Innlevering 3

tysdag 12. mars 2019 12:00

Gruppe 80 for oblig 3

Oppgåve 1:

i) Programlisting og utskrift av køyring av a), b) og c):

```
Summen av dei 100 første naturlege tal: 5050

Dei 10 første ledda (startar med a0) i den gitte tallfølga:
2
5
15
47
147
455
1395
4247
12867
38855
```

Ved 4 diskar - tårna i Hanoi:

```
Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
Move one disk from 1 to 2
Move one disk from 3 to 1
Move one disk from 3 to 2
Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
Move one disk from 2 to 1
Move one disk from 3 to 1
Move one disk from 2 to 3
Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
```

ii) Dokumentasjon av minst 3 ulike tidsforbruk på c):

```
Total moves: 65535
Tal på diskar: 16
Før: 2019-03-19T16:33:08.713018300Z, etter: 2019-03-19T16:33:08.714018600Z.
Tid brukt: PT0.0010003S
```

```
Total moves: 268435455
Tal på diskar: 28
Før: 2019-03-19T16:40:14.447073100Z, etter: 2019-03-19T16:40:14.999197600Z.
Tid brukt: PT0.5521245S
```

```
Total moves: 4294967295
Tal på diskar: 32
Før: 2019-03-19T16:33:26.575050400Z, etter: 2019-03-19T16:33:35.392041Z.
Tid brukt: PT8.8169906S
```

```
Total moves: 17179869183
Tal på diskar: 34
Før: 2019-03-19T16:38:57.167627Z, etter: 2019-03-19T16:39:40.395385800Z.
Tid brukt: PT43.2277588S
```

iii) Sjekke at tid stemmer

Tid n	1. Køyring	2. Køyring	3. Køyring	4. Køyring	5. Køyring	Gjennsomsnitt
Tid32	9.083554	8.8144941	8.8239918	8.8239309	8.8364996	8,87649408
Tid16	0.0009999	0.0010003	0.0010006	0.0010003	0.0010003	0,00100028

(2^32 - 1) / (2^16 - 1) = 65 537

8,87649408 / 0,00100028 = 8 874,00935738

Får ikkje lik verdi... Noko eg gjere feil?
(n = 64 tar for lang tid... Sjølv på ein rimeleg rask PC)

Oppgåve 2:

2 - 3 - 6 (midten = 3) odde
2 - 3 - 4 - 6 (midten = 4) -> par, måtte oppdaterast (la til bak midten)
2 - 3 - 4 - 5 - 6 (midten = 4) -> odde
berre ein oppdatering på 2 lagt til

2 - 3 - 4 - 5 - 6 (midten = 4) odde
2 - 3 - 4 - 6 (midten = 4) -> par
2 - 3 - 6 (midten = 3) -> odde, måtte oppdaterast (fjerna bak midten)
berre ein oppdatering på 2 fjerna

a. ok
b. ok
c. nyMidten er O(n). Det er fordi me har ei for-løkke som vil gjenta seg antall / 2 ganger som då er lik n/2 og det er då O(n). Kan oppdatere metoden til å berre vere O(1)? **Forklar:**

   Tenkjer to nye metodar, ein for å kalkulere midten når ein legg til og ein når ein fjernar eit element.
   Legg til: sjekkar om elementet som blei lagt til kjem før eller etter midten og så flyttar midten opp og ned basert på det + om det nye antallet er oddetal eller partal (treng ikkje nødvendigvis alltid å flytte midten). Legg til før -> flytte midten ned + sjekke tal på element om ein faktisk skal flytte. Legg til etter -> flytte midten opp + sjekke tal på element om ein faktisk skal flytte.
   Fjerne: som i legg til, men omvendt her. Altså fjernar etter -> flytte midten ned, og fjernar før -> flytte midten opp, men ein må sjekke talet på element då før ein flyttar.

   Ein sjekkar talet på element då ved f. Eks. 3 element allereie så peikar midten på den i midten. Om ein då legg til ein før midten blir midten dytta opp på "rett plass" automatisk (om ein vel at den til høgre av dei i midten ved tal på element er eit partal), mens om ein hadde lagt til bak måtte ein ha flytta midten opp manuelt.

   Ein må også sjekke på når det er tomt og ein legg til eit nytt element, og når ein fjernar det siste elementet.

   Altså:
   - Legg til bak + blir partal -> ny midten (opp)
   - Legg til bak + blir oddetal -> gjer ingenting
   - Legg til før + blir partal -> gjer ingenting
   - Legg til før + blir oddetal -> ny midten (ned)

   - Fjerne bak + blir partal -> gjer ingenting
   - Fjerne bak + blir oddetal -> ny midten (ned)
   - Fjerne før + blir partal -> ny midten (opp)
   - Fjerne før + blir oddetal -> gjer ingenting

   Dette blir berre ~ein-to samanlikning(ar) + sjekke tal på element. Ikkje noko løkker eller slikt, berre if-else setningar.

   (kom ikkje på noko anna, sikkert noko som er betre / kan gjerast i ein metode... Men eg kjem ikkje på nå...)

d. Kor mange noder må ein undersøkja i gjennomsnitt / verste tilfelle når ein leitar etter ein dataverdi som **finst** i ei lista på n element.
   i) Liste utan midtpeikar:
      Gjennomsnitt: ca. n/2 element

Verste tilfelle: n element

ii) Liste med midtpeikar og berre søking ein veg:
Gjennomsnitt: ca. n/4 (deler lista i 2)
Verste tilfelle: ca. n/2 element

iii) "Foretar ein enkel utrekning og samanlikning (du må sjølv komme på), skal med midtpeikar avgjere korleis ein skal søkje:"
(Noder ein må undersøkje i gjennomsnitt / verste tilfelle på n element)

1) Midten mot første:
   Gjennomsnitt: ca. n/4
   Verste: ca. n/2 (om det første er langt mindre enn tala etter)
2) Første mot midten:
   Gjennomsnitt: ca. n/4
   Verste: ca. n/2 (om midten er langt større enn tala før)
3) Midten mot siste:
   Gjennomsnitt: ca. n/4 (halve av halve)
   Verste: ca. n/2 (om det siste er langt større enn tala før)
4) Siste mot midten:
   Gjennomsnitt: ca. n/4
   Verste: ca. n/2 (om midten er langt mindre enn tala etter)

(blei litt usikker på d) iii) - veit ikkje om det blei heilt rett...)

Eksempel tabell:

0	2	4	6	8	**10**	12	14	16	18	*99*

0 og 99 er kunstige, 10 er midten, n = 9

Eksempel tabell 2:

0	1	2	3	**4**	5	6	*99*

0 og 99 er kunstige, 4 er midten, n = 6

Oppgåve 3:

a. Forklare binærsøking ved å presentera ein algoritme:
Sjekke om elementet ein søkjer etter er likt som det i midten
Viss ikkje -> sjekke om elementet ein søkjer etter er større eller mindre enn elementet i midten.
Mindre -> søkje rekursivt på deltabellen som er dei elementa som ligg før midten
Større -> søkje rekursivt på deltabellen som er dei elementa som ligg etter midten

Ein søkjer rekursivt til ein kjem til basistilfellet (det er likt elementet i midten) -> funne det ein søkjer etter (og gir tilbake posisjonen), eller det ikkje er fleire element i deltabellane -> ikkje funne elementet (gir tilbake -1 eller noko).

Sjekking om det er likt / mindre / større enn kan gjerast ein gong ved å ta vare på compareTo verdien.

Sortert tabell:

2	4	5	7	8	10	12	**15**	18	21	23	27	29	30	31

b. Søkje etter 8 (som finst):
Er 8 lik 15? Nei.
Er 8 mindre enn 15? Ja.
-> søkjer i tabellen under 15 (min = 0, max = 6):

2	4	5	**7**	8	10	12

Er 8 lik 7? Nei.
Er 8 mindre enn 7? Nei.
-> søkjer i tabellen over 7 og under 15 (min = 4, max = 6):

8	**10**	12

Er 8 lik 10? Nei.
Er 8 mindre enn 10? Ja.
-> søkjer i tabellen under 10 og over 7 (min = 4, max = 4):

8	

(Eit element i tabellen / min == max:) Er 8 lik 8? Ja
-> funne på posisjon 4, returnerer ***4***

c. Søkje etter 16 (som ikkje finst):

Er 16 lik 15? Nei.
Er 16 mindre enn 15? Nei.
-> søkjer i tabellen over 15 (min = 8, max = 14):

18	21	23	**27**	29	30	31

Er 16 lik 27? Nei.
Er 16 mindre enn 27? Ja.
-> søkjer i tabellen under 27 og over 15 (min = 8, max = 10):

18	**21**	23

Er 16 lik 21? Nei.
Er 16 mindre enn 21? Ja.
-> søkjer i tabellen under 21 og over 15 (min = 8, max = 8):

18	

(Eit element i tabellen / min == max:) Er 16 lik 16? Nei
-> ikkje funne, returnerer ***FINNES_IKKJE (-1)***

(tok vekk at ein reknar ut midten = (min + max) / 2 og bestemmer nye min / max)
(-> søkjer i tab = rekursivt kall på søkinga)
(er lik / mindre / større enn er ein operasjon med compareTo(), tar berre vare på verdien)

Tal på rekursive kall er log2n (runda opp) i verste tilfelle ved binærsøking (- det første metodekallet).

Korleis stemmer det med søk av 16 som ikkje finst?
Det er 15 verdiar, $\log_2(15)$ (runda opp) = **4**
Ved søk av 16 blir metoden kalla rekursivt 3 ganger
-> 1 sml i første metodekall + 3 rekursive kall = **4** compareTo samanlikningar
Dei to verdiane er like, så det stemmer.

Oppgåve 4:

a. Måle tid på soteringsmetoder:

T(n) = c*f(n)
Må bestemme c, varierer på algoritme, implementasjon og maskina.
f(n) er lik n^2, $n*\log_2 n$ eller n

<u>Alle tider er i sekunder</u>

Resultat Quick-sort

n	**Tal på målingar**	**Målt tid (gjennomsnitt)**	**Teoretisk tid c*n*log₂n**
32000	1000	0.004313973	0,004383176
64000	1000	0.009352111	0,009352111 **(fant c)**
128000	1000	0.019727453	0,019875741

T(62000) = 0.009352111 = c * 64000 * $\log_2(64000)$
C = 9,152493343 * $10^{(-9)}$

Dei teoretiske tidene passa ganske godt når eg brukte mange målingar. Ved få målingar hadde dei teoretiske tidene store avvik. Eg brukte også n = 64000 som brukar litt lengre tid for å få fleire liknande målingar slik at gjennomsnittet blir betre. Det er nok viktigare å bruka mange målingar på den ein skal finne **c** på enn dei andre, men mindre målingar (kan) gi større avvik.

Resultat flettesortering

n	**Tal på målingar**	**Målt tid (gjennomsnitt)**	**Teoretisk tid c*n*log₂n**
32000	1000	0.007546207	0,007863198
64000	1000	0.016777219	0,016777219 **(fant c)**
128000	1000	0.035382923	0,035656085

T(64000) = 0.016777219 = c * 64000 * $\log_2(64000)$
C = 1,641911491 * $10^{(-8)}$

Dei teoretiske tidene passa ganske godt når eg brukte mange målingar. Ved få målingar hadde dei teoretiske tidene store avvik. Eg

brukte også n = 64000 som brukar litt lengre tid for å få fleire liknande målingar slik at gjennomsnittet blir betre. Det er nok viktigare å bruka mange målingar på den ein skal finne **c** på enn dei andre, men mindre målingar (kan) gi større avvik.

Resultat boblesortering

n	**Tal på målingar**	**Målt tid (gjennomsnitt)**	**Teoretisk tid c*n²**
32000	10	3.07009308	3,82208488
64000	10	15.28833952	15,28833952 **(fant c)**
128000	5	64.38772788	61,15335808

$T(64000) = 15.28833952 = c * 64000^2$
$C = 3{,}732504766 * 10^{(-9)}$

Desse er så ineffektive for store verdiar for n at eg måtte gå ned på talet på målingar då det tok for lang tid... Avvika blir difor større. Kunne nok ha funne c med n = 32000 (eller mindre) med langt fleire målingar for å (kanskje) få litt mindre avvik.

Resultat sortering ved utval

n	**Tal på målingar**	**Målt tid (gjennomsnitt)**	**Teoretisk tid c*n²**
32000	20	0.646796015	0,825679646
64000	20	3.302718585	3,302718585 **(fant c)**
128000	10	15.14134991	13,21087434

$T(64000) = 3.302718585 = c * 64000^2$
$C = 8{,}063277795 * 10^{(-10)}$

Desse er så ineffektive for store verdiar for n at eg måtte gå ned på talet på målingar då det tok for lang tid... Avvika blir difor større. Kunne nok ha funne c med n = 32000 (eller mindre) med langt fleire målingar for å (kanskje) få litt mindre avvik.

Resultat sortering ved innsetting

n	**Tal på målingar**	**Målt tid (gjennomsnitt)**	**Teoretisk tid c*n²**
32000	30	1.002324213	1,3775425
64000	20	5.5101698	5,5101698 **(fant c)**
128000	8	26.381066475	22,040679

$T(64000) = 5.5101698 = c * 64000^2$
$C = 1{,}345256299 * 10^{(-9)}$

Desse er så ineffektive for store verdiar for n at eg måtte gå ned på talet på målingar då det tok for lang tid... Avvika blir difor større. Kunne nok ha funne c med n = 32000 (eller mindre) med langt fleire målingar for å (kanskje) få litt mindre avvik.

b. Kvikk- / flettesortering:

Kvikksortering er ca. 1,79 gangar raskare enn flettesortering med mine målingar.
n = 32000 gir 1,749247619 ~= 1,75 gangar raskare.
n = 64000 gir 1,793949943 ~= 1,79 gangar raskare.
n = 128000 gir 1,793588001 ~= 1,79 gangar raskare.

c. Radix-sortering
Best tid/mest effektiv ved bruk av sirkulær kø. Kvifor? - Fordi det å leggje til og fjerne i frå ein sirkulær kø er begge O(1).

Usortert tabell med 3-sifra tal som ein ønskjer å sortere med radix-sortering:

123	398	210	019	528	513	129	294

Vis ved å lage figur korleis nøklane blir fordelt i køar for dei 3 fasane:

1) Sifferet lengst til høgre (elementet lengst "framme"/mot venstre er det første i køen)

0-kø	210	
1-kø		
2-kø		
3-kø	123	513
4-kø	294	
5-kø		

6-kø		
7-kø		
8-kø	398	528
9-kø	019	129

"Ny" tabell:

210	123	513	294	398	528	019	129

2) Sifferet i midten

0-kø			
1-kø	210	513	019
2-kø	123	528	129
3-kø			
4-kø			
5-kø			
6-kø			
7-kø			
8-kø			
9-kø	294	398	

"Ny" tabell:

210	513	019	123	528	129	294	398

1) Sifferet lengst til venstre

0-kø	019	
1-kø	123	129
2-kø	210	294
3-kø	398	
4-kø		
5-kø	513	528
6-kø		
7-kø		
8-kø		
9-kø		

"Ny" tabell (sortert):

019	123	129	210	398	513	528

Forklar korleis me til slutt kan få data sortert:
Ein legg først inn tala i ein av køane basert på einerplassen til talet. Når ein då tar ut av alle køane (startar med 0 og går oppover) vil dei ligge sorterte etter einerplassen.
Ein gjentar dette, men nå med sifferet på tiarplassen til talet. Når ein har tatt tala ut av køane er tala sorterte på tiarane, så einarane.
Til slutt gjentar ein det igjen, men nå med sifferet på hundrarplassen til talet. Når ein då tar ut tala ein siste gang vil dei vere sorterte på hundrarane, så tiarane, og så einarane. Dermed er tabellen blitt sortert.
(ved lengre tal gjentar ein det med tusen-plassen, osv.)

d. Ny kvikksorteringsmetode ved bruk av innsetting i tillegg:

MIN	**n**	**Tal på målingar**	**Ny kvikksort (gjennomsnitt)**	**Gammal kvikksort (gjennomsnitt)**	**Differanse (ny - gammal)**
10	8000	4000	0.000847191	0.001054988	-0,000207797
10	16000	4000	0.001783402	0.00203407	-0,000250668
10	32000	3000	0.004112261	0.00463838	-0,000526119
10	64000	2000	0.008723197	0.008640165	0,000083032

10	128000	1000	0.019599368	0.02147929	-0,001879922
20	32000	3000	0.004198614	0.004278957	-0,000080343
50	32000	3000	0.004516019	0.004278957	0,000237062
100	32000	3000	0.005412221	0.004278957	0,001133264
150	32000	3000	0.006641334	0.004278957	0,002362377
200	32000	3000	0.007621888	0.004278957	0,003342931

Feil i koden i oppgåva! Det var kall til kvikksort og ikkje kvikksort2 inni kvikksort2 (difor eg fekk så "rare" tal) - skulle vel ha sett dette då...

Når MIN er liten så er den nye kvikksorteringa "ein del" / litt raskare. Kan sjå ut som om det veks litt med n, altså det blir meir effektivt, men det kan også vere unøyaktig tidsmåling.
Når MIN derimot veks så blir den nye sorteringa mindre og mindre effektiv og dette skjer ganske fort rundt MIN = 20+. Dette skjer jo på grunn av at sortering ved innsetting er $O(n^2)$ som veks langt raskare enn kva vanleg kvikksortering er ($O(n*\log_2 n)$).
20 * 20 = 400 mens 20 * log2(20) ~= 86
30 * 30 = 900 mens 30 * log2(30) ~= 147
Større "deltabellar" betyr meir flytting på element ved innsettingssortering -> tar lengre tid.