ALGORITHMICS - Bachelor of Software Engineering, 2024

LAB GUIDE. SESSION 1.2

GOALS:

- **Examples of execution time measurements in iterative algorithms**
- **Comparison of two algorithms**

1. Introduction

In this session, we will see the measurement of execution times of iterative algorithms (on some example models) and how to compare a pair of algorithms or development environments with each other.

2. Some iterative models

Loop1.java, **Loop2.java**, **Loop3.java** and **Loop4.java** classes are provided to you. They are iterative models on which we must work to determine their time complexity.

Execute the programs with the Eclipse IDE and check the times for different sizes of the problem to conclude whether the times obtained are the expected considering the complexity of each program.

YOU ARE REQUESTED TO:

After measuring times, fill in the table:

TABLE1 (times in milliseconds and WITHOUT OPTIMIZATION):

We can indicate "OoT" for times above 1 minute.

N	*tLoop1*	*tLoop2*	*tLoop3*	*tLoop4*
100				
200				
400				
800				
1600				
3200				
6400				
12800				
25600				
51200				

Explain whether the different times obtained agree with what was expected, according to the theoretical complexity of the four cases.

3. Creation of iterative models of a given time complexity

YOU ARE REQUESTED TO:

Implement three new classes **Loop5.java**, **Loop6.java** and **Loop7.java**, that simulate iterative algorithms with a complexity $O(n^2 \log^2 n)$, $O(n^3 \log n)$ and $O(n^4)$ respectively.

After implementing the classes, measure their execution times and fill in the table:

TABLE2 (times in milliseconds and WITHOUT OPTIMIZATION):

We can indicate "OoT" for times above 1 minute.

N	*tLoop5*	*tLoop6*	*tLoop7*
100			
200			
400			
800			
1600			
3200			
6400			

Explain whether the different times obtained agree with what was expected, according to the theoretical complexity of the four cases.

4. Comparison of two algorithms

To compare the execution time of two algorithms (with each other) we will calculate the **division ratio** of the execution times that these algorithms take for the same size of the problem.

When the size of the problem grows and the algorithms to be compared have different complexity:

- If the ratio tends to **0** ➔ the one associated with the numerator is the least complex (that is, the best).
- If the ratio tends to **infinity** ➔ the one associated with the numerator is the most complex (that is, the worst).

When the size of the problem grows, and the algorithms have the same complexity, the ratio tends to a **constant**, called **implementation constant**. That constant is the one that tells us which one of the two algorithms with the same complexity is better:

- If the implementation constant < 1 ➔ The one in the numerator is better.
- If the implementation constant > 1 ➔ The one in the denominator is better.
- If the implementation constant = 1 ➔ The algorithms are exactly equal.

A. Two algorithms with different complexity

YOU ARE REQUESTED TO:

After measuring times, fill in the table:

TABLE3 (times in milliseconds and WITHOUT OPTIMIZATION):

We can indicate "OoT" for times above 1 minute.

n	*tLoop1*	*tLoop2*	*t1/t2*
100			
200			
400			
800			
1600			
3200			
6400			
12800			
25600			
51200			

Explain whether the different times and their quotient agree with what was expected according to the theoretical complexity.

B. Two algorithms with the same complexity

YOU ARE REQUESTED TO:

After measuring times, fill in the table:

TABLE4 (times in milliseconds and WITHOUT OPTIMIZATION):

We can indicate "OoT" for times above 1 minute.

n	*tLoop3*	*tLoop2*	*t3/t2*
100			
200			
400			
800			
1600			
3200			
6400			
12800			
25600			
51200			

Explain whether the different times and their quotient agree with what was expected according to the theoretical complexity.

C. Same algorithm in different development environments

Finally, we are going to work with the same algorithm in two different environments with the goal of comparing, for that algorithm, such environments (**Loop4.py** and **Loop4.java**). Subsequently, we will measure the times (both in Python and in the case of Java for the cases of WITHOUT OPTIMIZATION and WITH OPTIMIZATION). The quotient for the same problem sizes will oscillate around a value (a constant that relates both environments). If this constant is less than 1, the one we have in the numerator is the best one and if it is greater than 1, the one we have in the denominator is the best one.

NOTE: it should not be surprising that the time ratio for the two Java cases is not stable at all, once seen in lab 0 what the execution of Java classes with the optimizer (JIT) entails.

YOU ARE REQUESTED TO:

After measuring times, fill in the table:

TABLE5 (times in milliseconds):

We can indicate "OoT" for times above 1 minute.

n	*tLoop4 (Python) – t41*	*tLoop4 (Java without optimization) – t42*	*tLoop4 Java with optimization) – t43*	*t42/t41*	*t43/t42*
200					
400					
800					
1600					
3200					
6400					

Explain whether the different times and their quotient agree with what was expected according to the theoretical complexity.

5. Work to be done

- An `algstudent.s12` **package** in your course project. The content of the package should be the Java files used and created during this session.
- A `session12.pdf` **document** using the course template (the document should be included in the same package as the code files). You should create one activity each time you find a "YOU ARE REQUESTED TO" instruction (e.g., in this document you should do 5 different activities answering the different questions you will find in each point).

Deadline: The deadline is one day before the next lab session of your group.