Seksjon 1

1 OPPGAVE

1. (25%)

Skriv en metode (du kan fritt bruke Java-biblioteket)

int threeSumNotZero(int[] a)

som returnerer antallet tripler *i,j,k* med *0 <= i < j < k < a.length* slik at *a[i] + a[j] + a[k] != 0*. Du kan anta at alle elementer i tabellen *a* er forskjellige. For å oppnå full skåre bør din metode kjøre i *O(N^2 log N)* tid, der *N = a.length*.

Skriv ditt svar her...

2 OPPGAVE

2. (25%)

I denne oppgaven kan du bruke følgende API uten å ha implementert den.

```
public class MinPQ<Key extends Comparable<Key>>
        MinPQ()  //  create a minimum priority queue
void  insert(Key v)  //  insert a key into the priority queue
Key   delMin()  //  return and remove the smallest key
```

a) Bruk API-en og ingen andre for å skrive en metode *void sort(int[] a)* som sorterer tabellen *a*.

b) Anta at API-en er implementert slik at *MinPQ()* tar konstant tid, og *insert(Key v)* og *delMin()* tar logaritmisk tid i størrelsen på køen. Estimer kjøretiden i verste fall til *void sort(int[] a)* for *N = a.length*.

Skriv ditt svar her...

3 OPPGAVE

3. (25%)

Se på følgende fragment av en enkel implementering av TwoThreeTree.

```
public class TwoThreeTree<Key extends Comparable<Key>> {
private Node root;
private class Node{
  private Key key1;
  private Key key2;
  private Node left, mid, right;
  public Node(Key k1, Key k2, Node l, Node m, Node r){
    key1 = k1; key2 = k2;
    assert key1!=null && (key2==null || key1.compareTo(key2) < 0);
    left = l; mid = m; right = r;
    assert key2!= null || mid==null ;}
} // End of class Node

public TwoThreeTree(Key k1, Key k2, Node l, Node m, Node r) {
  root = new Node(k1,k2,l,m,r); }

public Key max(Node r){ ... } // returns the greatest key if r is not null; returns null otherwise

} // End of class TwoThreeTree
```

Et objekt av denne klassen representer et såkalt 2-3-tre, med datainvariantene som vist i påstandene (assertions) i *Node*-konstruktøren. En 2-Node er en *Node* med en nøkkel og to barn; En 3-Node er en *Node* med to nøkler og tre barn. Barna kan være *null*.

a) Hvilke av følgende gir korrekte *TwoThreeTree*-objekter: *TwoThreeTree(0,null,null,null,null); TwoThreeTree(0,0,null,null,null); TwoThreeTree(null,0,null,null,null); TwoThreeTree(-1,1,null,null,null); TwoThreeTree(0,null,null,null,0)*? Grunngi alle svar.

b) Gi definisjonen av et 2-3 søketre og av et *balansert* 2-3 søketre.

c) Under antakelse av at *Node r* er et 2-3 søketre, skriv en metode *Key max(Node r)* som returnerer den største nøkkelen som forekommer under r hvis r ikke er *null*, og *null* ellers.

d) Analyser kjøretiden i verste fall for din metode under c), både for et 2-3 søketre og for et balansert 2-3 søketre.

Skriv ditt svar her...

4 OPPGAVE

4. (25%)

Dette er en oppgave om urettede grafer der enhver kant har en vekt

a) Gi definisjonen av et utspennende tre av minimal vekt til en slik vektede urettede graf.

b) Beskriv en algoritme for å finne et utspennende tre av minimal vekt til en vektede urettede graf. Sortering og API-en til Union-Find kan brukes uten videre.

c) Forklar hvordan din algoritme under c) fungerer med et eksempel av en graf med fire noder og seks kanter med alle kanter av ulik vekt.

Skriv ditt svar her...