1 # 1. (25%)

Se på følgende metode som tester om det finnes et trippel som har sum 0 i en tabell av heltall:

```
boolean existsThreeSumZero(int[] t) {
  int N = t.length;
  for (int i = 0; i < N-2; i++)
    for (int j = i+1; j < N-1; j++)
      for (int k = j+1; k < N; k++)
      if ( t[i] + t[j] + t[k] == 0 ) { return true; }
  return false;
}
```

a) Hva er kjøretid i verste fall til existsThreeSumZero(t) ? Begrunn svaret ditt.

b) Forklar hvordan du kunne forbedret kjøretiden til existsThreeSumZero(t).

c) Skriv en metode existsThreeSumPos(int[] t) som tester om det finnes i tabell t et trippel som har sum større enn 0. For å få alle poeng skal din metode kjøre i O(N) tid (ja, det er mulig!).

Skriv ditt svar her...

```
1
```

Maks poeng: 25

2 # 2. (25%)

La følgende metode være gitt:

```
// precondition: 0 <= lo < hi <= a.length
   int partition(int[] a, int lo, int hi)
// postcondition: for p = partition(a, lo, hi) we have lo <= p <= hi, and for all
// integers i and j: if lo <= i < p, then a[i] <= a[p], and if p < j <= hi, then a[p] <= a[j]
```

a) Ved hjelp av metoden partition() og ingen andre, skriv en metode

```
    void sort(int[] a, int lo, int hi)
```

som sorterer tallene i tabell a mellom posisjonene lo og hi (inkludert lo og hi).

b) Hva er kjøretiden til din metode sort() i verste fall? Grunngi svaret ditt.

c) Hvordan kan du forbedre sorteringsalgoritmen din slik at det verste tilfellet forekommer sjeldent?

d) Forklar i en setning hva en stabil sorteringsalgoritme er. Er dine algoritmer under punkt a) og c) stabile?

Skriv ditt svar her...

1

Maks poeng: 25

3 **3. (25%)**

Se på følgende ufullstendige program for en hash-tabell med linear probing.

```
//simple ST based on hashing with linear probing
public class LinearProbingHashST<Key extends Comparable<Key>, Value> {

private int M;     // length of hash table, NOT number of keys
private Key[] keys;
private Value[] values;
private int size;  // number of keys, INVARIANT: size < M-1 (at least one empty place)


public LinearProbingHashST() {
   M = 31; // typical values of M are 31, 997, 65521, 1048573
   keys  = (Key[]) new Comparable[M];
   values  = (Value[]) new Comparable[M];
   size = 0;
}

private int hash(Key key){
    return ( key.hashCode() & 0x7fffffff ) % M;
}

private int getpos(Key key){ // returns position where key should be
   <<< insert answer for (a) here >>
   return pos; // correct since there is always one empty position
}

public Value get(Key key) {
    return values[getpos(key)];
}
```

```
public void put(Key key, Value v) { // updates value of key if key is present,
// adds (key,v) pair if key is not present and size < M-1,
// prints message "hashtable capacity exceeded" otherwise.
    <<< insert answer for (b) here >>
}

public static void main(String[] args)  {
    In infile = new In(args[0]);
    LinearProbingHashST<String,Integer> st = new LinearProbingHashST<>();
    while (!infile.isEmpty()) {
        String key = infile.readString();
        <<< insert answer for (c) here >>
}
} // End of class
```

a) Skriv ferdig metoden getpos(Key key).

b) Skriv ferdig metoden put(Key key, Value v).

c) Skriv ferdig metoden main(String[] args) slik at hash-tabellen lagrer for enhver streng som forekommer i input-filen hvor ofte den forekommer.

Skriv ditt svar her...

```
1
```

Maks poeng: 25

4 4. (25%)

Se på følgende metode for å spasere gjennom en urettet graf fra en gitt kildenode, samt at nodene i kildens sammenhengende komponent blir markert. Grafen G har noder 0,...,V-1 og E kanter som er representert gjennom nabolister. Mer presist, for enhver node v, er adj[v] en kjedet liste som inneholder alle noder w slik at G har en kant mellom v og w.

```
boolean[] walk(Integer s) {
   boolean[] marked = new boolean[V];
   Stack<Integer> stack = new Stack<>();
   stack.push(s);
   while (!stack.isEmpty()) {
      Integer u = stack.pop();
```

```
            StdOut.print(u+" ");
            marked[u]=true;
            for (Integer v : adj[u]) { if (!marked[v]) { stack.push(v); } }
        }
        StdOut.println();
        return marked;
    }
```

Se på den komplette grafen med fire noder, dvs. med noder 0,1,2,3 der hver naboliste adj[u] er [0,1,2,3], som itereres fra venstre til høyre i for-løkken.

a) Hva blir skrevet ut av kallet walk(0) ?

b) Vis at metoden walk() bruker i verste fall en stabel på størrelse O(E).

c) Skriv en bedre metode walk() som markerer de samme nodene men bruker i verste fall en stabel på størrelse O(V).

Skriv ditt svar her...

```
1
```

Maks poeng: 25