ID1020 – Lab 2
Complexity of Algorithms

1 Organisation

This is a theoretical lab that you must answer in Canvas.

1.1 Goals

This lab has the following goals:

- Reason about computational complexity of algorithms.
- Explore the relation between different asymptotic behaviour.
- Estimate time complexity from data.
- Reason about space complexity.

2 Tasks

2.1 Asymptotic behaviour (R) – 20P

Write the Big O-notation for each of the following functions, then order them. (all logs are base 2 unless specified otherwise)

$$f_1 : n \mapsto 3n^2 - 10n, f_2 : n \mapsto n^3 - 17, f_3 : n \mapsto 2^{n \log n}, f_4 : n \mapsto \sqrt{n^2 + n}, f_5 : n \mapsto \log(n^5),$$
$$f_6 : n \mapsto \sqrt{n}, f_7 : n \mapsto n \log n, f_8 : n \mapsto n^n, f_9 : n \mapsto \log n, f_{10} : n \mapsto \log_3 n$$

2.2 Complexity Estimation (R) – 40P

(a) Imagine that you wrote a program and measured its execution time as a function of N. The results of your execution are shown in the table below.

N	time in seconds
64	0
128	0
256	0.0015
512	0.0135
1024	0.09
2048	0.6225
4096	3.9015
8192	25.6515
16384	166.3815
32768	1068.1575
65536	6945.4245

You can assume that the program's running time follows a power law $T(N) \sim a \cdot N^b$. Estimate the order of growth of the running time as a function of N.

(b) In the book, TwoSum and ThreeSum algorithms were discussed and followed by an optimized version of both. We have packaged the four algorithms **TwoSumFast**, **TwoSum**, **ThreeSumFast** and **ThreeSum** in a jar file that you can download from (`https://github.com/id1020/lab2-runningtimes/blob/master/runningtimes.jar?raw=true`). Also, a simple program has been added to measure the execution time of each of the algorithms for some given input.

For example, if you want to measure the time taken by TwoSum for a file of 1000 numbers, you should run

```
> java -jar runningtimes.jar 2sum 1000
 1000 0.006
```

The program prints back the input size which is 1000 and the time taken which is 0.006 seconds.

Estimate the amount of time it would take to run **TwoSumFast**, **TwoSum**, **ThreeSumFast** and **ThreeSum** on your computer to solve the problems for a file of 1048576 numbers. Notice that measuring the execution time is not feasible for all points, so you have to think about predicting time for those point you can't measure.

2.3 Space Complexity (Memory Usage) (R) – 40P

Assuming a 64-bit CPU architecture and Java 7, given the class definitions described in Listing 1 compute the amount of memory needed by each of the following objects considering their respective assumptions:

(a) **Accumulator**.

(b) **Transaction**, assuming references to *String* and *Date* objects.

(c) **FixedCapacityStackOfStrings**, assuming capacity *C=8* and fixed size string entries of size *S=8* chars. Remember to include the amount of memory required to store the strings in your calculation.

(d) **Block**

(e) **INodeFile**, assuming that an inode name consist of 8 chars, and that there are 3 Blocks per INodeFile. Remember to include the amount of memory required to store the Blocks in your calculation.

(f) **INodeDirectory**, consider the same assumptions from **INodeFile**, and also assume that there are 10 INodeFiles per INodeDirectory. Remember to include the amount of memory required to store the INodeFiles in your calculation.

Listing 1: M_1

```java
public class Accumulator
{
    private double total;
    private int N;
}

public class Transaction
{
    private final String who;
    private final Date when;
    private final double amount;
}

public class FixedCapacityStackOfStrings
{
    private String[] a = new String[C]; // stack entries
    private int N; // size
}



class INode implements Comparable<byte[]>
{
    protected byte[] name;
    protected long modificationTime;
    protected long accessTime;
}

class INodeFile extends INode {
    private long header;
    private Block[] blocks;
}

class INodeDirectory extends INode
{
    private INodeFile[] children;
}

public class Block implements Comparable<Block>
{
    private long blockId;
    private long numBytes;
    private long generationStamp;
}
```