Seksjon 1

1 OPPGAVE

1 (25 %)

Se på følgende fragment av en enkel implementering av Union-Find.

```
import java.util.LinkedList; import java.util.ArrayList; import edu.princeton.cs.algs4.StdOut;

public class MyUF {

private int[] id; // id[p] is the identifier of p
public MyUF(int N){ id = new int[N]; for (int i=0; i<N; i++) id[i]=i; }
public int find(int p) { return id[p]; };
public void union(int p, int q) {
  int idp = find(p);
  int idq = find(q);
  if (idp != idq)
    for (int i=0; i<id.length; i++) if (id[i]==idp) id[i]=idq;
}

public boolean isComponentId(int p) { ... } // returns true iff p is identifier of a component

public LinkedList<Integer> componentIdList(){ ... }
// returns all component identfiers in a linked list

public void showComponents(){ ... } // prints for each component all its elements
} // end of class MyUF
```

a) Skriv metoden *isComponentId(int p)* som returnerer *true* hvis og bare hvis *p* er en identifikator til en komponent.

b) Skriv metoden *componentIdList()* som returnerer alle komponentidentifikatorer i en kjedede liste.

c) Skriv metoden *showComponents()* som skriver ut for hver komponent alle elementer i denne komponenten.

d) Analyser kjøretiden til din metode *showComponents().*

Skriv ditt svar her...

2 OPPGAVE

2 (25%)

Se på følgende kode-fragment:

```
public class MySort{
public static void insertionSort(int[] a){
  for(int i=1; i<a.length; ++i){
    ...
  }
}
```

a) Skriv ferdig metoden *insertionSort.*

b) Gi eksempler hvor *insertionSort* kjører i lineær tid i lengden på *a.*

c) Gi eksempler hvor *insertionSort* kjører i kvadratisk tid i lengden på *a.*

d) Forklar hva ShellSort er og hvorfor den har bedre kjøretids oppførsel enn *insertionSort.*

Skriv ditt svar her...

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 Node search(Key k, Node r){ ... }
// returns a Node under r in which k occurs, if such a Node exists; 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(1,0,null,null,null); TwoThreeTree(null,0,null,null,null); TwoThreeTree(-1,0,null,null,null); TwoThreeTree(0,null,0,null,0)* ?

Et 2-3-tre er et 2-3 *søketre* hvis følgende ekstra invariant også gjelder: i enhver 2-Node er alle nøkler i *left* mindre enn *key1*, og alle nøkler i *right* større enn *key1*; i enhver 3-Node er alle nøkler i *left* mindre enn *key1*, og alle nøkler i *mid* mellom *key1* og *key2*, og alle nøkler i *right* større enn *key2*.

b) Under antakelse av at *Node r* er et 2-3 søketre, skriv en metode *search(Key k, Node r)* som returnerer noden under *r* hvor *k* forekommer, dersom en slik Node finnes, og som returnerer *null* ellers.

c) La *N* være antallet noder i et 2-3 søketre. Gi kjøretiden i verste fall for ditt svar under b).

d) Hvordan kunne man forbedret kjøretiden i verste fall som du fant under c) ?

Skriv ditt svar her...

4 OPPGAVE

4 (25%)

Gitt en rettet graf *G* med noder *0,...,V-1,* representert vha nabolister av ut-piler. Mer presist, for enhver node *v*, *er adj[v]* en kjedet liste som inneholder alle noder *w* slik at *G* har en pil fra *v* til *w*.

a) Skriv en metode *boolean reachable(int s, int t)* som returnerer en boolsk verdi *true* hvis og bare hvis *G* har en sti fra kilde *s* til mål *t*.

b) (Bonus: +10%) Skriv en metode *boolean acyclic()* som returnerer en boolsk verdi *true* hvis og bare hvis *G* er asyklisk.

Skriv ditt svar her...