

## OBLIG 3

### Oppgave 1

a)

Vi modifiserte koden slik at første tallet i tabellen er det minste tallet, så sorterte vi tabellen. Vi runnet da dette med 100 000 element i tabellen. Gjennomsnittstiden ble da 7.084 sekund.

Deretter prøvde vi å bytte til å skrive kun «true» inni i while-løkken i metode «sorteringVedInnsetting» og skrive break på slutten. Så runnet vi koden på nytt etter modifiseringen og fikk en gjennomsnittstid på 8,9625 sekund. Dermed ser vi at koden runner litt senere med denne modifikasjonen.

b)

I denne oppgaven modifiserte vi koden til å sortere 2 element om gangen isteden for bare 1. Selv om koden nå sorterer to element om gangen ser vi at run-timen er nesten helt identisk med i oppgave a.

### Oppgave 2a

Ved teoretisk tid brukte vi formelen  $c = T(n) / f(n)$  for å finne c.

#### Kvikksortering:

$$C = 0,00001309$$

n	Antall målinger	Målt tid, s (gjennomsnitt)	Teoretisk tid $C * f(n)$
32000	10	6,27 s	6,27 s
64000	10	24,24 s	13,38 s
128000	10	199,00 s	28.42 s

#### Utvalgssortering:

$$C = 0,000000007167$$

n	Antall målinger	Målt tid, s (gjennomsnitt)	Teoretisk tid $C * f(n)$
32000	10	7,34 s	7,34 s
64000	10	29,14 s	29,35 s
128000	10	117,53 s	117,42 s

Sorter ved innsetting:

C = 0,000000007216

n	Antall målinger	Målt tid, s (gjennomsnitt)	Teoretisk tid C * f(n)
32000	10	7,39 s	7,389 s
64000	10	24,82 s	29,55 s
128000	10	220,73 s	118.22 s

Flettesortering:

C = 0,0000003549

n	Antall målinger	Målt tid, s (gjennomsnitt)	Teoretisk tid C * f(n)
32000	10	0,17 s	0,17 s
64000	10	0,276 s	0,36 s
128000	10	0,54 s	0,77 s

b)

Vi ser at når vi kvikksorterer en tabell med 32 000 like element bruker vi i snitt 0,2 sekund, mens når vi sorterer en tabell med 32 000 tilfeldige element bruker vi i snitt 0,77 sekund. Dermed observerer vi at de går raskere å sortere en tabell med like element.

### OPPGAVE 3

a)

Binært tre:

en datastruktur som grener ut som et tre. Kvar node har to nye subnoder.

Deretter blir kvar nye sub node en «parent» til to nye sub noder. Den første noden i treet kalles rot noden, slik som en tre har røter.

**Høgda til et binært tre:**

Defineres som det største antallet kanter fra roten (første noden) til den ytterstliggende noden (bladet).

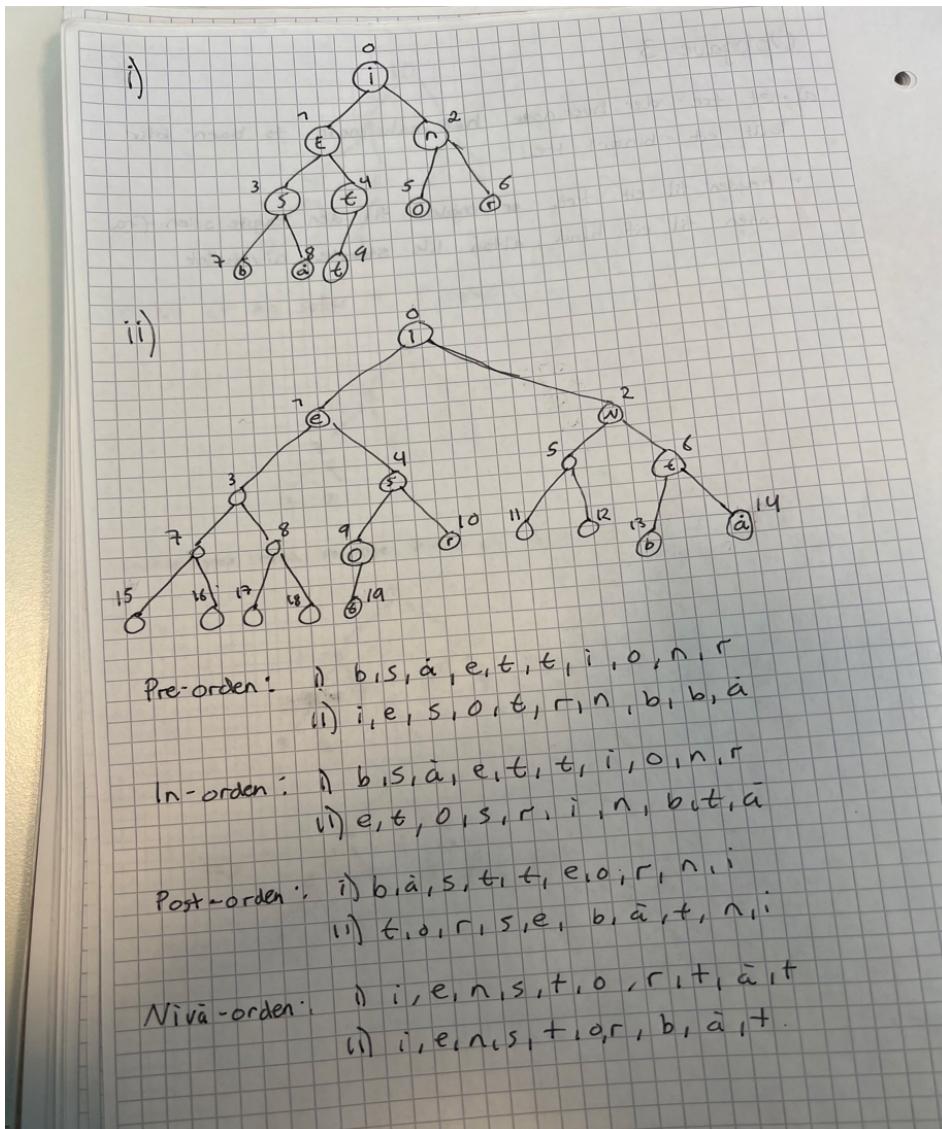
### Fullt binært tre:

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

### Komplett binært tre:

Et tre der alle noder ligger så langt til venstre så mulig, og alle lag i treet er heilt fulle med unntak av de siste nodene.

b g c)



## Oppgave 4

c)

Me brukte formelen for høgda c og fikk som svar at konstanten c er lik 13. Det samsvarer lite med svaret vi fikk i java der vi fikk at den gjennomsnittlige høgda var 29. Derfor antar vi kanskje at de er noe vi har gjort feil i utrekningen.

## OPPGAVE 5

Oppgave 5 oblig 3 Dat.1C2

a) ein haug er ein datastruktur som kan brukes til

- å sortere data
- å lage prioriteringskø (element med høyest prioritet tas ut først i motsetning til det som vart lagt inn først).

b) a(0)

```
graph TD; 15 --- 7; 15 --- 12; 7 --- 6; 7 --- 5; 12 --- 2; 12 --- 1;
```

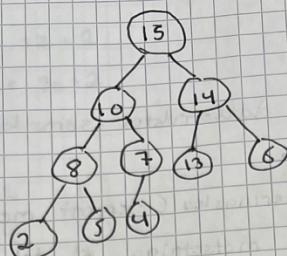
= ikke en maks haug siden rotnode ikke er  $\geq$  venstre og høyre barn.

b(0)

```
graph TD; 15 --- 12; 15 --- 10; 12 --- 11; 12 --- 2; 10 --- 6; 10 --- 3; 11 --- 4; 11 --- 8;
```

= Dette er en maks haug

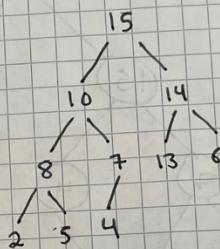
c(0)



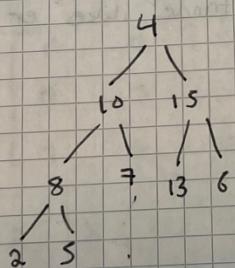
= Dette er en maksbaum

5c) For noe er fjernet:

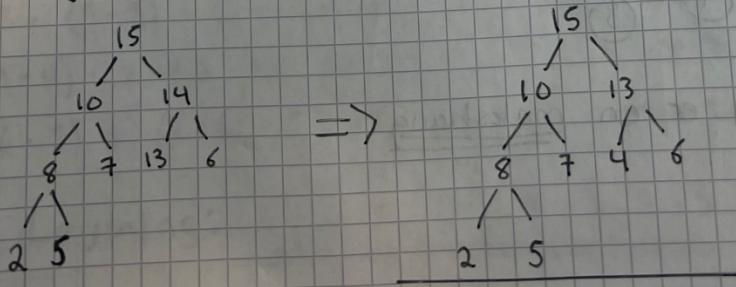
i)



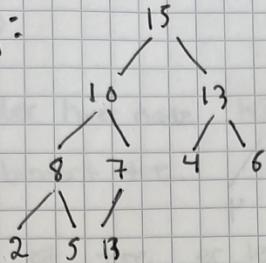
Tar ut 14, og setter 4 i rotposisjon:



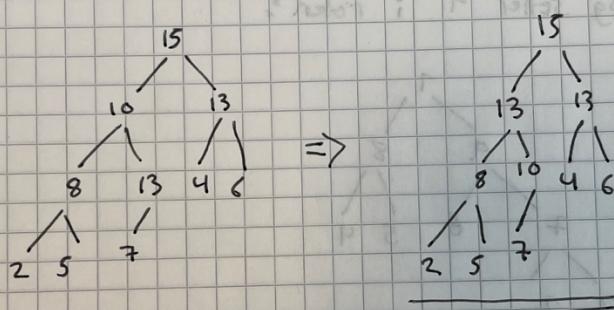
Samenligner med største barn og bytter med den:



ii) Før innsetting:



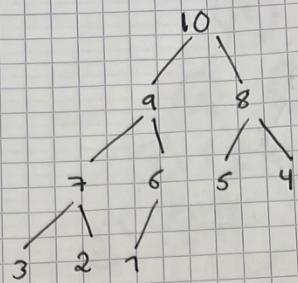
Sammenligner med foreldrer og bytter om:



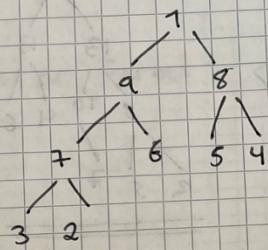
d) Du kan alltid ta ut roten å legge det i en liste  
og da vil du få det på sortert form.

Fordi roten alltid er det minste tallet.

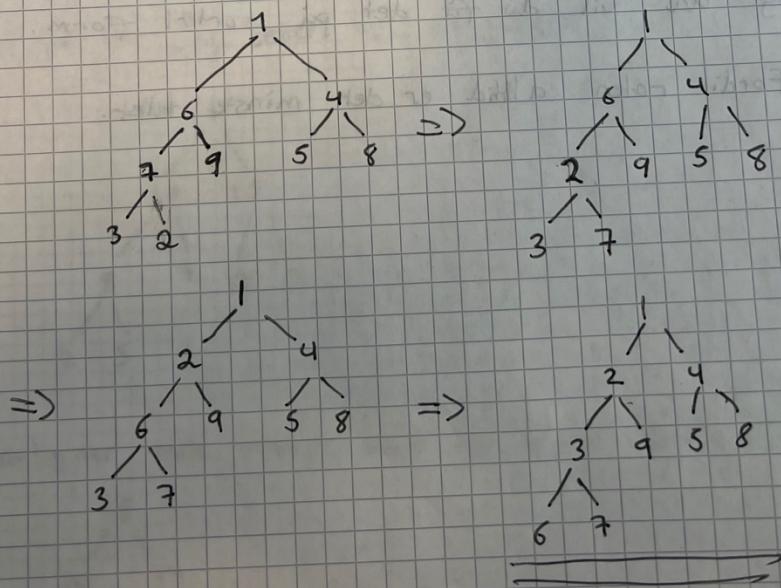
e)



fjerner 10 og setter 1 i rotten:



Sammenligner med minste barn og bytter om:



### Oppgave 5f)

Javakode:

```
private void reparerOpp() {
    T hjelpt;
    boolean ferdig = false;
    int aktuell = antall -1;
    int forelder= aktuell/2;

    while (!ferdig) {
        if (data[aktuell].compareTo(data[forelder]) < 0) {
            swap(data, aktuell, forelder);
        }
        aktuell--;
        forelder = aktuell / 2;
        if (aktuell <= 0) {
            ferdig = true;
        }
    }
}

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

**Bilde av kjøring:**

The screenshot shows a terminal window titled "Run: KlientHaug". The window contains the following text output:

```
/Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Cor  
Verdiene i tabellen er nøy:  
1 2 18 10 33 100 30 200 300 54  
Haugen i sortert rekkefølge:  
1 2 10 18 30 33 54 100 200 300  
Process finished with exit code 0
```

The terminal interface includes standard icons for file operations (New, Open, Save, Print, Find, Delete) and a toolbar with various icons.