SkulePrøve 2022

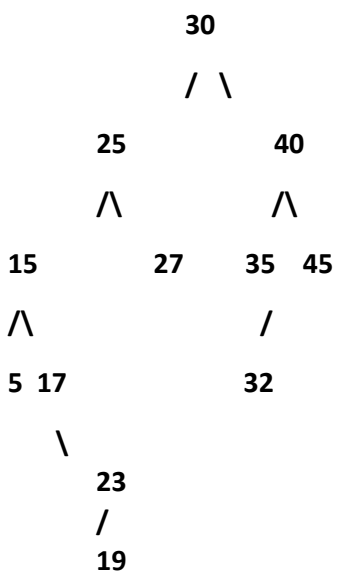
Oppgåve 1

a) O(n^2)
b) $O(n*\log_2(n))$
c) O(n)

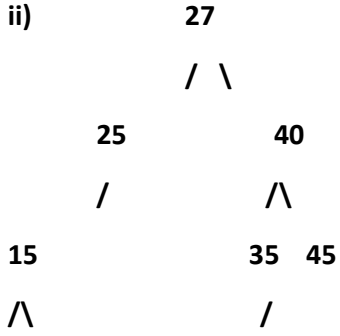
Oppgåve 2

A)

I)



ii)



5 17 32

b)

```
private void visRekPostorden(BinaerTreNode<T> p) {

    if (p== null)
        return;

    visRekPostorden(p.getVenstre());

    visRekPostorden(p.getHoyre());

    System.out.println(p.getElement());

}
```

c)

```
private T finnRek(T element, BinaerTreNode<T> p) {

    p = rot;
    T resultat = null;

    if (p!=null) {
        T rotelement =p.getElement();

        if (element.equals(rotelement))
            resultat = rotelement;
        else if (element.compareTo(rotelement)<0) {
            resultat = finnRek(element,p.getVenstre());
        } else {
            resultat = finnRek(element, p.getHoyre());
        }

    }
    return resultat;
```

d)

```java
private boolean erIdentiskRek(BinaerTreNode<T> t1,  BinaerTreNode<T> t2){

    if (!t1.equals(t2)) {
        return false;
    }
    boolean erIdentiskVenstre = false;
    if (t1.getVenstre()==null||t2.getVenstre()==null) {
        erIdentiskVenstre = t1.getVenstre()==null&&t2.getVenstre()==null;
    } else {

    erIdentiskVenstre = erIdentiskRek(t1.getVenstre(),t2.getVenstre());}

    boolean erIdentiskHøgre = false;
       if(t1.getHoyre()==null||t2.getHoyre()==null) {
           erIdentiskHøgre = t1.getHoyre()==null&&t2.getHoyre()==null;
       } else {
           erIdentiskHøgre = erIdentiskRek(t1.getHoyre(),t2.getHoyre());
       }
       return (erIdentiskVenstre&&erIdentiskHøgre);

    }
```

Oppgåve 3

i)

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

ii)

```java
public static <T extends Comparable<T>> void InsertionSort(T[] data) {

    for (int i = 0; i<data.length; i++) {

        T temp = data[i];

        int o = i-1;
        boolean avsluttet = false;

        while(o>=0&&!avsluttet) {
            if (temp.compareTo(data[o])<0) {
                data[o+1] = data[o];
                o--;
            }

            else {
                avsluttet = true;
            }
        }
        data[o+1] = temp;
    }

}
```

iii)

i verste tilfellet er effektivitetn av insertionsort lik $O(n^2)$ metoden har en indre og ytre løkke, den ytre løkka vil uansett gå n ganger da den starter på 0 og ender på n-1. Den indre løkka vil i værste fall gå n-1 ganger siden kvart element må sammenlignes med alle elementene foran den i tabellen. Da blir antall CompareTo kall lik n*(n-1)/2 og bigO = $O(n^2)$.

iv)

bigO = O(n^2) begrunnelse i forrige oppgave.

B)

i)

```
public static <T extends Comparable<T>> void quickSort(T[] data, int min, int maks) {

    if (min<maks) {
        int pi = partition(data, min, maks);
        quickSort(data, min, pi-1);
        quickSort(data,pi+1,maks);
    } else {
        return;
    }

}
```

ii)

partition metoden deler tabellen i to deler, den gjør det ved å velge et element i tabellen som vi kaller for pivot dermed deler den tabellen i to deler slik at elementene i tabellen til venstre for pivot elementet er mindre eller like stort som pivot elementet og slik at elementene til høgre for pivot elementet er større.

iii)

gjennomsnittstidskompleksitet = O(n*logn).

iv)

valget av pivot er veldig viktig for effektiviteten av quicksort. I verste tilfellet velges pivot elementet slik at den ene halvdelen av tabellen er tom. Det gjør tidskompleksiteten om til O(n^2)

Oppgave 4

a)

i)

man finner venstrebarn i tre[i*2].

ii)

man finner høgrebarn i tre[(i*2)+1];

iii)

man finner forelder i tre[i/2].

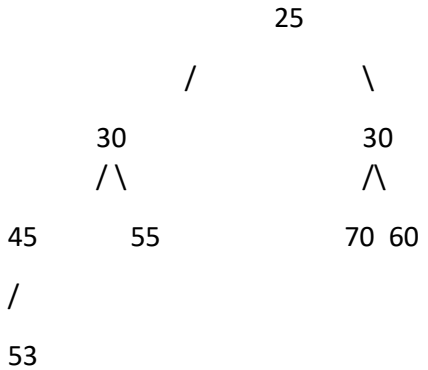
iv)

minimumshøgden til et tre med n noder er floor($\log_2(n)$)

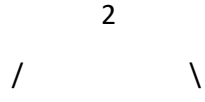
b)

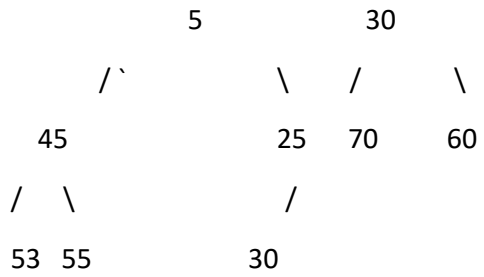
en minimumshaug er et binært tre der foreldernoden alltid er mindre enn barna sine.

C)



D)





d) reparerNed er en metode som gjør en nestenhaug om til en min haug.
Metoden starter i roten og finner den minste av de to barna.
Dermed sjekker den om det minste barnet er større enn forelderen.
Viss det minste barnet ikkje er større enn forelderen bytter det minste barnet og forelderen plass. Og den sjekker igjen om den minste av de nye barna til foreldernoden som nettop bytta plass er mindre enn begge barna helt til den er på rett plass.

fjernMinste() metoden fjernerdet minste elementet i minimumshaugen som skal være roten med det siste elementet som ble lagt til i haugen, dermed bruker metoden reparerned metoden for å sørge for at alle nodene havner på korrekt plass.

f)
en prioritetskø er en tabell som sorterer elementene basert på prioriteten til elementet, hvordan vi prioriterer elementene kan vi definere i objektets compareTo metode

g)

i)
@BeforeEach sier at denne metoden skal utføres før hver test.
@Test sier ifra til programmet at dette er en test.

ii)

```java
class priKoeTest {

    private PriKoeADT<Integer> koe;

    @BeforeEach
    public void reset() {
        koe = new PriKoe<Integer>();
    }

    @Test
    void setteInnOgFjerne() {
        koe.settInn(5);
        koe.settInn(7);
        koe.settInn(6);
        koe.settInn(1);
        assertEquals(koe.fjern(), 1);
        assertEquals(koe.fjern(), 5);
        assertEquals(koe.fjern(),6);
        assertEquals(koe.fjern(),7);
    }

}
```

Oppgave 5

a)

```java
//Andre metoder
public boolean inneholder(T element) {

    LinearNode<T> temp = første;
    boolean finnes = false;
    int i = 0;
    while(i<antall&&!finnes) {
    if (temp.getElement().equals(element)) {
        finnes = true;
    } else {
        i++;
    }
    }
    return finnes;

}
```

b)

```java
public void settInn(T element){

    if (element.compareTo(første.getElement())<0) {
        LinearNode<T> nyNode = new LinearNode<T>(element);
        nyNode.setNeste(første);
        første = nyNode;
        antall++;
    } else {

    LinearNode<T> temp = første.getNeste();
    LinearNode<T> forrige = første;
    LinearNode<T> nyNode = new LinearNode<T>(element);
    boolean funnet = true;
    int i = 0;
    while (i<antall&&!funnet) {
        if (element.compareTo(temp.getElement())<=0) {
            forrige.setNeste(nyNode);
            nyNode.setNeste(temp);
            antall++;
            funnet = true;
        } else {
            i++;
        }
    }
    if (!funnet) {
        temp.setNeste(nyNode);
        antall++;
    }

    }
}
```