# INF102 Inlevering 1 hausten 2013

Paul Grant

September 27, 2013

## Oppgave 1

### a)

What is the running time for these code fragments? Answer with a $\Theta$ -relation.

Given $n \leq p \leq m$ there are three separate loops that must be taken into account giving us the relation $\Theta(n * p * m)$.

### b)

Given $p \leq q = n$ and both p and q are affected by the if/else statement depending on the value of m – this is a binary search. So $\Theta(logn)$.

### c)

Given $n * \sum_{k=0}^{log(n)} \frac{1}{2^k}$ this geometric set converges on 2 giving us $\Theta(n)$.

## Oppgave 2

### a)

$20n^4 + 123n^2 + 1000log(n)$<br>
$\sim (20n^4)$

### b)

$120n^3 + 5n^2log(n)$
$\sim (120n^3)$

### c)

$100n^4 + 2^n$
$\sim (2^n)$

### d)

$n + n^log(n) + log(n)$
$\sim (n)$

### e)

$2^{10} + n$
$\sim (n)$

### f)

$2^{10}n$
$\sim (2^{10}n)$

## Oppgave 3

### a)

To implement sorterFrekvense(Comparable[] a), I started by creating a class called Tuples to create a key, value pair to maintain the values in the Comparable array.

---

```
  //Tuples.java
import org.omg.CORBA.OMGVMCID;

public class Tuple implements Comparable {

    private int key;

    private Comparable value;

    public Tuple(int key, Comparable value) {
      this.key = key;
      this.value = value;
    }
    public int getKey( {
      return key;
    }
    public Comparable getValue( {
      return value;
    }

    @Override
    public int compareTo(Object o) {
      Tuple tuple = (Tuple) o;
      if (key > tuple.getKey( {
        return 1;
      } else if (key < tuple.getKey( {
        return -1;
      } else {
        return 0;
      }
    }
    public String toString( {

      return ‘‘Key, Value: ’’+key+‘‘\t ’’+value;
    }
```

```
}
\\\\

//Oppgave3a.java
import java.util.ArrayList;

public class Oppgave3a {

  @SuppressWarnings(``unchecked'')
  static Comparable[ sorterFrekvens(Comparable[] input) {
    Merge.sort(input);
    int counter = 0;

    ArrayList<Tuple> myData = new ArrayList<Tuple>();

    while (counter < input.length) {

      int howmany = 0;
      while (( counter + howmany + 1 < input.length)
      && (input[counter + howmany].compareTo(input[counter
      + howmany + 1])) &=& 0) {
        howmany++;\
      }

      myData.add(new Tuple(howmany + 1, input[counter + howmany]));
      counter += howmany + 1;
    }

    Tuple[ returnable = new Tuple[myData.size()[]<++>;

    for (int i = 0; i < myData.size(; i++) {
      returnable[i] = myData.get(i);
    }

    for (int i = 0; i < returnable.length; i++) {
      System.out.printf(``Value, Key: %d\t %d\n'',
      returnable[i].getValue(, returnable[i].getKey();
    }
    int countThis = 0;
    for (int i = input.length -1, j = 0; (i >= 0) &&
        j<returnable.length; j++) {
      while (++countThis < returnable[j].getKey( {
        input[i--] = returnable[j].getValue(;
      }
      countThis = 0;
    }
    return input;
  }

  public static void main(String[] args) {
    Integer[ stuff = {3,4,1,8,5,3,5,5};

      Comparable[ somethingsomethingdarkside =
          sorterFrekvens(stuff);
  for (int i = 0; i < somethingsomethingdarkside.length; i++)
    {
   System.out.printf(‘‘%s \n’’,
     somethingsomethingdarkside[i].toString();
  }
 }
}
```

### b)

The $\Theta$ relation for my sorterFrekvens(Comparable[] a) method is $\Theta(n^2)$ because of the for loop that goes through the input array twice.

### c)

i) copy elements in b to a and after merge sort a

This uses a for loop to transfer all the elements from b to a and then to merge sort which takes $Nlog(N)$. So $N^2log(N)$ gets the $\sim(N^2)$. ii) sort each row in b, and after merge together the rows over in a

This uses a merge sort for each row, so we have $\frac{Nlog(N)}{N} * Nlog(N)$. Then we get the $\sim(NlogN)$ both rows and columns in b er $n = 2^k$, when k is a whole number. Proposition F says that top-down mergesort uses between $1/2N\log(N) and Nlog(N)$ compares to sort any array of length N.

### d)

### e)

The frequence sorting algorthm that we created in the section a can be used to write this implementation. All that we have to do is not take in the set of numbers, but instead take in a set of strings and compare the frequency of the given strings. This is fine because we have used Comparable.

## Oppgave 4

### a)

selection sort, shell sort, merge sort java files attached.

### b)

shell sort and merge sort further

### c)

give an estimation of how large a table each of these algorthms can sort in a minute given the machine that you are using.