vil laste opp flere inputfiler på semestersiden som du kan bruke til å kjøre eksperimenter. Du bør kunne kjøre eksperimenter på ulike kombinasjoner av sorteringsalgoritmer (du trenger ikke lage et brukergrensesnitt for dette, men kan kommentere ut/inn kode utifra hva du trenger for å kjøre eksperimentene dine). Det er for eksempel lurt å kjøre eksperimenter hvor kun $O(n \log(n))$ algoritmene kjøres, fordi da vil du kunne teste større inputfiler.

Filene som genereres er CSV-filer, og de kan åpnes i vanlige regnearkverktøy, som Excel, Numbers, etc. Her kan du lage enkle linjediagrammer, som kan gi deg innsikt i hvordan antall sammenligninger, bytter og tid henger sammen, samt hvor effektive de forskjellige sorteringsalgoritmene er.

Vi ønsker at du gjør noen egne refleksjoner. Du må ta stilling til disse spørsmålene:

- I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene (store O) for de ulike algoritmene?
- Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor?
- Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?
- Har du noen overraskende funn å rapportere?

Alle svar må begrunnes. Legg gjerne ved skjermbilder eller lignende som gir grunnlag for påstandendene dine.

Repetisjon av Pensum

De følgende oppgavene tar utgangspunkt i pensumet dekket så langt i kurset og gir en god indikasjon på vanskelighetsgraden for eksamensoppgavene.

Oppgave 2: Mobilnettverk

I denne oppgaven hjelper du et selskap med å etablere et mobilnettverk. Selskapets plan er å benytte seg av eksisterende signaltårn og koble disse sammen med kabler. For at nettverket skal fungere, må alle tårn være i et sammenhengende nettverk, dvs. at hvert tårn må være (direkte eller indirekte via andre tårn) koblet til hvert andre tårn.

Tabellen under viser lengden av kabelen som trengs for en forbindelse direkte mellom to signaltårn. Denne forbindelsen er symmetrisk, dvs. at å forbinde `T1` med `T2` koster det samme som å forbinde `T2` med `T1` (nemlig 10). Tabellen er komplett, som betyr at hvis en forbindelse ikke forekommer i tabellen så er det heller ikke mulig å koble de respektive tårnene sammen direkte (for eksempel er en direktkobling av `T1` og `T7` ikke mulig).

Fra tårn	til tårn	Kabellengde i km
`T1`	`T2`	10
`T1`	`T4`	5
`T4`	`T2`	6
`T2`	`T5`	3
`T4`	`T6`	11
`T5`	`T6`	4
`T7`	`T5`	17
`T7`	`T6`	10
`T3`	`T5`	7

1. Din første oppgave er å beregne en øvre grense for hvor mye kabel selskapet trenger. Hva er kostnaden av den dyreste måten å koble sammen alle signaltårn på? Oppgi svaret ditt som et tall.
2. Hva er den billigste måten å koble sammen alle signaltårn på? Oppgi dette som et tall. Videre, oppgi hvilken algoritme du brukte, samt hvordan du formaliserte problemet. Til slutt, oppgi hvilke andre algoritmer som er blitt dekket i pensum, som man kunne ha brukt i stedet.

Oppgave 3: O Notasjon

I denne oppgaven skal du approksimere kjøretiden til forskjellige algoritmer. Oppgi en kjøretidsanalyse ved bruk av O-notasjon for hver av de følgende algoritmene. Du trenger ikke å telle alle primitive steg, men begrunn svarene dine.

Algorithm 1

Input: Et tall n

```
1 Procedure Algo1(n)
2 |   return n + 1
```

Algorithm 2

Input: To positive tall m og n

```
1 Procedure Algo2(m, n)
2     output = 0
3     for i = 1 to m do
4         for j = 1 to n do
5             output = Algo1(output)
6         end
7     end
8     return output
```

Algorithm 3

Input: Et positivt tall n

```
1 Procedure Algo3(n)
2     output = 0
3     j = n
4     while j > 1 do
5         Algo1(output)
6         j = ⌊j/2⌋
7     end
8     return output
```