DAT102

V15

Oblig 4

Gruppedeltakarar:

Navn	Studentnr
Jonas Holme	h146176
Nils Terje Krumsvik	h146160
Martin Anaton Robertsen	H146167

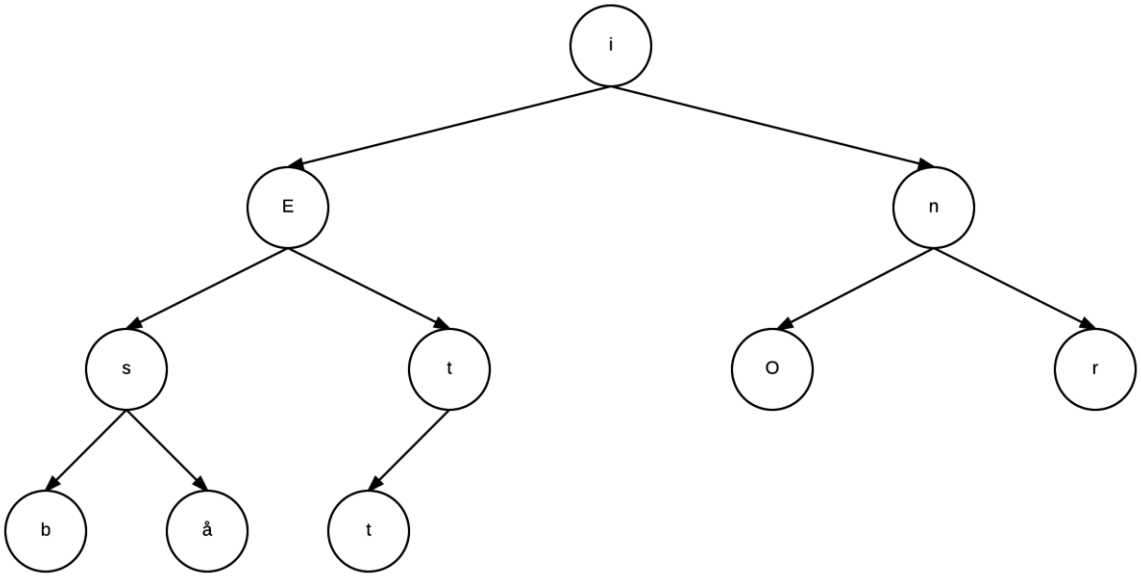
Oppgave 1

a)

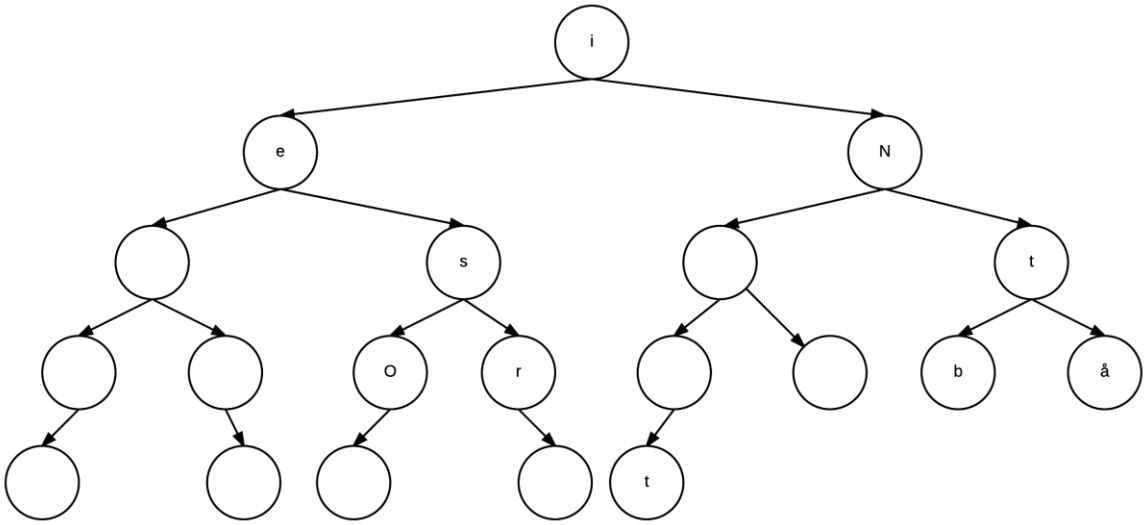
Et binært tre er et tre der man kan ha maks to undertre per node. Denne datastrukturen sørge for at det blir fort mange nivåer som disse nodene ligger på.

b)

i)



ii)



c)

i)

0	1	2	3	4	5	6	7	8	9
i	E	s	b	å	t	t	n	O	r

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
i	e						s	O		r		N			t		t	b	å

ii)

0	1	2	3	4	5	6	7	8	9
b	s	å	E	t	t	i	O	n	r

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
					e		O	s	r		i	t				N	b	t	Å

iii)

0	1	2	3	4	5	6	7	8	9
b	å	s	t	t	E	O	r	n	i

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
						O		r	s	e	t				b	å	t	N	i

iv)

Nivå-orden for disse binære trærne blir akkurat samme som de som er blitt oppgitt i oppgaveteksten.

Oppgave 2

a)

Rekursiv metode for å finne høyden til det binære treet.

```
private int findBinaryTreeHeight(BinaerTreNode<T> element) {

   if (element == null) {
      return 0;
   }
   //return 1 + Math.max(findBinaryTreeHeight(element.getVenstre()),
findBinaryTreeHeight(element.getHoyre()));

   int lefth = findBinaryTreeHeight(element.getVenstre());
   int right = findBinaryTreeHeight(element.getHoyre());
```

```
    if (lefth > right) {
        return lefth + 1;
    } else {
        return right + 1;
    }

}
```

b)

Klientprogram:

```
package Oppgave2;

/**
 * Created by Martin on 29.04.2015.
 */

import java.util.*;

public class ClientBSTree {
    public static void main (String [] args){

        final int NODES = 1024;
        Random random = new Random();
        int heigths [] = new int [100];

        KjedetBinaerSokeTre<Integer> tree = new KjedetBinaerSokeTre<Integer>();

        for (int i = 0; i < 100; i++){

            tree = new KjedetBinaerSokeTre<Integer>();

            for (int j = 0; j < NODES; j++) {
                Integer element = new Integer(random.nextInt(30));
                tree.leggTil(element);
            }

            heigths [i] = tree.showHeight();


        }


        System.out.println("Tree with  " + NODES + " nodes.");

        tree.visInorden();
```

```
    /*********************************************/

    System.out.println();
    System.out.println("The minimal theoretical height of the tree is " +
    tree.findMinimalHeight(NODES));

    /*********************************************/

    System.out.println();
    System.out.println("The maximum theoretical height of the tree is " + (NODES -1));

    /*********************************************/

    System.out.println();
    System.out.println("Smallest height was " + tree.findSmallestHeight(heigths));

    /*********************************************/

    System.out.println();
    System.out.println("Largest height was " + tree.findLargestHeight(heigths));

    /*********************************************/

    System.out.println();
    System.out.println("Average height of all " + heigths.length + " trees was " +
    tree.findAverageHeight(heigths));




  }
}
```

Metoder lagd for klientprogram:

```
public int findMinimalHeight(int nodes){

  int result = (int)(((Math.log(nodes)/Math.log(2))+1)-1);

  return result;

}

public int findSmallestHeight(int heights []){
  int smallest =  heights[0];
```

```
    for (int elem : heights){
        if (smallest > elem){
            smallest = elem;
        }
    }

    return smallest;
}

public int findLargestHeight(int heights []){
    int largest =  heights[0];

    for (int elem : heights){
        if (largest < elem){
            largest = elem;
        }
    }

    return largest;
}

public int findAverageHeight(int heights []){
    int average = 0;

    for (int elem : heights){
        average += elem;
    }

    average /= heights.length;

    return average;
}
```

c)

Den gjennomsnittlige høyden man fant for n = 1024 I dette tilfellet var 52. Dette ga:

$$52 = \ c * \log_2 1024 \ \leftrightarrow \ \frac{52}{\log_2 1024} = c \ \leftrightarrow c = 5.2$$

Den teoretiske gjennomsnittlige høyden for n = 4096 ble da:

$$h = 5.2 \ * \ \log_2 4096 \ \leftrightarrow h = 62.4$$

Det man fikk i forhold på når man kjørte programmet for n = 4096 var at gjennomsnittshøyden var 167. Dette betyr enten at programmet oppfører seg annerledes i forhold til formelen, eller så kan det være at formelen er unøyaktig.

d)

```
@Test
public final void erElementIBSTre() {
```

```
   bs = new KjedetBinaerSokeTre<Integer>();
   bs.leggTil(e0);
   bs.leggTil(e2);
   bs.leggTil(e6);
   bs.leggTil(e3);
   bs.leggTil(e5);
   bs.leggTil(e1);
   bs.leggTil(e4);

   int values [] = {e0,e1,e2,e3,e4,e5,e6};
   for (int elem : values){
     assertTrue(bs.finn(elem) != null);
   }
   assertTrue(bs.finn(e7) == null);


/* Her legger du inn e0...e6 i treet i en vilk�rlig rekkef�lge.
 * Etterp� sjekker du om elementene fins og til slutt sjekker du
 * at e7 ikke fins
 */

}

/**
 * 1.  Tester ordning  ved � legge til elementer og fjerne minste
 */
@Test
public final void erBSTreOrdnet() {
   bs = new KjedetBinaerSokeTre<Integer>();
   bs.leggTil(e0);
   bs.leggTil(e2);
   bs.leggTil(e6);
   bs.leggTil(e3);
   bs.leggTil(e5);
   bs.leggTil(e1);
   bs.leggTil(e4);

Integer values[] = {e0, e1, e2, e3, e4, e5, e6};

for (int elem : values){
   bs.fjernMin();
   assertTrue(bs.finn(elem) == null);
}

 /* Her legge du f�rst inn e0...e6 i en vilk�rlig rekkef�lge
  * og s� fjerne du minste hele tiden
  */
}
```

Oppgave 3

a)

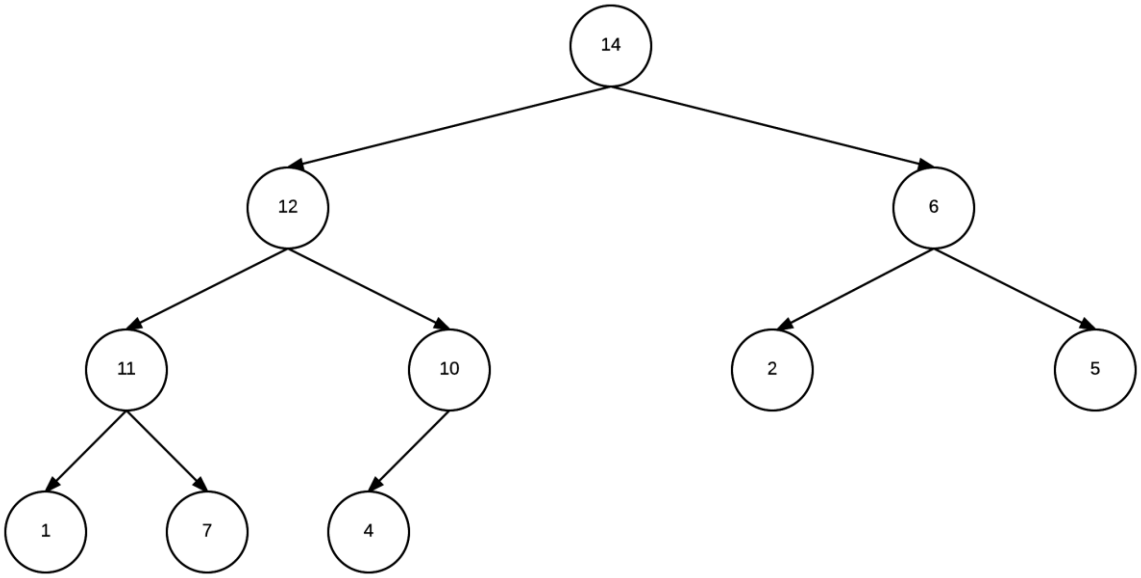
En heap er ett binært tre som er komplett, og i tillegg så er det for alle noder slik at både «left child» og «right child» er lik eller større noden over. Dette er også beskrivelsen for en minheap. Det motsatte av dette er en maxheap der «left child» og «right child» er lik eller mindre enn noden over.

b)

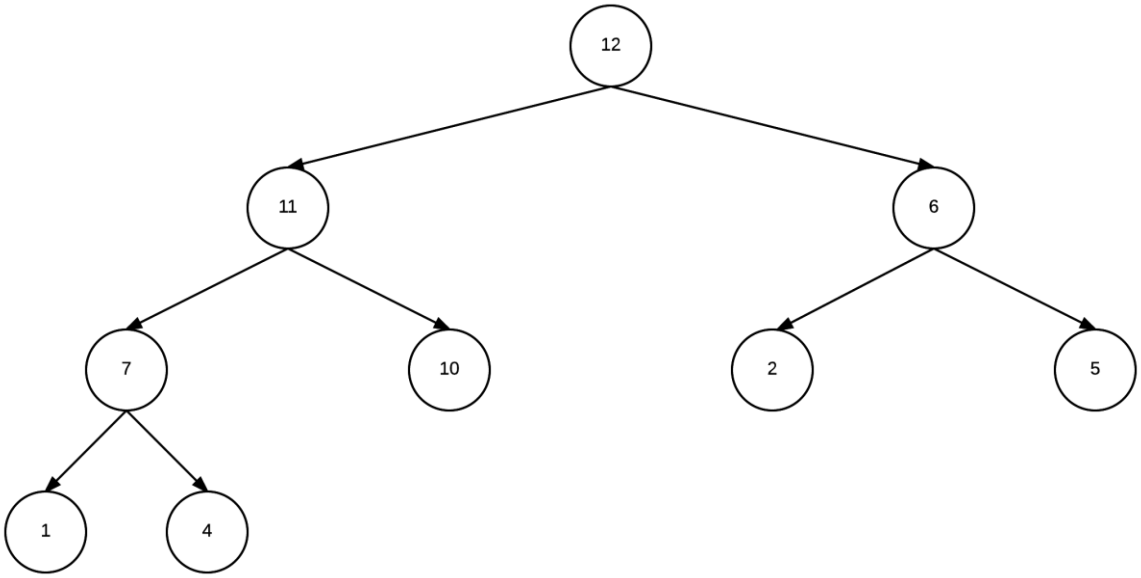
Både B og C er en «maxheap» siden alle «left children» og «right children» er mindre enn noden over. Viss man tegner tabellene over til en graf så ser man dette med engang. De er i tillegg komplette, med tanke på at nederste noder er venstrejustert, og det bare mangler noder på nederste dybde.

c)

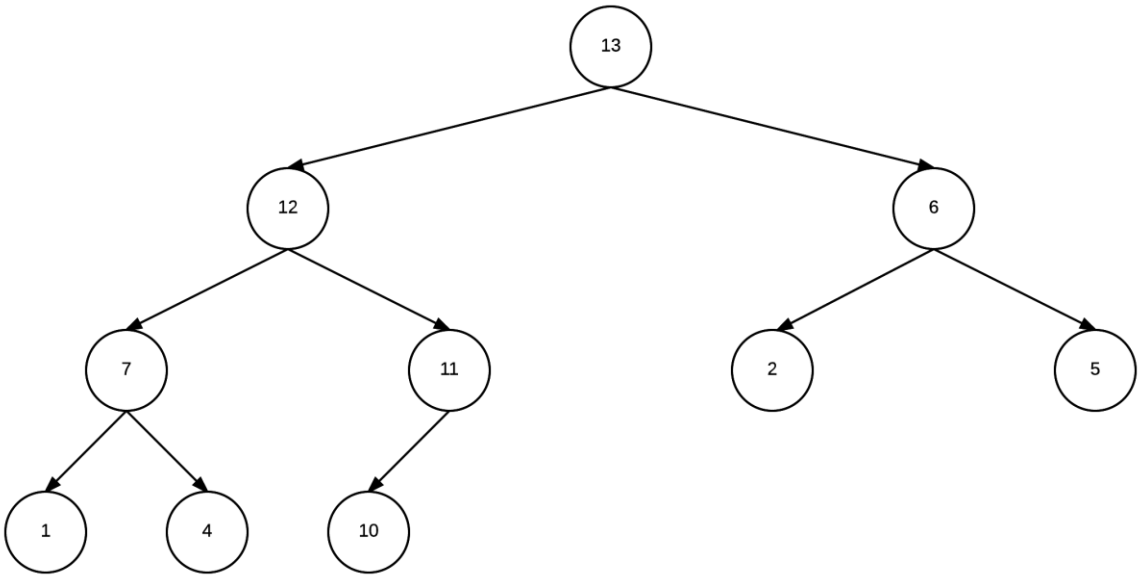
Treet tegnet fra tabellen:



Treet etter at 14 er fjernet og reorganisert:



Treet etter at 13 har blitt puttet inn og reorganisert:



d)

Prinsippet i pkt. C kan bruke til å sortere elementer slik at man får minste eller største verdi først og synkende verdier under den. Man kan sørge for at uansett verdi får en spesifikk plassering i arrayen som man måtte bruke for å illustrere et binært tre på java.

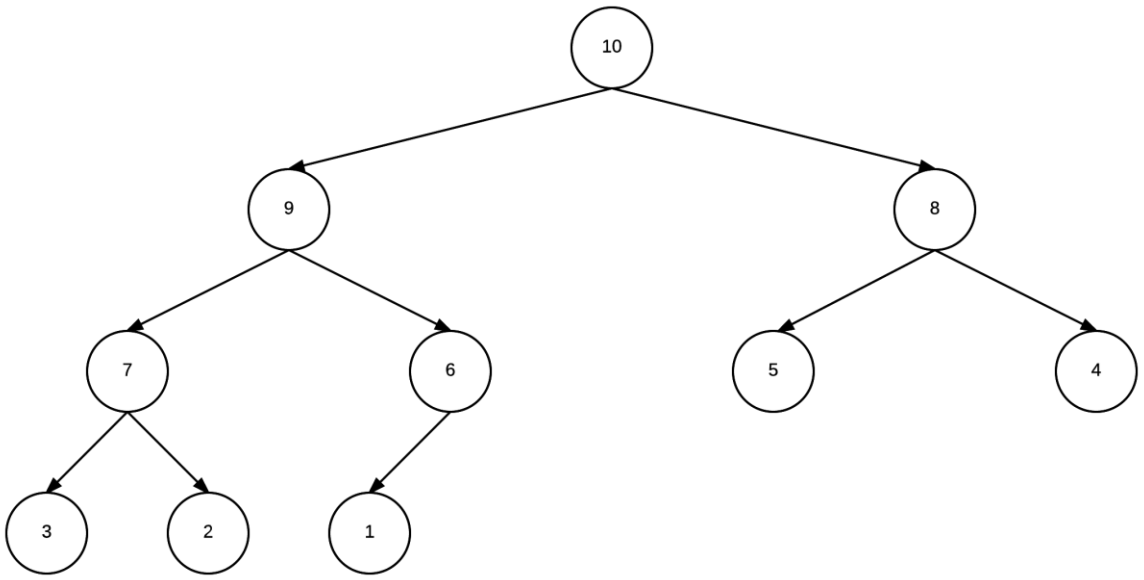
e) Antall noder i et fullt binært tre med nivå 0..K er $2^0 + 2^1 + \cdots + 2^K = 2^{K+1} - 1$

Høyden til et komplett binært tre med nivåer 0...K er h=K. Setter dette inn i utrykket og får et nytt ut trykk for høyden H: $n = 2^{K+1} - 1 = 2^{H+1} - 1$

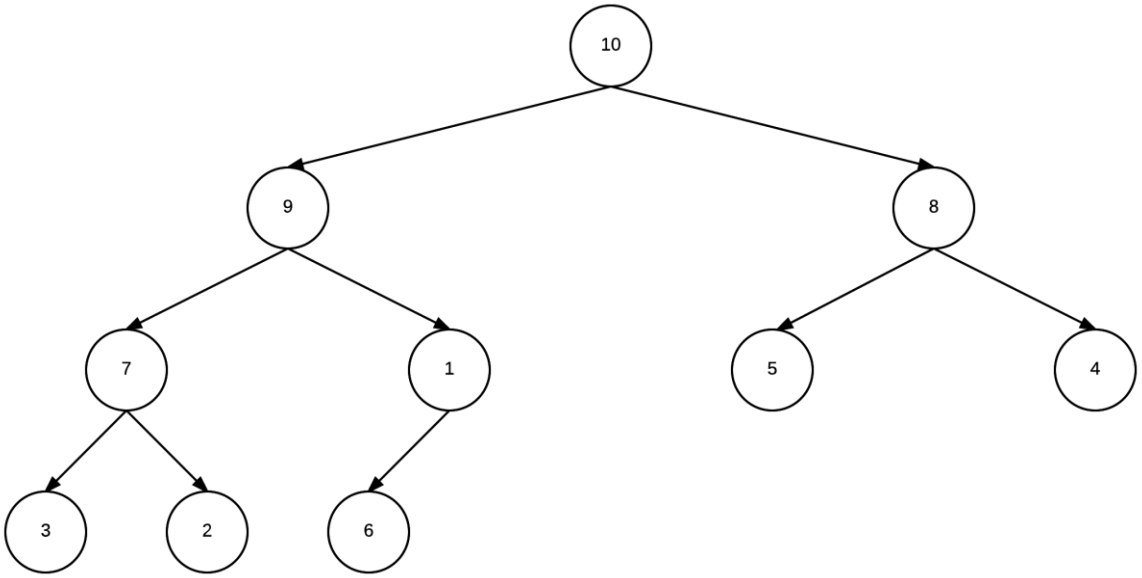
$$2^{K+1} = 2^{H+1} \rightarrow H + 1 = \log_2(n+1) \rightarrow H = \log_2 n$$

Det er i alt (n/2)-1 interne noder som skal sammenlignes. Antall sammenligniner blir da $\log_2 n * \frac{n}{2}$ som er O(n*log*n).

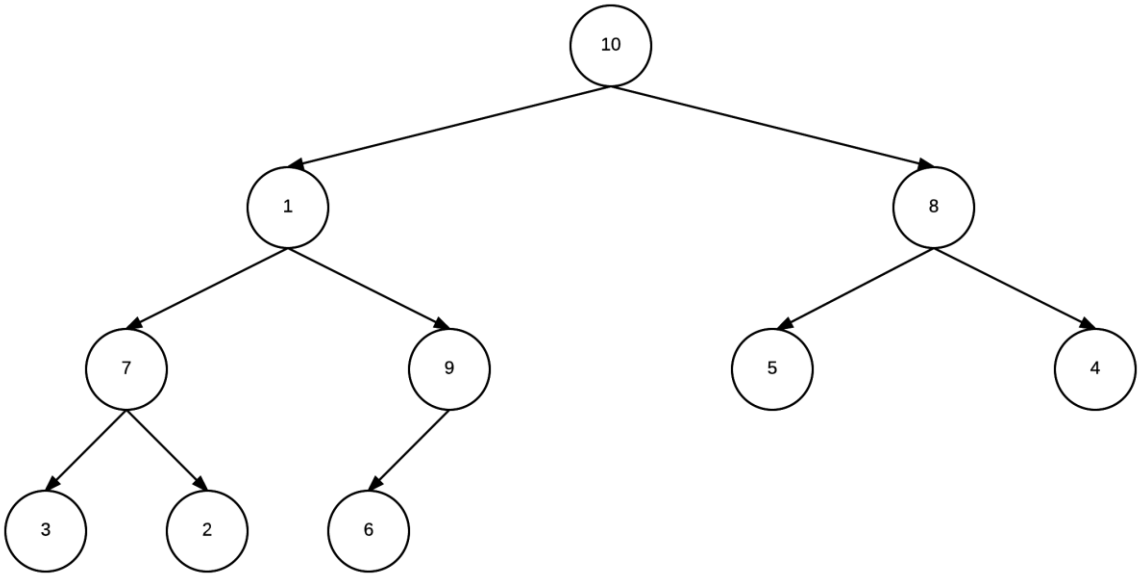
f)



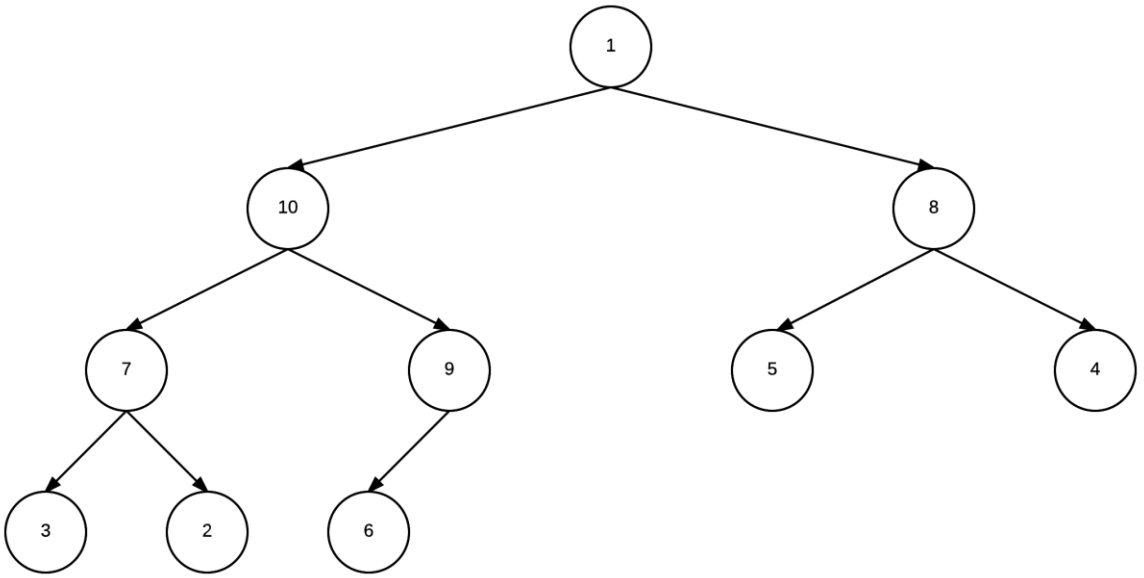
Begynner sortering:



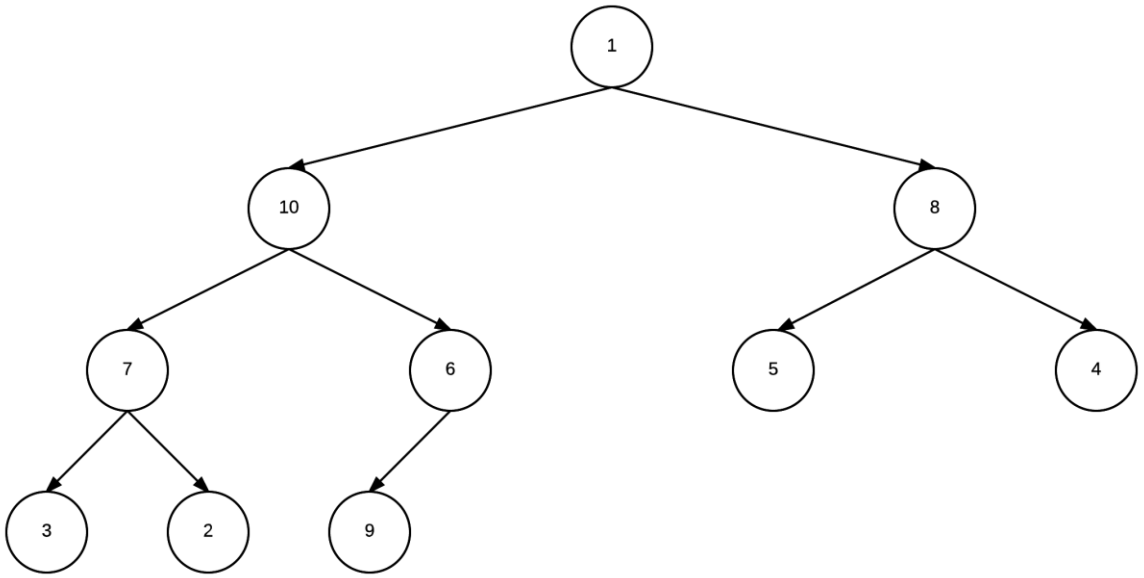
Steg 2:



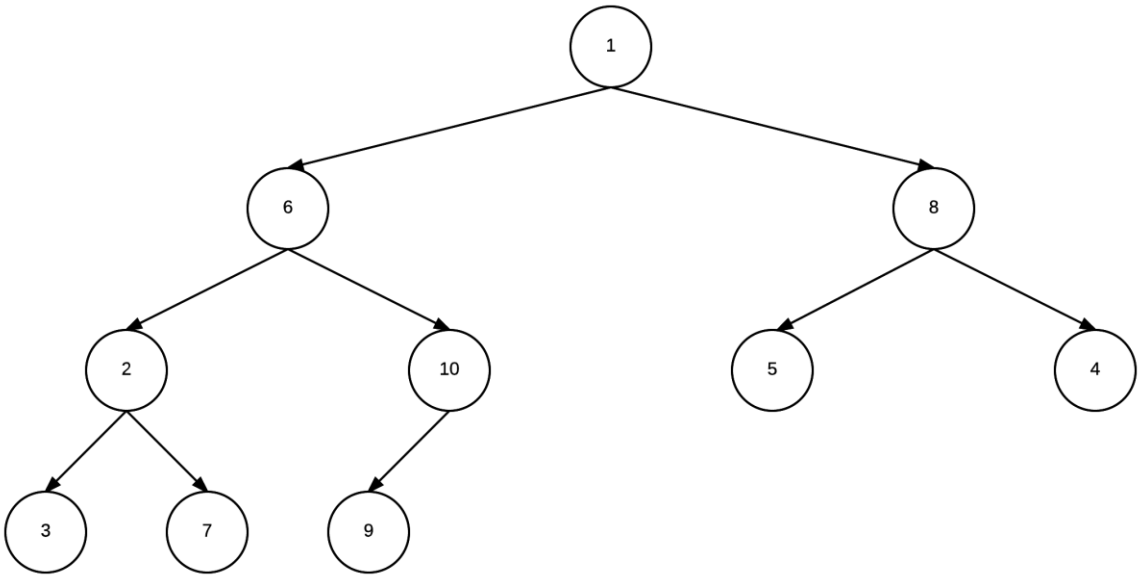
Steg 3:



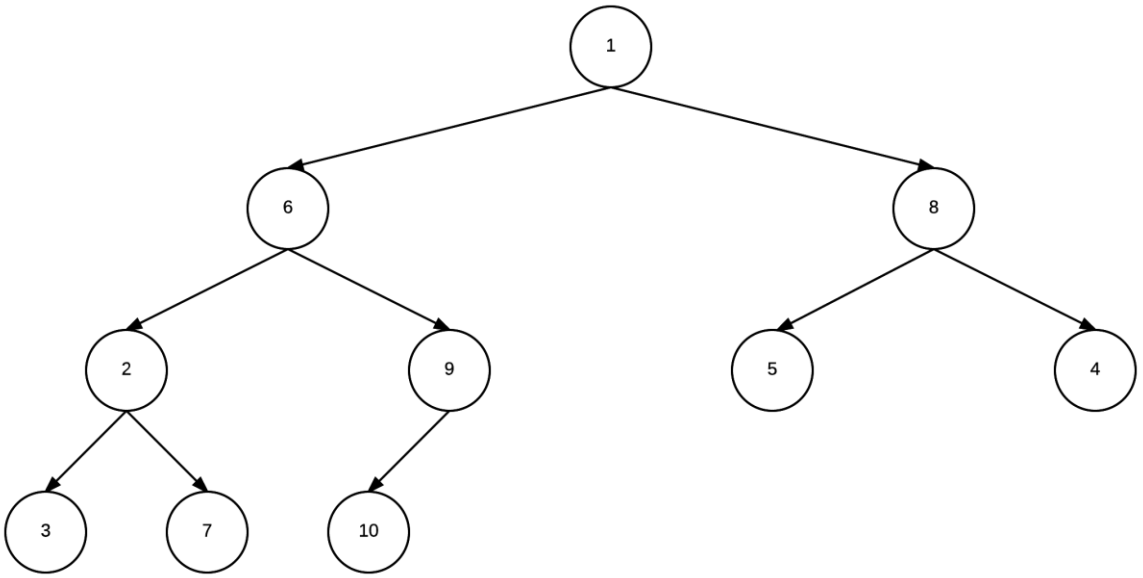
Steg 4:



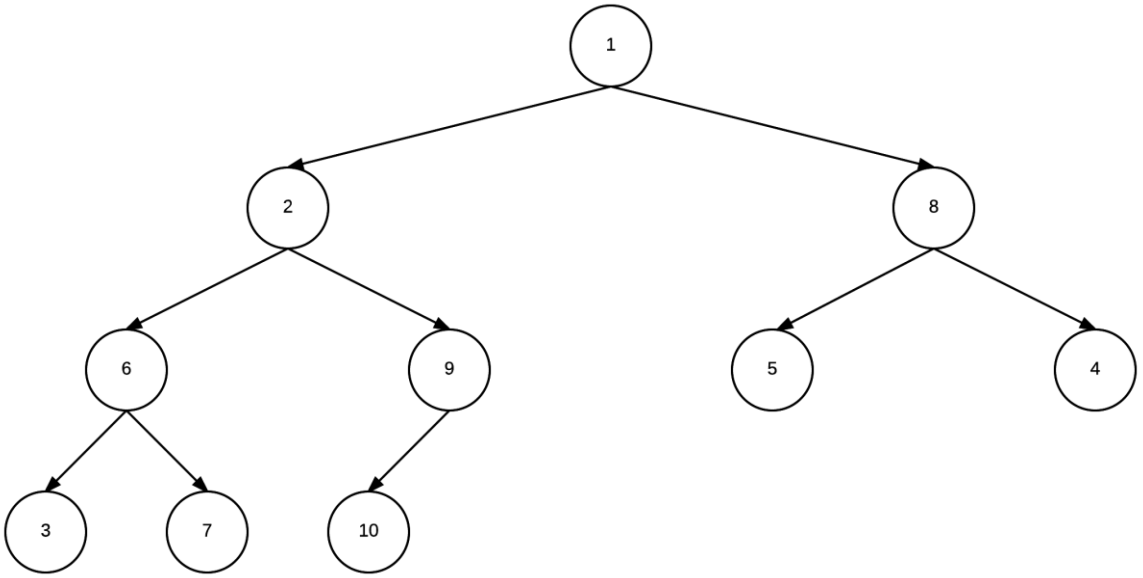
Steg 5:



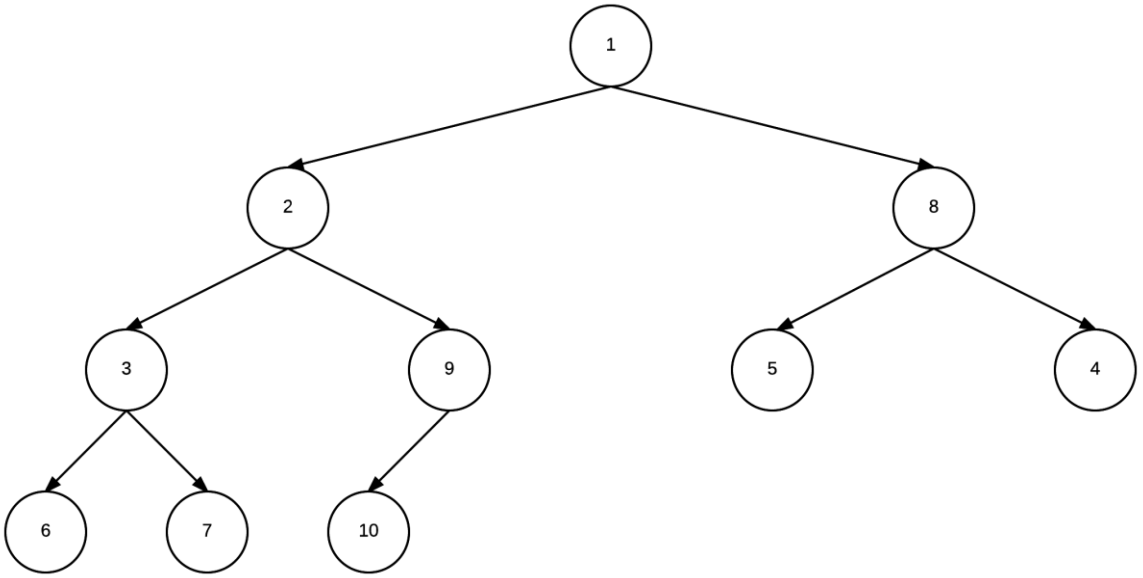
Steg 6:



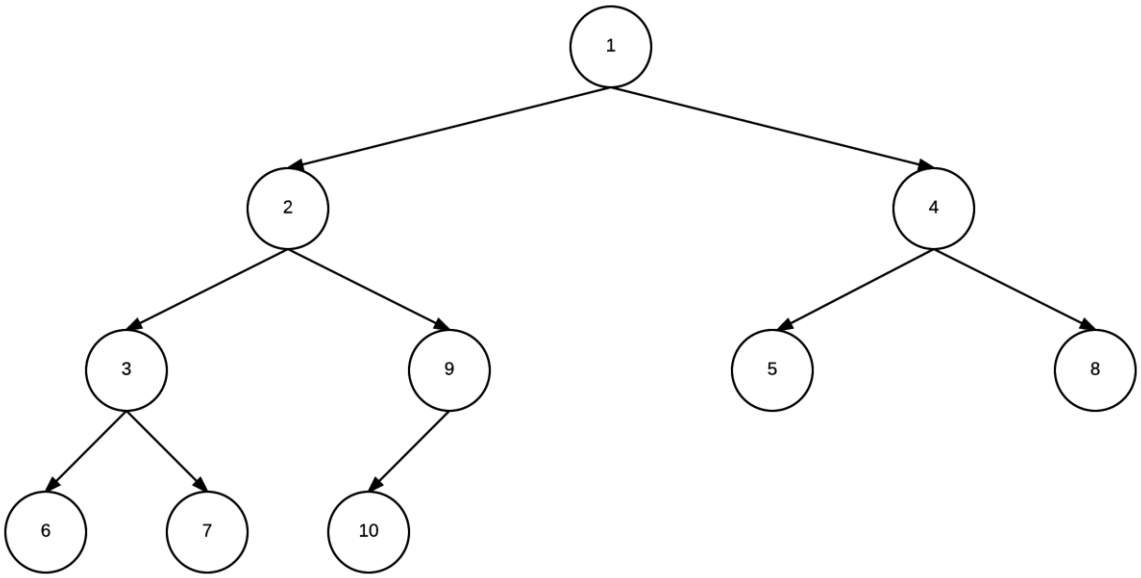
Steg 7:



Steg 8:



Steg 9:



g)

reparerOpp klassen

```
private void reparerOpp() {
   int aktuell = antall - 1;
   T help;
   while (aktuell != 0 && data[aktuell / 2].compareTo(data[aktuell]) < 0) {
      help = data[aktuell];
      data[aktuell] = data[aktuell / 2];
      data[aktuell / 2] = help;
      aktuell /= 2;
   }
}
```

Oppgave 4

a)

Ett 2-3 tre operer likt et binært søk tre i og med at venstre undertre inneholder element som er mindre enn noden og høyre undertre inneholder element som er større eller lik noden. Det er også bestemte krav til mengden undertre til 2-noder og 3-noder. Ett 2-3 er nyttig med at det reduserer nivåene man trenger for å lagre informasjon i ett tre, og vil derav være nyttig i å redusere ressursbruk.

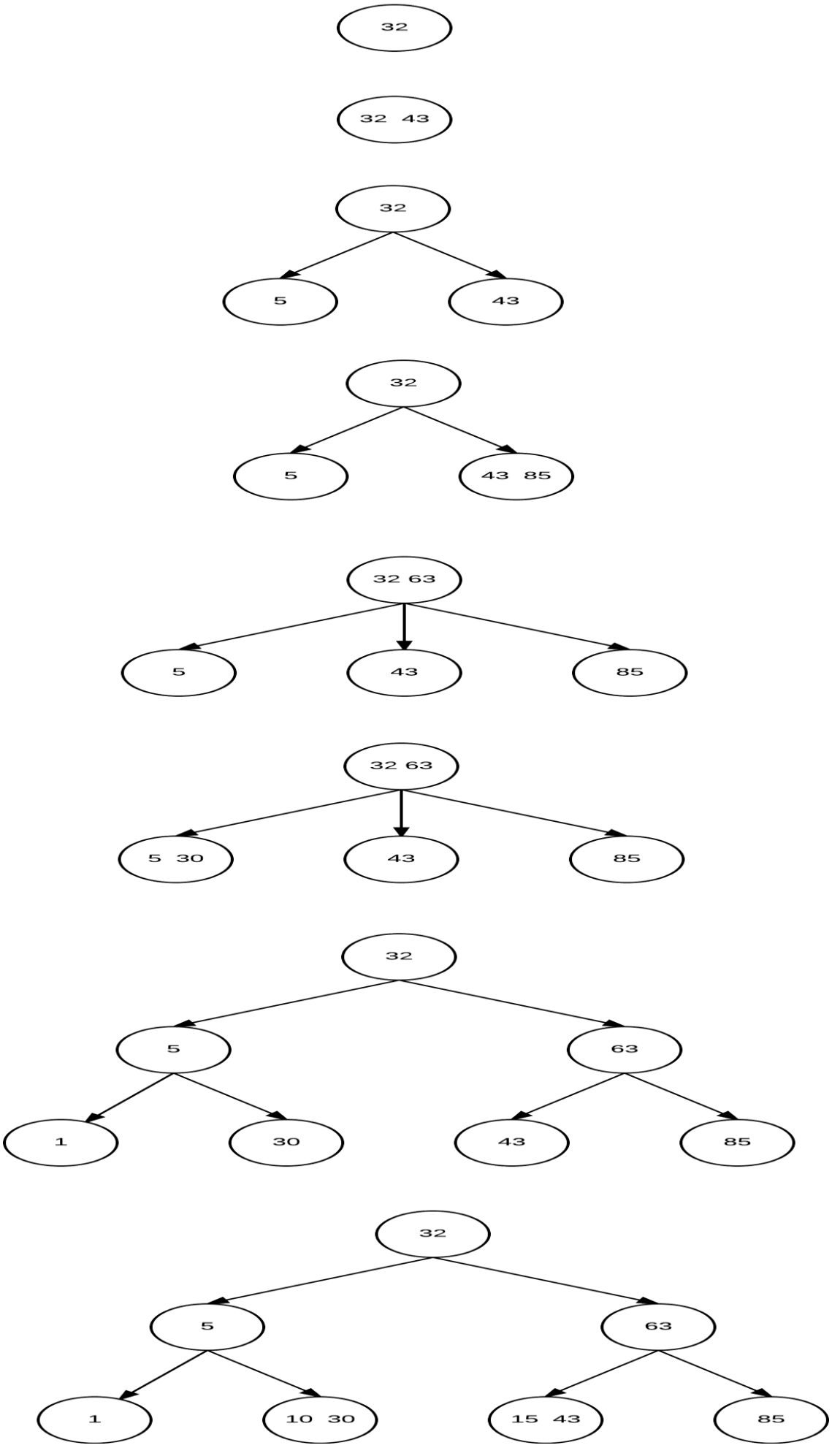
b)

En 2-node er en node som har null eller to undertre og inneholder bare ett element. Undertrærne av 2-noden funker på samme måte som i et binært søke tre.

En 3-node har null eller tre undertre og inneholder to elementer der det ene elementet er designert til å ha et mindre element og et element som er større. Hvis en 3-node har undertre så vil venstre undertre inneholde mindre element enn det minste i 3-noden, og høyre element vil inneholde større eller like stort som største element i 3-noden. Det midterste undertreet vil inneholde element som er større enn eller lik minste element i 3-noden og mindre enn største element i 3-noden.

c)

Starter i roten (45), går til venstre siden 42 er mindre. Går så til høyre siden 42 er større enn 30. Vi kommer til en 3-node uten barn, og den største verdien i noden er mindre enn 42. Dermed finnes ikke 42 i dette treet.


32
32 43
32
5
43
32
5
43 85
32 63
5
43
85
32 63
5 30
43
85
32
5
63
1
30
43
85
32
5
63
1
10 30
15 43
85

Oppgave 5

a)

Ved bruk av hashing så lagrer man gjerne et element ved bruk av en nøkkel i en hash table, og ved bruk av en funksjon så kan man igjen hente ut dette elementet ved bruk av nøkkelen. En hashing funksjon er en funksjon som distribuerer elementer utover en tabell, og denne funksjonen må kunne gjøre dette effektivt slik at man unngår kollisjoner, eller det at man overskriver elementer som ligger i tabellen fra før av. Clustering går ut på at man ikke har en effektiv hashing funksjon, og derifra får kollisjoner som reduserer effektiviteten av programmet kraftig hvis det går riktig galt. Og der kommer inn den perfekte hashing funksjonen. En hashing funksjon som kan plassere alle elementer til en unik posisjon regnes som perfekt. Dette fjerner risiskoen av kollisjoner og clustering.

b)

0	VV50000
1	SU65431
2	ZX87181
3	
4	TA14374
5	UV14544
6	SU32944
7	ST47007
8	
9	

c) “ab” og “123”:

For hånd:

“ab”:

$97 * 31^{(2-1)} + 98 * 31^{(2-2)} = 3105$

“123”:

$49 * 31^{(3-1)} + 50 * 31^{(3-2)} + 51 * 31^{3-3} = 48690$

Java-Program:

```
public static void main(String[] args)  {
        System.out.println("ab".hashCode());
        System.out.println("123".hashCode());
}
```

Utskrift:

```
3105
48690
```

d) De gir ikke samme utskrift siden de er forskjellige objekter, selv om variablene har samme verdi. F or at de skal ha samme hashCode(), må vi sette b = a. Da vil de peke til samme objekt og dermed ha s amme hash.

Test:

```
public static void main(String[] args) {
   Student a = new Student(10, "Ole");
   Student b = new Student(10, "Ole");
   System.out.println(a.hashCode());
   System.out.println(b.hashCode());
   b = a;
   System.out.println(b.hashCode());
}
```

Utskrift:

```
1836019240
325040804
1836019240
```

c)

Sammenlikning av HashSet og Integer-tabell:

```
package Oppgave5.e;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Random;

/**
 * Created by Martin on 11.05.2015.
 */
public class Main {
   public static void main(String[] args) {
      final int ANT = 100000;
      final int GRENSE = 999999;
      Random t = new Random();
      Integer[] tab = new Integer[ANT];
      HashSet<Integer> hs = new HashSet<Integer>();
      int tall = 376;
      for (int i = 0; i < ANT; i++) {
         tab[i] = tall;
         hs.add(tall);
         tall = (tall + 45713) % 1000000;
      }
      Arrays.sort(tab);
```

```
        //Genererer tilfeldige tall
        Integer[] randomNumbers = new Integer[10000];
        for (int i = 0; i < 10000; i++) {
            randomNumbers[i] = t.nextInt(GRENSE);
        }

        //Søker først gjennom 'hs', HashSet
        int hashTeller = 0;
        long hashTidStart = System.currentTimeMillis();
        for (int i = 0; i < 10000; i++) {
            if (hs.contains(randomNumbers[i])) {
                hashTeller++;
            }
        }
        long hashTid = System.currentTimeMillis() - hashTidStart;

        //Søker så gjennom Integer tabellen
        int intTeller = 0;
        long intTidStart = System.currentTimeMillis();
        for (int i = 0; i < 10000; i++) {
            if (Arrays.binarySearch(tab, randomNumbers[i]) >= 0) {
                intTeller++;
            }
        }

        long intTid = System.currentTimeMillis() - hashTidStart;
        System.out.println("Fant " + hashTeller + " i HashSet etter " + hashTid + "ms.\n" + "Fant " +
intTeller + " i Int-tabell etter " + intTid + "ms.");
    }
}
```

Utskrift:

```
Fant 977 i HashSet etter 1ms.
Fant 977 i Int-tabell etter 10ms.
```

Oppgave 6

a)

Kø	Besøkt
abcdef	c
abdef	ce
abdf	cef
abd	cefb
ad	cefbd
a	cefbda
tom	cefdba

b)

Kø	Besøkt
abcdef	c
abdef	cf
abde	cfb
ade	cfbe
ad	cfbed
a	cfbeda
tom	cfbeda

c)

V = {a,b,c,d,e,f}

E = {ad,db,de,eb,bf,ef,ec,cf}

Nabomatrise:

	a	b	c	d	e	f
a	F	F	F	T	F	F
b	F	F	F	T	T	F
c	F	F	T	F	T	T
d	T	T	F	F	T	F
e	F	T	T	T	F	T
f	F	T	T	F	T	T

Naboliste:

a	Nabo til	d
b	Nabo til	d,e,f
c	Nabo til	e,f
d	Nabo til	a,b,e
e	Nabo til	b,c,d,f
f	Nabo til	b,c,e