ALGORITHMICS - Bachelor of Software Engineering, 2021

LAB GUIDE. SESSION 3

GOALS:

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

1. Basic recursive models

You are provided with the following 6 classes that you should try to understand one by one:

`Subtraction1.java` and `Subtraction2.java` classes have a scheme 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 a scheme 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 (untreatable time). The consequence is that we must try not to use "by subtraction" solutions with multiple calls (`a>1`).

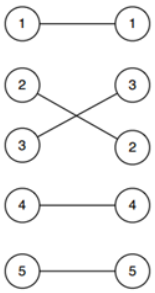
`Division1.java`, `Division2.java` and `Division3.java` classes have a recursive scheme by division, being the first of type $a<b^k$, the second of type $a=b^k$ and the remaining one $a>b^k$.

2. Counting inversions

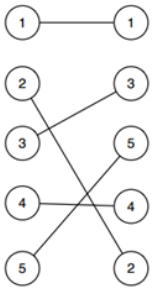
The fact of **counting inversions** is highly useful for measuring **similarity between rankings** of any type of elements (e.g., most viewed movies, favorite products, most visited websites, article recommendations, etc.).

Let's suppose that a ranking is a list of elements $\mathbf{a_1, a_2, a_3, \ldots, a_n}$. An inversion would be any pair **(i, j)** such that **i < j** but $\mathbf{a_j < a_i}$. Obviously, two identical rankings have no inversions.

Let's see an example below where two rankings are compared. Numbers can represent any type of product. As an example, numbers on the left could be the favorite web pages of a user and the number on the right of another user.



It can be seen that both rankings are very similar, having only one inversion (2, 3). The following two rankings are much different, having 4 inversions on the right side (4, 5), (2, 3), (2, 5) and (2, 4).



One way to solve this problem (to know how many inversions there are) would be to check each pair of elements (method commonly called **brute force**). It's not a great idea because we would have a complexity **O(n^2)**. Using **divide and conquer**, this problem can be solved with a much better complexity: **O(nlogn)**.

TO DO:

A. Work to be done

- An `algstudent.s3` **package** in your course project. The content of the package should be:
  - All the basic recursive models that were given with the instructions for this session together with two new classes:
    - `Substraction4.java`. You should create a new class and implement a recursive method by subtraction with a complexity $O(3^{n/2})$.
    - `Division4.java`. You should create a new class and implement a recursive method by division with a complexity $O(n^2)$ and a number of subproblems = 4.
- A **PDF document** (`session3_1.pdf`) using the course template. The activity of the document should be the following:
  - **Activity 1. Basic recursive models**.
    - A brief explanation for each of the given classes indicating how you calculated the complexity of that class.
    - A brief explanation for each of the two new classes indicating how you calculate the complexity to get the requested one.
- An `algstudent.s32` **package** in your course project. The content of the package should be:
  - All the files that were given with the instructions for this session (`InversionTimes.java`, `ranking1.txt`, `ranking2.txt`,

`ranking3.txt`, `ranking4.txt`, `ranking5.txt`, `ranking6.txt` and `ranking7.txt`) together with two new classes:

- `InversionsQuadratic.java`. Class that solves this problem with a quadratic complexity.
- `Inversions.java`. Class that solves this problem with a `O(nlogn)` complexity.

- A **PDF document** (`session3_2.pdf`) using the course template. The activity of the document should be the following:
  - **Activity 1. Counting inversions.**
    - Make sure that your both algorithms obtain the same number of inversions (last column). Measure the times of both algorithms with the `InversionsTimes.java` (class that is provided to you and fill in the following table:

file	*t O(n2)*	*t O(nlogn)*	*t O(n2)/t O(nlogn)*	*n inversions*
`Ranking1.txt`	.....	.....	.....	14.074.466
`Ranking2.txt`	.....	.....	.....	56.256.142
`Ranking3.txt`	.....	.....	.....	225.312.650
`Ranking4.txt`	.....	.....	.....	903.869.574
`Ranking5.txt`	.....	.....	.....	3.613.758.061
`Ranking6.txt`	.....	.....	.....	14.444.260.441
`Ranking7.txt`	.....	.....	.....	57.561.381.803

    - Explain if the results are as expected and why.

B. Delivery method

First, you should include in your Java project a new `algstudent.s3` package with the following content inside it:

- All the requested source files for that package.
- The requested PDF document called `session3_1.pdf` with the corresponding activity.

Second, you should include in your Java project a new `algstudent.s32` package with the following content inside it:

- All the requested source files for that package.
- The requested PDF document called `session3_2.pdf` with the corresponding activity.

Deadlines:

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