ALGORITHMICS - Bachelor of Software Engineering, 2025

LAB GUIDE. SESSION 3

GOALS:

- **Divide and Conquer: recursive models and examples**

1. Introduction

The study of the time complexity and execution time of recursive Divide and Conquer (D&C) algorithms will be addressed, both for the **subtraction** and **division** approaches.

The second part of the lab consists of providing a solution to a specific problem using the D&C technique, measuring the execution times of the algorithm that is implemented and proceeding to the final analysis of the times obtained.

We will continue this lab measuring the times WITHOUT OPTIMIZATION, since in this case of recursive models the JIT seems to produce greater time distortion than in the case of iterative models (seen in the previous sessions). In the tables of times that we will create, if any time exceeds 1 minute, we will indicate Out of Time (OoT).

2. Divide and Conquer by subtraction

You are provided with the following 10 classes that you should try to understand one by one:

`Subtraction1.java` and `Subtraction2.java` classes have an approach by subtraction with `a=1`, which involves a large expenditure of **stack memory**. In fact, it overflows when the problem size grows to several thousands. Fortunately, that type of problems will be better solved with an iterative solution (with loops) than with this solution (subtraction with `a=1`).

`Subtraction3.java` class has an approach by subtraction with `a>1`, which involves a large time of execution (exponential; not polynomial). This means that for a relatively large size problem the algorithm does not end (unfeasible time). The consequence is that we must try not to use “by subtraction” solutions with multiple calls (`a>1`).

YOU ARE REQUESTED TO:

After analyzing the complexity of the three previous classes, you are not asked to make the timetables, but to reason whether the times match the theoretical time complexity of each algorithm.

For what value of **n** do the **Subtraction1** and **Subtraction2** classes stop giving times (we abort the algorithm because it exceeds 1 minute)? Why does that happen?

How many years would it take to complete the **Subtraction3** execution for n=80? Reason the answer.

Implement a **Subtraction4.java** class with a complexity $O(n^3)$ and then fill in a table showing the time (in milliseconds) for n=100, 200, 400, 800, ... (until OoT).

Implement a **Subtraction5.java** class with a complexity $O(3^{n/2})$ and then fill in a table showing the time (in milliseconds) for n=30, 32, 34, 36, … (until OoT).

How many years would it take to complete the **Subtraction5** execution for n=80? Reason the answer.

3. Divide and conquer by division

`Division1.java`, `Division2.java` and `Division3.java` classes have a recursive approach by division, being the first of type $a<b^k$, the second of type $a=b^k$ and the remaining one $a>b^k$.

YOU ARE REQUESTED TO:

After analyzing the complexity of the three previous classes, you are not asked to make the timetables, but to reason whether the times match the theoretical time complexity of each algorithm.

Implement a **Division4.java** class with a complexity $O(n^2)$ (with $a<b^k$) and then fill in a table showing the time (in milliseconds) for n=1000, 2000, 4000, 8000, … (up to OoT).

Implement a **Division5.java** class with a complexity $O(n^2)$ (with $a>b^k$) and then fill in a table showing the time (in milliseconds) for n=1000, 2000, 4000, 8000, … (up to OoT).

4. Two basic examples

`VectorSum1.java` solves the simple problem of adding the elements of a vector in three different ways, and `VectorSum2.java` measures times of these three algorithms for different sizes of the problem.

`Fibonacci1.java` solves the problem of calculating the Fibonacci number of order `n` in four different ways, and `Fibonacci2.java` measures times of these four algorithms for different sizes of the problem.

YOU ARE REQUESTED TO:

After analyzing the complexity of the various algorithms within the two classes, executing them and after putting the times obtained in a table, compare the efficiency of each algorithm.

5. Petanque championship organization

The organizers of a petanque championship hire us to take care of the management of the games. What nobody knows yet is the success it will have in participation. We only know that the number of participants accepted will be power of two (2, 4, 8, 16, 32, 64, etc.) but we have no idea of the number of participants that will eventually enroll. However, we must prepare a computer

application that automates the pairing of the games so that it is an almost instantaneous process as soon as the final list of participants is available. We have also been given a lot of emphasis on using the Divide and Conquer technique to prepare the application for parallel computing in the future, increasing the speed of the creation of the calendar much more if the number of participants shoots up.

The idea is to minimize the total duration of the championship, as this will save costs. Another fact that we have is that, according to the rules of the championship, every day each participant can play only against another participant.

YOU ARE REQUESTED TO:

FIRST PART

Implement the solution in a file called **Calendar.java** considering that the information about the participants will be contained in a text file like the one shown below:

```
4
Vanesa
Victor
Tomas
Alba
```

In the previous example it is specified that there are 4 participants, followed by the names of each of them. All sample files will have the same format.

The way to run the program should be something like the following:

```
java algstudent.s3.Calendar fileName.txt
```

The result displayed by console for the previous entry should be a table like the one shown below:

PLAYER/OPPONENT	DAY 1	DAY 2	DAY 3
Vanesa	Victor	Tomas	Alba
Victor	Vanesa	Alba	Tomas
Tomas	Alba	Vanesa	Victor
Alba	Tomas	Victor	Vanesa

There are three days of games (n-1) in which all the participants play against all and none of the participants play on the same day more than once, following the rules of the championship.

What complexity does the algorithm created have? It is very important to know it to be sure that our solution is the best possible.

SECOND PART

The organizers of the championship are so optimistic that they believe that the attendance will be very high, and they do not trust too much in our ability to develop the application. Therefore, we are asked to design a way to verify the time that our algorithm would take to classify the participants starting from a sample size n = 2 until the memory of the computer we have "hold" (**CalendarTimes.java**).

Thus, to perform these tests we will not have a text file with the real information of the participants. Instead, we must design a small program in Java, which, using the algorithm created above, is able to automatically create classifications with "random" participants, whose number is increased following a progression in the form of power of two. Obviously, we must measure the times for each size without considering text outputs per console that distort the results.

n	*t Calendar*
2	…..
4	…..
8	…..
16	…..
32	…..
…	…..
Oveflow heap	

Check whether the times obtained in the previous section meet or not the theoretical complexity of the designed algorithm in the first step.

OPTIONAL WORK:

If you have time and interest, you can leave the organizers very satisfied if the design of the algorithm considers the possibility that in the future the rules of the championship will change, and the number of participants does not have to be power of two (e.g., there could be 10 participants). In that case, the number of days needed to play could be **n** instead of **n-1** as in the case of a number of participants that is power of two.

What changes should be made in the first version of the algorithm?

6. Work to be done

- An `algstudent.s3` **package** in your course project. The content of the package should be the Java files used and created during this session.
- A `session3.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.

Deadline: The delivery of this practice will be carried out, in time and form, according to the instructions given by the lab teacher.