Obligatorisk innlevering 5 – dat102

Gruppe: 19
Marius Larsen Arnevik
Julie Elea Fjellstad

Oppgaver fra uke 15

Oppgave 2.

a) Metoden skrivVerdierRek er ikke optimal fordi den går ned i venstre og høyre gren uansett. I tillegg vil alt i venstre side være mindre dersom vi finner et element i en av nodene som er mindre enn min, for venstre side er alltid minst. Samme logikk for større enn maks og høyre side. Det blir gjort flere unødvendige kall.

b)

```
    public void skrivVerdier(T nedre, T ovre) {
        skrivVerdierRek(rot, nedre, ovre);
    }

    private void skrivVerdierRek(Node<T> node, T min, T maks) {
        if (node != null) {
            int minst = node.data.compareTo(min);
            int storst = node.data.compareTo(maks);

            if (minst > 0) {
                skrivVerdierRek(node.left, min, maks);
            }

            if (minst >= 0 && storst <= 0) {
                System.out.print(node.data + " ");
            }

            if (storst < 0) {
                skrivVerdierRek(node.right, min, maks);
            }
        }
    }
}
```

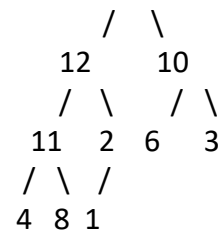
Oppgave 5.

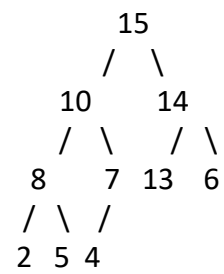
a)

> En haug er et komplett binært tre med noder som inneholder Comparable objekter. I tillegg vil foreldrenoden være mindre eller lik barna i en minimums-haug, og i en maksimumshaug vil foreldrenoden være større eller lik (s.690).

b)

Både tabell b og c danner en makshaug. Dette begrunnes med at foreldrenoden er større eller lik barna. Trærne ser slik ut

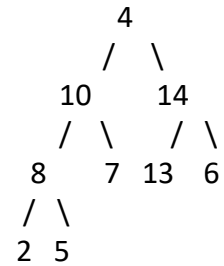




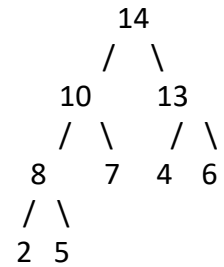
Her ser man tydelig at foreldrenoden er større eller lik barna.

c)

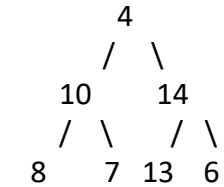
Når vi først fjerner elementet med verdi 15 vil treet se slik ut:

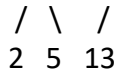


Ser så på hvilket barn som er størst, her har vi 14 og dermed bytter vi 4 og 14. Videre ser vi på barna under 4 igjen, det største er 13 og vi bytter igjen 4 og 13. Treet vil se slik ut:

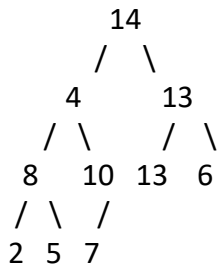


Når vi setter inn 13 vil treet se slik ut:





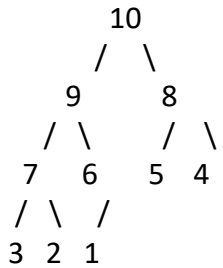
Vi bruker samme metode for å sortere og ser på foreldrenoden. 13 er større enn 7, vi bytter de, 13 er større enn 10 og vi bytter de, og 13 er større enn 4 så vi bytter de. Til høyre vil vi også måtte bytte 13 og 14. Etter at vi har sortert det ser det slik ut:



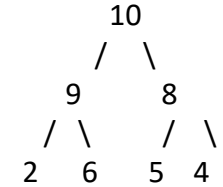
d)

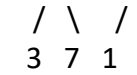
Fjerner rotverdien og bytter den ut med siste elementet, så heapify ned fra roten og gjenta helt til alle elementene er fjernet fra haugen og plassert riktig.

e)

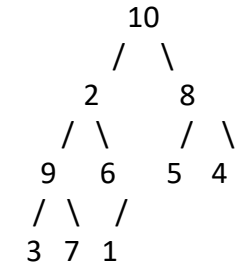


Starter da i nederste node som her er 3 og sammenligner med forelderen. 7 har to barn, 3 og 2, 2 er minst, bytter 7 og 2.

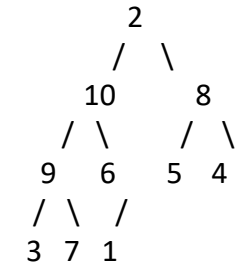




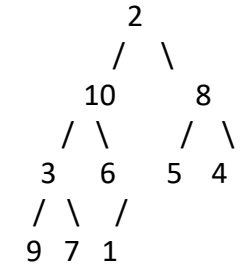
Sammenligner igjen med forelderen. 2 er mindre enn 9, 2 er mindre enn 6, bytter 9 og 2.



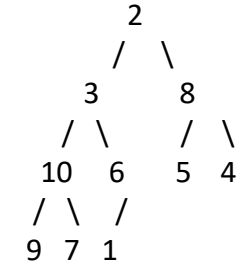
Bytter 2 og 10

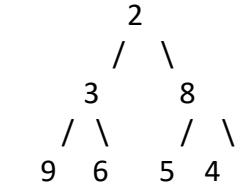


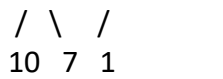
Går igjen til nederste node som er 3 og bytter 3 og 9

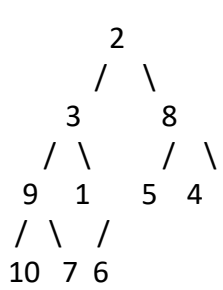


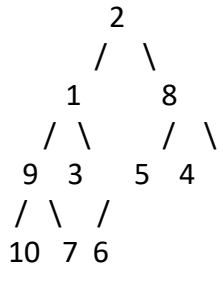
Bytter så 10 og 3

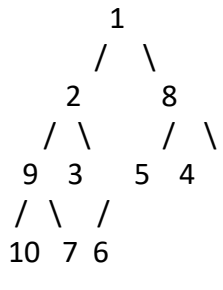


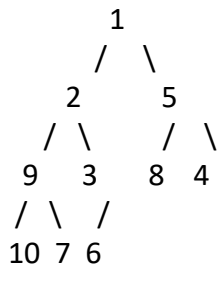


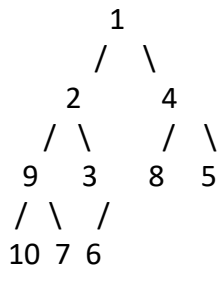












Oppgaver fra uke 17

Oppgave 1

```java
    public int[] finnKMinsteSelectionSort(int[] tabell, int k) {
        for (int i = 0; i < k; i++) {
            int minIndex = i;
            for (int j = i + 1; j < tabell.length; j++) {
                if (tabell[j] < tabell[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = tabell[i];
            tabell[i] = tabell[minIndex];
            tabell[minIndex] = temp;
        }
        int[] resultat = new int[k];
        System.arraycopy(tabell, 0, resultat, 0, k);
        return resultat;
    }
```

Kjører k runder med selectionsort. Kjøretid blir da O(n*k).

```java
public int[] finnKMinsteInsertionSort(int[] tabell, int k) {
    int[] resultat = new int[k];
    for (int i = 0; i < k; i++) {
        resultat[i] = tabell[i];
    }

    for (int i = 1; i < k; i++) {
        int hold = resultat[i];
        int j = i - 1;
        while (j >= 0 && resultat[j] > hold) {
            resultat[j + 1] = resultat[j];
            j--;
        }
        resultat[j + 1] = hold;
    }
    for (int i = k; i < tabell.length; i++) {
        if (tabell[i] < resultat[k - 1]) {
            int j = k - 2;
            while (j >= 0 && resultat[j] > tabell[i]) {
                resultat[j + 1] = resultat[j];
                j--;
            }
            resultat[j + 1] = tabell[i];
        }
    }

    return resultat;
}
```

Samme kjøretid som selection, O(n*k).

```java
public int[] finnKMinsteHeap(int[] tabell, int k) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

    for (int i = 0; i < tabell.length; i++) {
        if (maxHeap.size() < k) {
            maxHeap.offer(tabell[i]);
        } else if (tabell[i] < maxHeap.peek()) {
            maxHeap.poll();
            maxHeap.offer(tabell[i]);
        }
    }

    int[] resultat = new int[k];
    int i = 0;
    for (int tall : maxHeap) {
        resultat[i++] = tall;
    }
    Arrays.sort(resultat);
    return resultat;
```

Med heapsort får vi O(n log k).

Oppgave 2

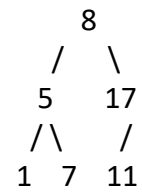
a)

Et tre er balansert hvis det er mindre enn 1 forskjell mellom høyden på venstre og høyre side, for hver node er venstre undertre et balansert tre og for hver node er høyre undertre et balansert tre.

b)

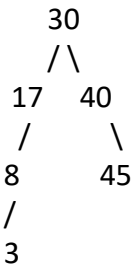
Det er ubalansert fordi det er over 1 node forskjell fra venstre og høyre side.

Slik ser treet ut etter en høyrerotasjon:



Nå er det balansert.

c)



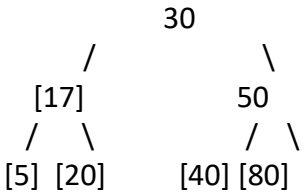
d)

Dersom trærne er skeive (ikke balansert) vil de være ineffektive å utføre operasjoner på.

Oppgave 3

a)

20, 50, 30, 5, 40, 80, 17



b)

1. Ingen noder i treet har nøyaktig ett barn. Sant
2. Treet inneholder maksimalt en 3-node. Usant
3. Alle blad er på samme nivå. Sant
4. Dersom treet bare inneholder 2-noder, så vil det tilsvare et balansert BS-tre. Sant
5. Alle 3-noder må være blad. Usant

Oppgave 4

a)

Rekkefølgen på nodene når man bruker depth first algoritmen med node 9 som startnode er: 9, 6, 3, 1, 2, 4, 7, 5, 8.

b)

Rekkefølgen på nodene når man bruker breadth first algoritmen med node 1 som startnode er: 1, 2, 3, 4, 6, 9, 7, 8, 5.

c)

Starter på node 6, som ikkje har naboer, og fortsetter å bruke algoritmen til vi får lagt til alle nodene og får da: 6, 5, 4, 2, 3, 1, 7

For å få topologisk ordning så må vi snu rekkefølgen: 7, 1, 3, 2, 4, 5, 6

d)

For å finne ut om det finnes en topologisk ordning, kan man legge til en teller som teller besøk. Så for kvar iterasjon, hvis du finner en node b som ikke er besøkt og har ingen uoppdaget naboer, så legger vi til v i stabel og øker telleren. Etter løkken er blitt ferdig så sjekker vi om telleren er mindre enn antallNoder, hvis dette stemmer fins det ikke en topologisk ordning.

e)

Steg	aktiv node	A	B	C	D	E	F	G	H	I	Uberørte noder
1	A	0	2	∞	∞	∞	9	5	∞	∞	B C D E F G H I
2	B	0	2	6	∞	∞	9	5	∞	∞	C D E F G H I
3	C	0	2	6	8	∞	9	5	11	∞	D E F G H I
4	D	0	2	6	8	9	9	5	9	∞	E F G H I
5	E	0	2	6	8	9	9	5	9	12	F G H I
6	F	0	2	6	8	9	9	5	9	10	G H I
7	G	0	2	6	8	9	9	5	9	7	H I
8	H	0	2	6	8	9	9	5	9	7	I
9	I	0	2	6	8	9	9	5	9	7	

Ved bruk av Dijkstras algoritme ser man at korteste vei fra A til H er: A, B, C, D, H