Seksjon 1

1 OPPGAVE

1. (25%)

Write a method (you can freely use the Java library)

int threeSumNotZero(int[] a)

that returns the number of triples *i,j,k* with *0 <= i < j < k < a.length* such that *a[i] + a[j] + a[k] != 0*. You can assume that all elements of the array *a* are different. To get the full score your method should run in *O(N^2 log N)* time, where *N = a.length*.

Fill in your answer here

2 OPPGAVE

2. (25%)

In this exercise you may freely use the following API without having to implement it.

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) Using the above API and no other, write a method *void sort(int[] a)* that sorts the array *a*.

b) Assume that the above API is implemented in such a way that *MinPQ()* takes constant time, and *insert(Key v)* and *delMin()* take time logarithmic in the size of the queue. Estimate the worst-case run-time of *void sort(int[] a)* for *N = a.length*.

Fill in your answer here

3 OPPGAVE

3. (25%)

Consider the following fragment of a simple implementation of 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
```

An object of this class represents a so-called 2-3-tree, satisfying the data invariants as expressed in the assertions of the Node-constructor. A 2-Node is a *Node* with one key and two children; a 3-Node is a *Node* with two keys and three children. Children can be *null*.

a) Which of the following give correct *TwoThreeTree*-objects: *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)*? Motivate all answers.

b) Give the definition of a 2-3 *search* tree and of a *balanced* 2-3 search tree.

c) Assuming *Node r* is a 2-3 search tree, write a method *Key max(Node r)* that returns the greatest key that occurs under r if r is not *null*, and returns *null* otherwise.

d) Analyze the worst-case run-time of your method under c), both for a 2-3 search tree and for a balanced 2-3 search tree.

Fill in your answer here

4 OPPGAVE

4. (25%)

This is an exercise about undirected graphs in which each edge has a weight.

a) Give the definition of a minimum spanning tree (MST) of such a weighted undirected graph.

b) Describe an algorithm for finding a MST of a weighted undirected graph. Sorting and the API of Union-Find can be freely used.

c) Explain how your algorithm under c) works with an example of a graph with four nodes and six edges of different weights.

Fill in your answer here