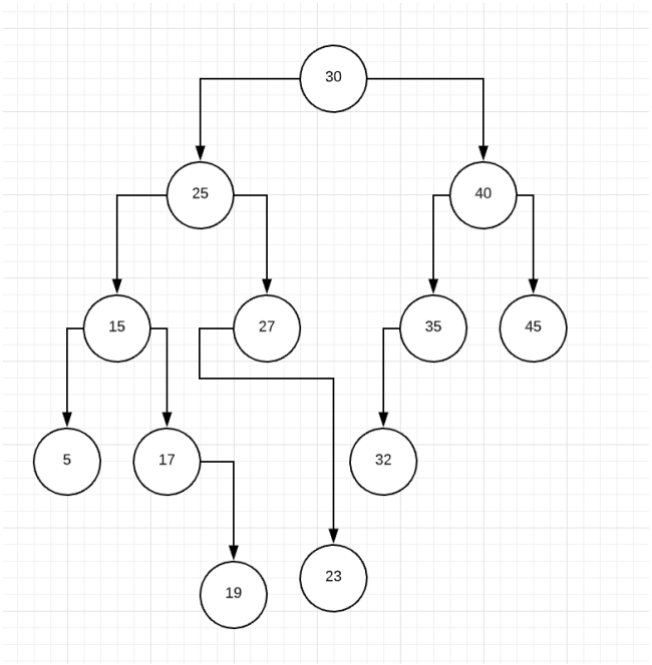
Skoleprøve – DAT102 – Joachim Leiros

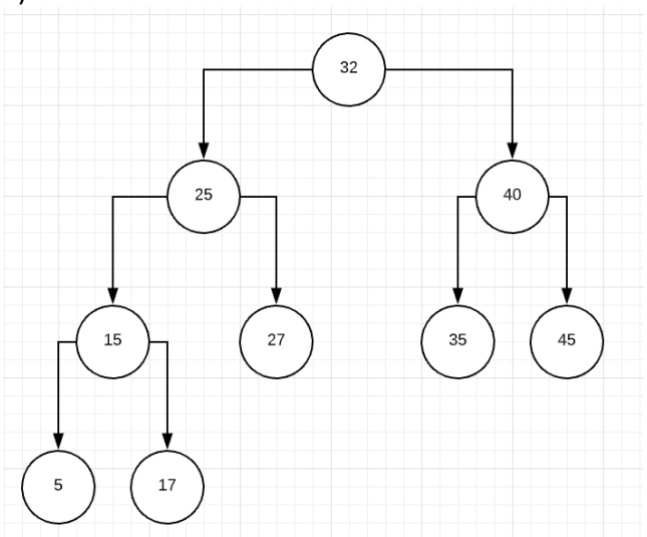
Oppgave 1)
1 – 5n^2 + 3n + 10 = O(n^2)
2 – 10nlog2n + 5n = O(log2n)
3 – 15n + 3 = O(n)

Oppgave 2)
a)

i)



ii)



Erstatter 30 med minste i høyre undertre, i dette tilfellet 32.

b)

```
private void visRekPostorden(BinaerTreNode<T> p) {
   if (p == null) {
     return;
   }

   postOrder(p.left);
   postOrder(p.right);
   System.out.printf("%s ", p.data);
 }
```

c)

Lag den rekursive metoden finnRek (se vedlegg) som returnerer en referanse til det spesifiserte elementet hvis det fins i dette bs-treet, ellers returnerer metoden null-referansen.

d)

```
private boolean erIdentiskRek(BinaerTreNode<T> t1, BinaerTreNode<T> t2) {
        if( t1 == null && t2 == null) {
                return true;
         } else if (p == null || q == null) {
                return false;
         }

        boolean venstre = BinaerTreNode(t1.venstre, t2.venstre);
        boolean hoyre = BinaerTreNode(t1.hoyre, t2.hoyre);
        return venstre.val == hoyre.val && venstre && hoyre;
}
```

Oppgave 3)
a)

i)

4	1	7	3	2
1	4	7	3	2
1	3	4	7	2
1	2	3	4	7

ii)

```
public static <T extends Comparable<T>> void insertionSort(T[] data) {
   for (int i = 1; i < data.length; i++) {
      T key = data[i];
      int position = i;

      while (position > 0 && data[position-1].compareTo(key) > 0) {
         data[position] = data[position-1];
         position--;
      }
      data[position] = key;
   }
}
```

iii) Antall sammenlikninger ved sortering ved insetting er i verste tilfelle n(n-1)/2 ettersom i ett verstefall vi alltid må sette inn det nye elementet først. Sammenlikninger blir da 1 + 2 + 3 + 4 + n-1 = n(n-1)/2

iv)

Tidskompleksiteten for insertion sort er O = n^2

b)

i)

```
private static <T extends Comparable<T>> void quickSort(T[] data, int min, int
max) {
   if (min < max) {
      int indexOfPartition = partition(data, min, max);
      quickSort(data, min, indexOfPartition-1);
      quickSort(data, indexOfPartition+1, max);
   }
}
```

ii)

```
public static <T extends Comparable<T>> void quickSort(T[] data, int min, int
max) {
   if (min < max) {
      //Koden lager en partition
      int indexOfPartition = partition(data, min, max);
      //Koden sorterer venstre siden av partition, de lavere verdiene.
      quickSort(data, min, indexOfPartition-1);
      //Koden sorterer den høyre siden av partition, de høyere verdiene.
      quickSort(data, indexOfPartition+1, max);
   }
}
```

iii)

Gjennomsnitts tidskompleksitet for kvikksortering er O(nlogn)

iv)

Kvikksortering fungerer dårlig i situasjoner hvor dataen allerede er sortert eller nesten sortert. Evt. I situasjoner hvor mengden data er veldig stor.

Oppgave 4)

a)

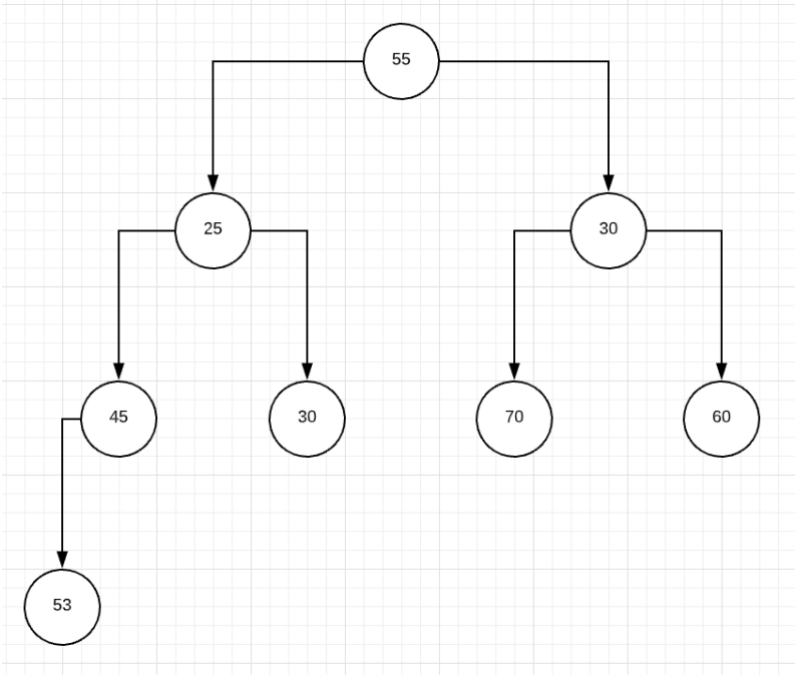
i) Venstrebarn vil vi finne i tre[1] siden tabell[2i+1] = 1.
ii) Høyrebarn vil vi finne i tre[2] siden tabell[2i+2] = 2
iii) Referansen til forelderen finner vi i tre[0] siden tabell[(i-1/2)].
iv) Minimumshøyden på et binært tre som har n noder vil være ca. log2(n)+1.

b)

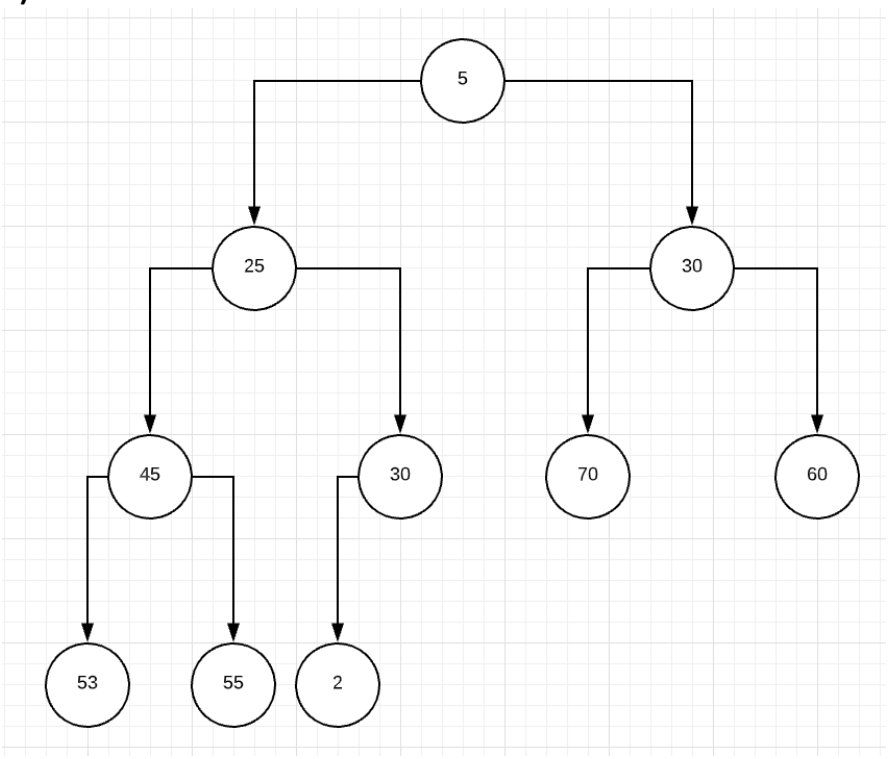
En minimumshaug er en haug hvor child-elementene er er lik eller større enn parent elementet. Div i haugen vist i pkt. A så er 25 og 20 større enn 5.

c)

For å fjerne en haug fjerner vi det minste elementet i haugen. Det finner vi i roten. Når vi har fjernet rotelementet så erstatter vi det med et annet erstatningselement fra treet. Erstatningselementet er det siste bladet på høyden og settes inn som ny rot.



d)



e)

fjernMinste() – Hvis antallet er større en null så setter vi haugen som svar. Vi fjerner det minste elementet i haugen, som vil være roten. (data[0]) Så kaller vi metoden reparerNed for å ordne plasseringene til elementene i haugen etter fjerning.

reparerNed() – Er en metode som blir brukt dersom minste element/rot blir f jernet fra haug. Den reparerer treet slik at vi får en haug etter å ha fjernet elementet.

f)

En prioritetskø er en samling elementer hvor de blir sortert etter prioritet. Elementer med høyest prioritet blir tatt først ut, dersom to elementer har samme prioritering gjelder first-in first-out prinsippet hvor elemtentet som først ble satt inn får prioritering over elementet som kom senere.

g)

i)

@Before annotasjonen sier at metoden skal utføres før en test.
@Test annotasjonen sier at metoden annotasjonen hører til kan kjøres som en test.

ii)

Oppgave 5)

a)

```
public boolean inneholder(T element) {
        LinearNode a = foerste;
        while (a != null && !element.equals(a.getElement())) {
        a = a.getNeste();
        } return (a != null);
}
```

b)