Activity 1. Execution times

n	tGreedy1	tGreedy2	tGreedy3
100	72	54	52
200	49	11	10
400	75	46	44
800	110	147	159
1600	201	589	584
3200	350	2558	2395
6400	609	14148	13956
12800	1100	83162	78243
25600	2370	357059	385534

***For greedy1 as it has a complexity O(n) the values weren't relevant until I used nTimes=`40000`**

Activity 2. Answer the following questions.

1-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 think that the approach used in greedy2 , which consists on sorting the segments from the largest to the smallest (Descendant order) is the best solution if the greatest cost is desired.

Since the segments will have larger sizes at the beginning the midpoints will be greater, but also , those smaller segments that would have small midpoints if placed first, will have their costs enlarged too .

For example:

Algorithmics	Student information	Date	Number of session
	UO:269546	22-03-21	4
	Surname: Fernández Arias		
	Name: Sara		

Having the following segment sizes [2,90,12,7]

If sorted in Descendant order , the vector would be [90,12,7,2] then :

(0,90) -> midpoint =45.

(90,(90+12))=(90,102)-> midpoint =96.

(102,(102+7))=(102,109) -> midpoint =105,5.

(109,(109+2))=(109,111)->midpoint =110.

Instead , if we ordered them in ascendant order :

(0,2)-> midpoint=1;

(2,7)-> midpoint=4.5.

(7,19) midpoint=22.5

(19,109)-> midpoint=64.

The values are reduced drastically, while the size of the intervals is enlarged. In descendant order, the size of the intervals is shortened, so the values grow quicker.

2-Explain if any of the greedy algorithms involves the optimal solution from the point of view of the player, who is interested in minimizing the number of “pufosos”.

Following the previous explanation , the best approach for clients would be greedy3. That is because the ordering of the segment sizes is done in ascendant order. Then , the values of the midpoints of the intervals are significantly lower, and so , the cost in pufosos will be lower too.

3-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`.

```
public long greedy1() {
    int[] nonSorted = copyToArray(segments);//O(n)
    //printSolution(nonSorted);
    return computeCost(nonSorted); //O(n)
}
```

The total complexity will be O(n) after applying asymptotic notation properties; provided that my implementation of the methods copyToArray() and computeCost() are:

<table>
<tr><td rowspan="4">Algorithmics</td><td>Student information</td><td>Date</td><td>Number of session</td></tr>
<tr><td>UO:269546</td><td>22-03-21</td><td>4</td></tr>
<tr><td>Surname: Fernández Arias</td><td></td><td></td></tr>
<tr><td>Name: Sara</td><td></td><td></td></tr>
</table>

```java
private long computeCost(int[] arrayOfSegments) {
    int start = 0;
    long cost = 0;
    int end;
    for (int si = 0; si < arrayOfSegments.length - 1; si++) {
        end = start + arrayOfSegments[si];
        cost += (start + end) / 2;
        start = end;
    }
    return cost;
}
```

```java
*/
private int[] copyToArray(List<Integer> list) {
    int[] toReturn = new int[list.size()];
    for (int i = 0; i < toReturn.length; i++) {
        toReturn[i] = list.get(i);
    }
    return toReturn;
}
```

Also , for greedy2 and greedy3, the complexities are cuadratic since:

```java
public long greedy2() {

    int[] sorted = sortDescendantOrder();
//  printSolution(sorted);
    return computeCost(sorted);
}

public long greedy3() {
    int[] sorted = sortAscendantOrder();
//  printSolution(sorted);
    return computeCost(sorted);
}
```

Since both of my sorting implementations are quadratic (two nested loops):

```java
private int[] sortAscendantOrder() {
    List<Integer> sorted = new ArrayList<Integer>(segments);// Copy constructor

    for (int i = 0; i < sorted.size(); i++) {//O(n)
        for (int j = i + 1; j < sorted.size(); j++) {//O(n)
            if (sorted.get(i) > sorted.get(j)) {
                Collections.swap(sorted, i, j); //O(1)

            }
        }

    }

    return copyToArray(sorted);

}

private int[] sortDescendantOrder() {
    List<Integer> sorted = new ArrayList<Integer>(segments);// Copy constructor

    for (int i = 0; i < sorted.size(); i++) {   //O(n)
        for (int j = i + 1; j < sorted.size(); j++) {//O(n)
            if (sorted.get(i) < sorted.get(j)) {
                Collections.swap(sorted, i, j); //O(1)

            }
        }

    }
```

Applying the asymptotic notation once again to O(n)+$\underline{O(n^2)}$, the total complexity is $O(n^2)$.

Algorithmics	Student information	Date	Number of session
	UO:269546	22-03-21	4
	Surname: Fernández Arias		
	Name: Sara		

4-Explain if the times obtained in the table are in tune or not, with the complexities set out in the previous section.

For *greedy1* , provided that it has a complexity of O(n):

Being t1= 110 n1=800 n2=1600 , t2 will be:

Provided that k=n2/n1=1600=800=2 and T2=$\frac{f(n2)}{f(n1)} xT1$,

T2=$\frac{n_2}{n_1} * T1 = K * T1 = 2 * 110 = 220\ ms.$

I obtained t2=201ms empirically, so it can be said that the result matches the complexity.

For *greedy2* , provided that it has a complexity of $O(n^2)$:

Being t1= 147 n1=800 n2=1600 , t2 will be:

Provided that k=n2/n1=1600=800=2 and T2=$\frac{f(n2)}{f(n1)} xT1$,

T2=$\frac{{n_2}^2}{{n_1}^2} * T1 = k^2 * T1 = 4 * 147 = 588\ ms.$

Empirically , I obtained t2=589ms , then it can be said that the result matches the complexity.

For *greedy3* , provided that it has a complexity of $O(n^2)$:

Being t1= 159 n1=800 n2=1600 , t2 will be:

Provided that k=n2/n1=1600=800=2 and T2=$\frac{f(n2)}{f(n1)} xT1$,

T2=$\frac{{n_2}^2}{{n_1}^2} * T1 = k^2 * T1 = 4 * 159 = 636\ ms.$

Empirically , I obtained t2=584ms , then it can be said that the result does not match the complexity.