Algorithmics	Student information	Date	Number of session
	UO: 270534	03/17/21	6
	Surname: Iglesias Préstamo		
	Name: Ángel		

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. [Execution times]

Greedy algorithms take all data in a particular problem, and then set a rule through which they will find the solution. We can clearly see that the algorithm must implement the following:

1) To achieve this, we can just sort the segments given a certain rule.
2) Compute the value at which the segment starts, remember it is an accumulative property, this is: we must set it as the point at which the previous segment ended. Where it ends: the sum of the starting point and the length of the segment itself. And the mid-point: the mean of both values.
3) Increment the value of the cryptocurrency by the mean value calculated.
4) Search for the next segment. Applying the same rule as before.

These rules are given in the statement the following way:

1) **Greedy 1**: place the segments in the same order is they appear in the file.
2) **Greedy 2**: place the segments from longest to shortest length → descending order.
3) **Greedy 3**: put them in increasing order of length → ascending order.

Going deeper in the implementation of the algorithms themselves, I will explain how I met the solution, in some sort of pseudocode:

1) We read the file.
2) The first line of such file will be the number of segments: n.
3) From the second line on, we will store the length of each segment in some sort of dictionary – which in my implementation is nothing more than a Linked List of Pairs, in a few I will explain the reason with – its corresponding position, starting from 0.
4) We sort the list – by the lengths – following the previous described rules.
5) For each element in the dictionary, we compute some operations:
    a) First, we update the value of *y* to the corresponding length, remember it's an accumulative property.
    b) We compute the mean value of *x* and *y*.
    c) We update the value of *x*, so that it's equal to *y*.

Last, in order me to explain – probably – the weirdest part of my code, I'm going to explain why I created the Pairs class. I'm willing to use the Collections.sort() method (with the helping hand of Collections.reverseOrder(), when needed), and so, I need to pass a List as an argument, but for each of the elements of the Linked List I also want to track the original values of the position they occupied in the file. In order me to track both, I had two options: first, I could use a HashMap, but it may be a bit hard to sort by the value, instead of the key, and I couldn't use the lengths for them to be the key as they could be repeated. So, I just created a new *comparable object*: the Pair. I also separated the rough implementation from the TUI, as I believed both were separated things, that's why I must complicate myself a little bit creating two separated methods: *calculatePufosos()* and *print()*. More in more, my focus was to create three methods: *greedy1()*, *greedy2()* and *greedy3()*; that were as relatable as possible: with the same structure but changing the sorting part.

n	*tGreedy1* $(10^{-6}\ s)$	*tGreedy2* $(10^{-6}\ s)$	*tGreedy3* $(10^{-6}\ s)$
100	147	973	946
200	304	2,660	2,935
400	745	8,886	8,814
800	1,627	19,905	14,887
1,600	5,242	31,284	36,707
3,200	11,650	67,938	62,712
6,400	21,242	150,358	144,174
12,800	34,213	305,253	310,570
	$O(n)$	$O(n \log(n))$	$O(n \log(n))$

Activity 2. [Answer the following questions]

a. **Task 1 – Division I.** *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".*

   i. According to the idea of the company to earn the maximum amount of pufosos, the best algorithm would be the second, as the first could be good, in the case the elements are sorted, in some order close to the descending one, and the third minimizes the length of the segments. Which is the opposite thing that the first does. As the first length, from 0 to whichever value, would be the maximum, then the next length will range from such a value to the next, but with the second maximum length, and so on.

b. **Task 2 – Division I.** *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".*

   i. According to finding the optimal solution for minimizing the number of pufosos, the answer is clear: the third algorithm, the one that sorts the lengths in ascending order, as it minimizes the steps between segments.

c. **Task 3 – Division I.** *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.*

   i. *Greedy 1:* each operation of this method: creating the pairs, then sorting, process which is not performed in this case, and calculating the pufosos, are done sequentially. This means, the complexity of the process – among all – with the highest time complexity, will be the one of the whole algorithm. Creating the pairs has a complexity of $O(n)$, as we just iterate n times, the size of the array of lengths. And we add a new Pair to the Linked List, which's complexity is constant. That's one of the reasons of choosing such collection. And the complexity of the *calculatePufosos()* method is just linear, as we iterate over the Linked List whose size is n, so n times. And the complexity of the nested code is constant. And so, the time complexity of the algorithm is $O(n)$.

   ii. *Greedy 2:* following the same reasoning as before, the only thing that changes now is the sorting part, which now has an impact on the final time complexity as we perform the operation: now we sort the list, in ascending order. Whose time complexity is $O(n \cdot \log n)$, as it is a modified version of merge sort. And so, the time complexity of the algorithm is $O(n \cdot \log n)$.

   iii. *Greedy 3:* Notice that the time complexity is the same as in the previous case: $O(n \cdot \log n)$. As this is essentially the same algorithm but with the elements sorted in the opposite direction.

d. **Task 4 – Division I.** *Explain if the times obtained in the table are in tune or not, with the complexities set out in the previous section.*

   i. Even if 1 value in the first column: the last, isn't quite good, that could be caused by the influence of background processors, or any other thing, I'm happy with the results as they stick quite good to the expected values. In order me to verify that, I just created the following tables:

tGreedy1 $(10^{-6}\ s)$	*tGreedy2* $(10^{-6}\ s)$	*tGreedy3* $(10^{-6}\ s)$
$t_1 = 11{,}650;\ n_1 = 3{,}200\ and\ n_2 = 6{,}400$ $t_3 = 21{,}242;\ n_3 = 6{,}400\ and\ n_4 = 12{,}800$ • $t_2 = \frac{6{,}400}{3{,}200} \cdot 11{,}650 = 23{,}300$ • $t_4 = \frac{12{,}800}{6{,}400} \cdot 21{,}242 = 42{,}484$	$t_1 = 67{,}938;\ n_1 = 3{,}200\ and\ n_2 = 6{,}400$ $t_3 = 150{,}358;\ n_3 = 6{,}400\ and\ n_4 = 12{,}800$ • $t_2 = 2 \cdot \frac{\log_2 2 + \log_2 3{,}200}{\log_2 3{,}200} \cdot 67{,}938 = 147{,}545$ • $t_4 = 2 \cdot \frac{\log_2 2 + \log_2 6{,}400}{\log_2 6{,}400} \cdot 150{,}358 = 324{,}450$	$t_1 = 36{,}707;\ n_1 = 1{,}600\ and\ n_2 = 6{,}400$ $t_3 = 62{,}712;\ n_3 = 3{,}200\ and\ n_4 = 12{,}800$ • $t_2 = 4 \cdot \frac{\log_2 4 + \log_2 1{,}600}{\log_2 1{,}600} \cdot 36{,}707 = 174{,}417$ • $t_4 = 4 \cdot \frac{\log_2 4 + \log_2 3{,}200}{\log_2 3{,}200} \cdot 62{,}712 = 293{,}934$

Values	*tGreedy1*	*tGreedy2*	*tGreedy3*
Actual$_1$	21,242	150,358	144,174
Expected$_1$	23,300	147,545	174,417
Actual$_2$	34,213	305,253	310,570
Expected$_2$	42,484	324,450	293,934