Algorithmics	Student information	Date	Number of session
	UO: 276244	21/03/2021	4
	Surname: Beltran		
	Name: Martin		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

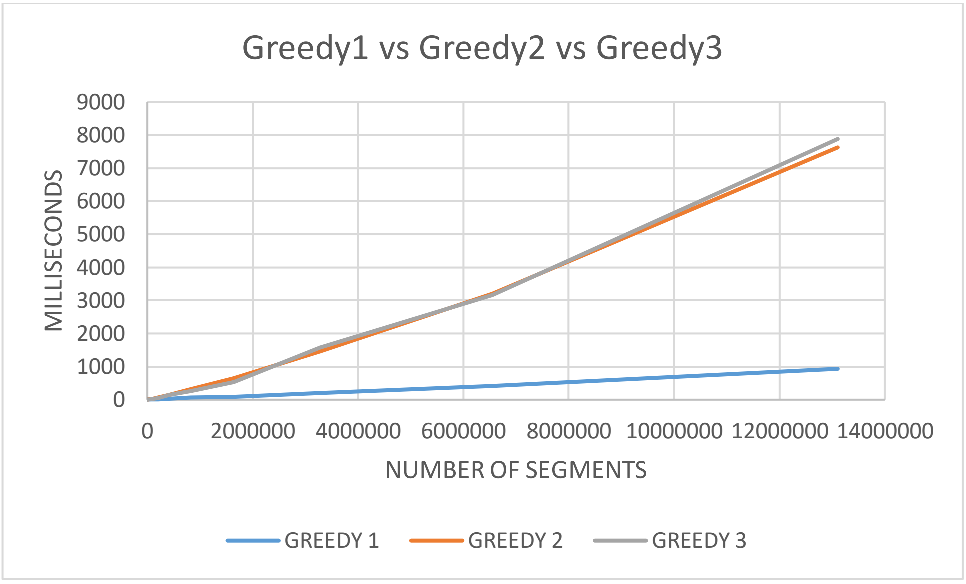
Activity 1. EXECUTION TIMES

Before starting ,I must clarify that i implement the storing of the segments with a HashMap, so that there is a relation between the name ("SX") and order of creation of the segment, with its length. The keys are the name of the segment, and the value, its length. I sort the keys in a separate list just to access them in the same order they are created, but this is done inside the constructor implementation, so it does not affect the measurements of other methods.

N	GREEDY 1	GREEDY 2	GREEDY 3
100	0	0	0
200	0	0	0
400	0	1	3
800	0	3	4
1600	0	1	1
3200	1	5	2
6400	5	6	7
12800	1	8	6
25600	3	16	11
51200	11	20	14
102400	6	21	23
204800	9	61	55
409600	21	137	136
819200	60	324	268
1638400	89	641	527
3276800	207	1457	1569
6553600	413	3195	3155
13107200	934	7625	7880

We can see the times measured in the table for each of the algorithms, in order to compare them, we should have a look at a chart:

Algorithmics	Student information	Date	Number of session
	UO: 276244	21/03/2021	4
	Surname: Beltran		
	Name: Martin		



As we can see, the execution times for the Greedy1 algorithm are much lower than those times for the remaining two algorithms. Why? Well, the heuristic for Greedy1 is just to take the segments as we find them, without sorting them. As no sorting is performed inside this algorithm, we save a lot of time. Its complexity is linear (O(n)), because It only contains one for-loop that iterate through all the segments, and the fact of getting a value from a HashMap has a complexity of O(1).

On the other hand, we see that both Greedy2 and Greedy3 take more time to be executed. The reason should be clear, it is because we sort the segments inside both algorithms, since we need to obtain the values from lower to greater or greater to lower. Anyway, the way of sorting is the same for both, the only thing that changes is the way of accessing the resulting sorted list: for Greedy2 we need to obtain values from longest to shortest (we traverse the list from the back) and for Greedy3 we need to obtain values from shorted to longest (we traverse from the start the list).

The overall complexity of both is the same: O(n * log(n)) when sorting the list with the Collections.Sort() method (it is a modification of mergesort implementation), and O(n) when using the for-loop. So in the end, the complexity would be O(n) + O(n*log(n)) = O(n*log(n)) for both Greedy2 and Greedy3.

Algorithmics	Student information	Date	Number of session
	UO: 276244	21/03/2021	4
	Surname: Beltran		
	Name: Martin		

Activity 2. ANSWER THE FOLLOWING QUESTIONS

A) Explain if any of the greedy algorithms involves the optimal solution from the point of view of the company, which is interested in maximizing the number of "pufosos":

The best algorithm for the company would be Greedy2, because it computes the greatest number of pufosos.

B) Explain if any of the greedy algorithms involves the optimal solution from the point of view of the player, which is interested in minimizing the number of "pufosos":

The best algorithm for the player would be Greedy3, because it computes the lowest number of pufosos.

C) Explain the theoretical time complexities of the three greedy algorithms, according to the implementation made by each student, depending on the size of the problem n:

This explanation has been made after presenting the results of the experiment, as I considered it a better idea to show the results and why I obtained those results before answering any question.

D) Explain if the times obtained in the table are in tune or not, with the complexities set out in the previous section:

We can see, thanks to the graph, that the times are in tune with the theoretical complexity of the three algorithms, as Greedy1 follows a linear representation, and Greedy2 and Greedy3 a representation following O(N * log(N)).