DAT102 Innlevering 3

Oppg 1a)

Antall element brukt for våre observasjoner er 50.000 ettersom det er «run'et» på en gammel mac. Når vi modifiserte koden tok vi først å observerte kor lang tid programmet brukte når vi ikkje hadde sortert første elementet og vi hadde boolean slik som i oppgaveteksten. Etter fem forsøk fekk me ein gjennomsnittlig run-time på **4.424sek**. Deretter tok vi å endra koden slik at den ikkje lengre brukte boolean ferdig, noe som gav oss et utslag på run-time'en vår. Etter fem forsøk hadde me nå en lengre gjennomsnittlig run-time, nå oppe i **5.529sek.**

Nå som vi hadde prøvd med og uten boolean tok vi å endra koden slik at vi manuelt sorterte første element, slik at den var plassert riktig. Med denne endringa prøvde vi å run'e uten boolean først, noko som gav en gjennomsnittlig run-time på **6.654sek**, deretter med boolean, **5.482sek**.

I våras tilfelle er det altså original koden som run'et raskast med en gjennomsnittlig run-time på 4.424sek. Det er derimot en god del feilkilder og ta i betraktning, derav det faktum at testane blei gjort på en litt gammel mac, noko som kan gje veldig varierande og upålitelige resultat. Det er også et viktig punkt å ta i betraktning at enkelte løsninger er betre for små program, mens andre kan være meir ideelle for større program.

Oppg 1b)

I oppgåve b har vi endra koden til å ta å sortere to element om gangen. Til vår overraskelse endra ikkje dette noko særlig på run-time'en vår, ettersom me no har en gjennomsnittlig run-time på 6,312sek. Det er, visst me skal stole på desse resultata som er basert på svært få forsøk, faktisk treigare enn nokre av observasjonane i frå oppgåve a der vi sorterte kvart element for seg sjølv.

Oppg 2a)

NB! Run'er på en gammel pc, derfor er det veldig store «run-times»
Usikker på at teoretisk tid er forstått riktig, men har skrive det eg fekk ved formelen
C = T(n) / f(n)

Quicksort

c = 0,000037397

n	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid c*f(n)
32.000	10	17,91 s	17,91 s
64.000	10	114,43 s	38,21 s
128.000	10	403,23 s	81,21 s

InsertionSort

c = 0,000000020039

n	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid c*f(n)
32.000	10	20,52 s	20,52 s
64.000	10	113,13 s	82,08 s

128.000	10	509,47 s	328,32 s

SelectionSort

c = 0,000000014746

n	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid c*f(n)
32.000	10	15,10 s	15,10 s
64.000	10	72,71 s	60,40 s
128.000	10	252,09 s	241,60 s

MergeSort

c = 0,0000012528

n	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid c*f(n)
32.000	10	0,60 s	0,60 s
64.000	10	1,11 s	1,28 s
128.000	10	1,30 s	2,72 s

Oppgåve 2b)

Når me run'et ein tabell med like tabell såg me at me fekk ein mykje lavere run-time. Me kjørte programmet med 32000 like tall og sammenlignet det med 32000 random tall.

- Like: 3.82 s
- Random: 10.00 s

Oppgåve 3a)

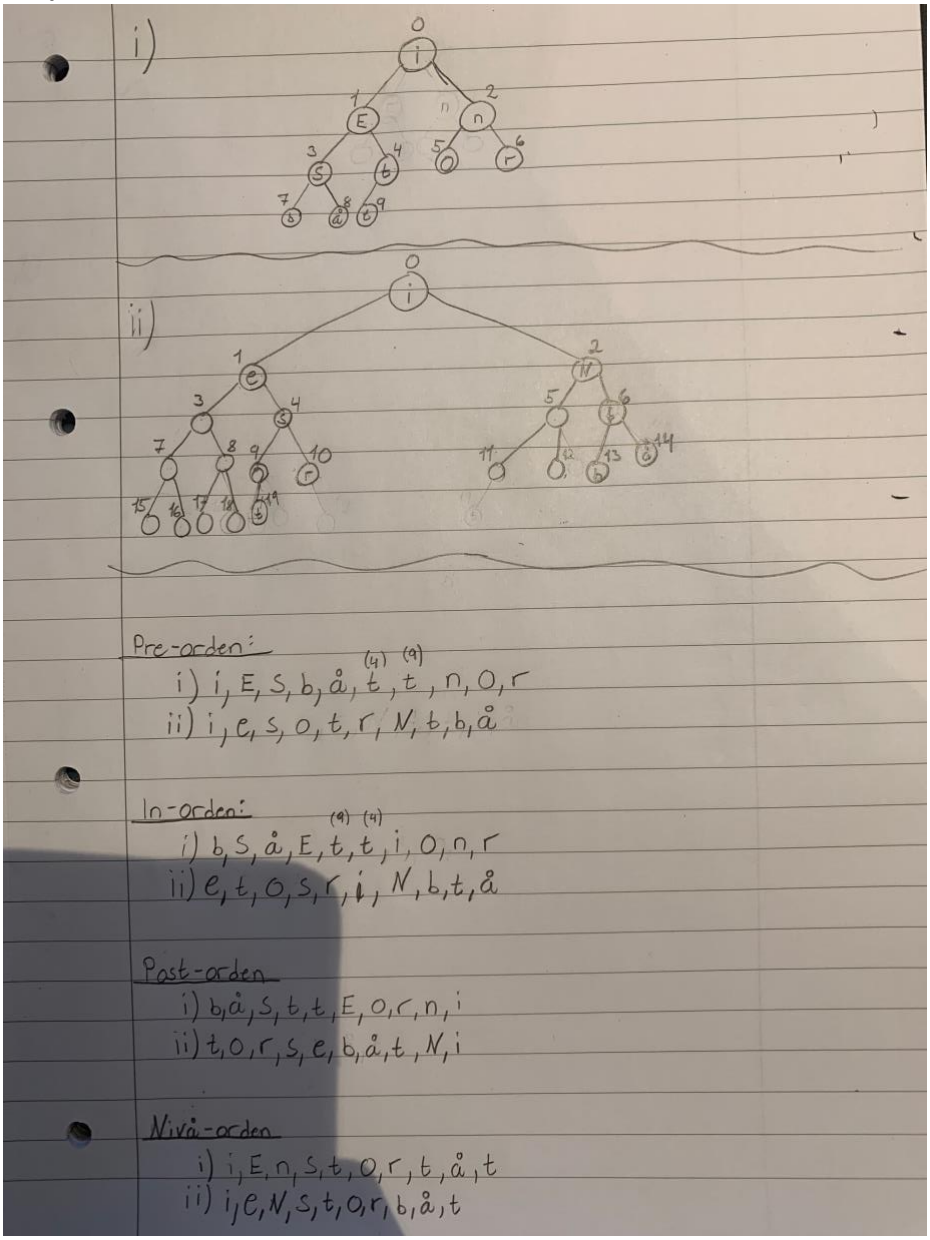
Binært tre: Ein datastruktur som «grener ut» som eit tre, der kvar node har to nye sub noder. Deretter blir kvar nye sub node ein «parent» til to nye sub noder. Den første noden i treet kalles rot noden, slik som eit tre har røter.

Høgda til et binært tre: Definerast som det største antallet kantar frå roten (første noden) til den yttermest liggande noden (bladet).

Fullt binært tre: Eit tre der alle noder, unntatt muligens bladene, har to sub noder. Det må altså enten ha 0 eller 2 sub noder.

Komplett binært tre: Eit tre der alle noder ligger så langt til venstre så mulig, og alle lag i treet er heilt fulle med unntak av dei siste nodene.

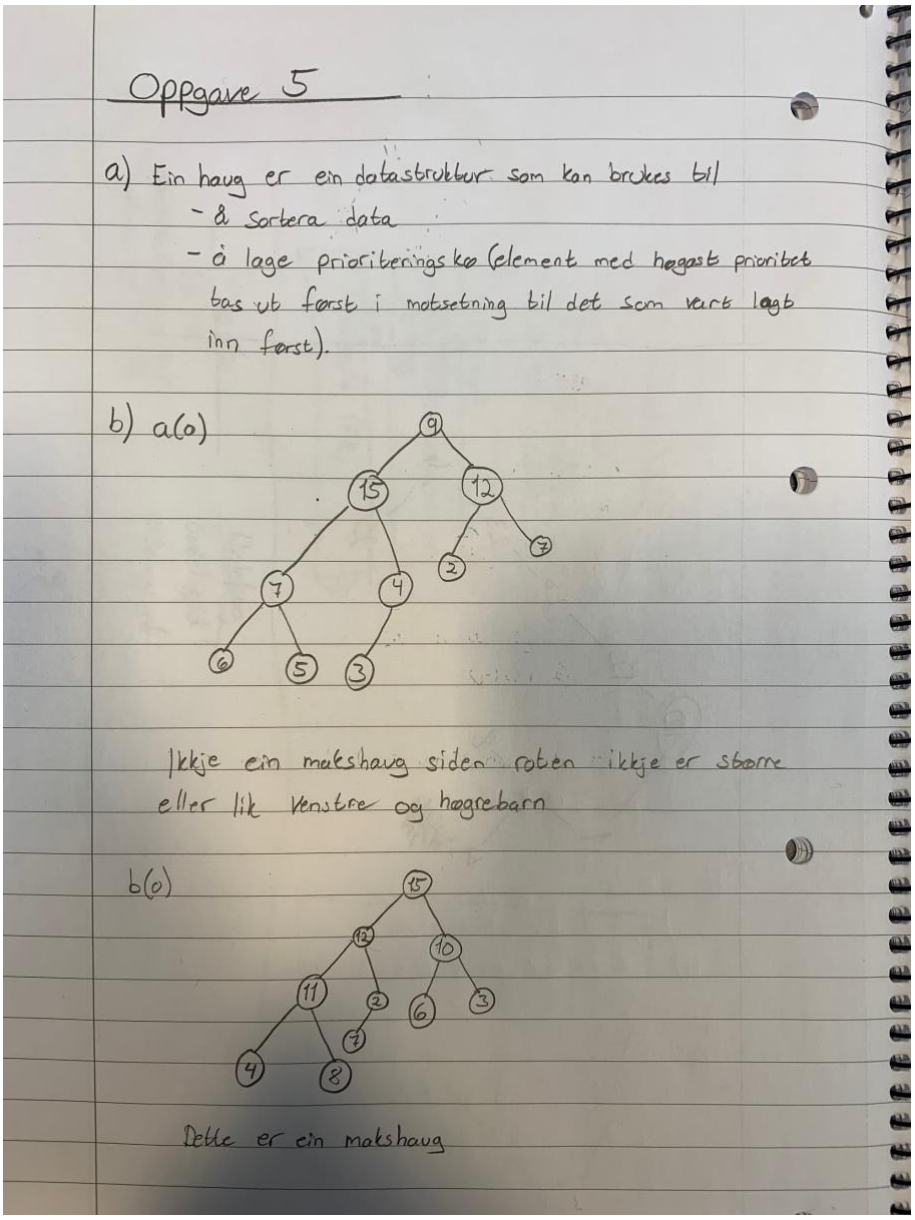
Oppgåve 3b, c)



Oppgave 4c)

Ved utrekning av formelen fekk me at konstanten c er lik 13. Dette er derimot den minste høyden den kan ha (komplett tre). Me fekk derimot ei gjennomsnittlig høgde på 28. Antar at me ikkje har forstått formelen heilt riktig.

Oppgave 5a-e)



Oppgave 5

a) Ein haug er ein datastruktur som kan brukes til
- å sortera data
- å lage prioriteringskø (element med høgast prioritet tas ut først i motsetning til det som vart lagt inn først).

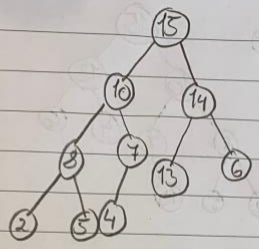
b) a(a)

Ikkje ein makshaug siden roten ikkje er større eller lik venstre og høgrebarn

b(a)

Dette er ein makshaug

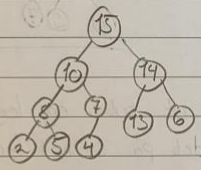
c(0)



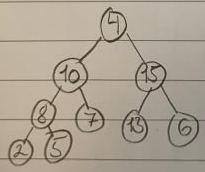
Dette er ein makshaug

5c) Før noko er fjernet:

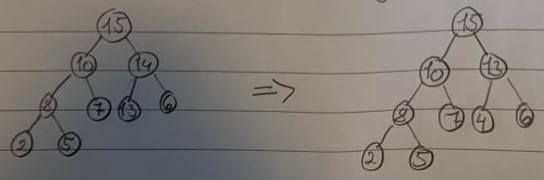
i)



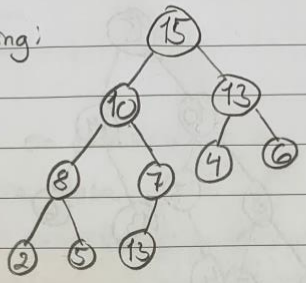
Tar ut 14, og setter 4 i roten:



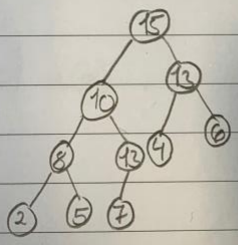
Samenlikner med største barn og byttar med dem:



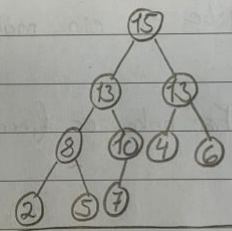
ii) Før innsetting:



Sammenlikner med foreldrer og bytter om:



=>



d) Du kan alltid ta ut rota og legge den i ei liste og då vil du få det på sortert form.

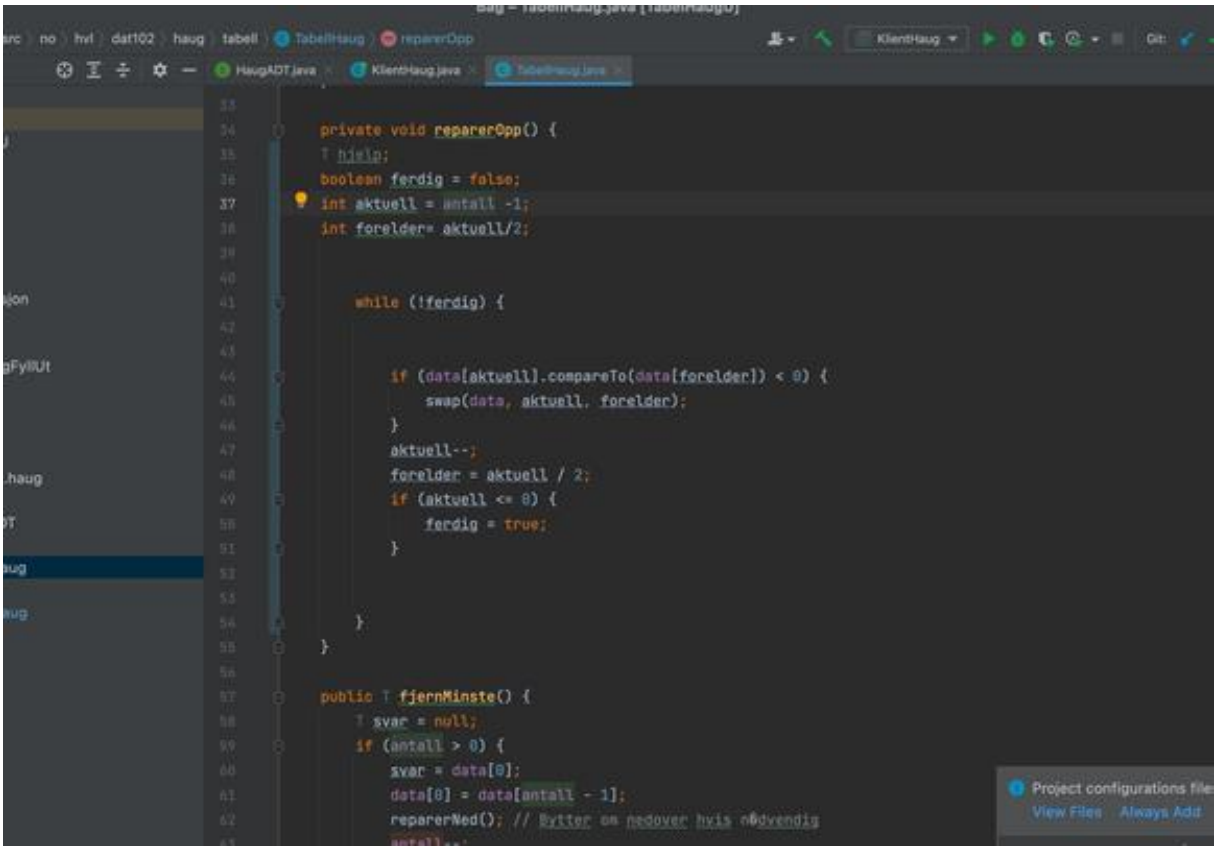
Fordi rota alltid er det minste tallet

Oppgåve 5f)

Hjelpemetode:

```
    }
        public void swap(Object[] tab, int pos1, int pos2){
            Object temp = tab[pos1];
            tab[pos1] = tab[pos2];
            tab[pos2] = temp;
        }
        }
```

Metoden vår:



Konsoll-utskrift: