ALGORITHMICS - Bachelor of Software Engineering, 2024

LAB GUIDE. SESSION 2

GOALS:

- **Sorting algorithms and their comparative study**

1. Introduction

In this lab, we address the ordering problem, that is, given **n** elements in any order, apply an algorithm to order them by a predetermined key. The elements can be of any type. Here, they will be of integer type, but the same algorithms would be used to order floats, strings or objects of any type on which there is an order relationship that allows us to know whether a given object is less than, greater than or equal to any other.

When there is a certain volume of data, sorting it makes access faster to do any search (or other basic operations) on it. Therefore, sorting is undoubtedly one of the operations that is carried out most frequently in any information processing and hence its interest.

This is one of the problems that has more algorithms capable of solving it although, in this session, we are going to focus only on four of them.

In this lab, we will take times without the optimizer (WITHOUT OPTIMIZATION) because although we know times will be higher, we will avoid surprises that happen when the JIT optimizer acts since times may not correspond to the time complexity of the different algorithms.

In the tables of times that you will make, if any time is longer than 1 minute we can indicate Out of Time (OoT) and if it is less than 50 milliseconds we can indicate Lack of Reliability (LoR), not being necessary to repeat the process to be measured several times (10, 100, ...), as seen in a previous session, in which it was necessary to proceed with times below 50 milliseconds.

2. Direct exchange or Bubble algorithm

The **Vector.java** class has some basic operations that allow generating the following: an ordered vector, a vector in reverse order, a vector in random order, and printing them all in the console.

Likewise, in the **Bubble.java** class the Bubble sorting algorithm is implemented and tested. It is necessary to study its operation on paper for some example cases and then analyze its time complexity for the 3 cases: vector already ordered, vector in reverse order and vector in random order.

Finally, in the **BubbleTimes.java** class, the size of the problem is increased, and the times of that algorithm are calculated for the three assumptions seen.

YOU ARE REQUESTED TO:

After measuring times, fill in the table:

TABLE 1 = BUBBLE ALGORITHM

(times in milliseconds and WITHOUT OPTIMIZATION):

We can type "OoT" for times over one minute and "LoR" for times less than 50 milliseconds.

n	*t ordered*	*t reverse*	*t random*
10000			
2*10000			
2**2*10000			
2**3*10000			
2**4*10000			

Explain whether the different times obtained agree with what is expected, according to the time complexity studied.

3. Selection algorithm

In the **Selection.java** class the Selection sorting algorithm is implemented and tested. It is necessary to study its operation on paper for some example cases and then analyze its time complexity for the 3 cases: vector already ordered, vector in reverse order and vector in random order.

YOU ARE REQUESTED TO:

Implement a **SelectionTimes.java** class to help you to fill in the following table:

TABLE 2 = SELECTION ALGORITHM

(times in milliseconds and WITHOUT OPTIMIZATION):

We can type "OoT" for times over one minute and "LoR" for times less than 50 milliseconds.

n	*t ordered*	*t reverse*	*t random*
10000			
2*10000			
2**2*10000			
2**3*10000			
2**4*10000			

Explain whether the different times obtained agree with what is expected, according to the time complexity studied.

4. Insertion algorithm

In the **Insertion.java** class the Insertion sorting algorithm is implemented and tested. It is necessary to study its operation on paper for some example cases and then analyze its time complexity for the 3 cases: vector already ordered, vector in reverse order and vector in random order.

YOU ARE REQUESTED TO:

Implement an **InsertionTimes.java** class to help you to fill in the following table:

TABLE 3 = INSERTION ALGORITHM

(times in milliseconds and WITHOUT OPTIMIZATION):

We can type "OoT" for times over one minute and "LoR" for times less than 50 milliseconds.

n	*t ordered*	*t reverse*	*t random*
10000			
2*10000			
2**2*10000			
2**3*10000			
2**4*10000			
2**5*10000			
….			
2**13*10000			

Explain whether the different times obtained agree with what is expected, according to the time complexity studied.

5. Quicksort algorithm

In the **Quicksort.java** class the Quicksort sorting algorithm is implemented and tested (this great algorithm was invented by C.A.R. Hoare in 1960). It is necessary to study its operation on paper for some example cases and then analyze its time complexity for the 3 cases: vector already ordered, vector in reverse order and vector in random order.

YOU ARE REQUESTED TO:

Implement a **QuicksortTimes.java** class to help you to fill in the following table:

TABLE 4 = QUICKSORT ALGORITHM

(times in milliseconds and WITHOUT OPTIMIZATION):

We can type "OoT" for times over one minute and "LoR" for times less than 50 milliseconds.

n	*t ordered*	*t reverse*	*t random*
250000			
2*250000			
2**2*250000			
2**3*250000			
2**4*250000			
2**5*250000			
2**6*250000			

Explain whether the different times obtained agree with what is expected, according to the time complexity studied.

After seeing how long it takes to sort 16 million items initially in random order, calculate and compare (from the complexities and data in the tables above), how many days would each of those three methods (Bubble, Selection and Insertion) take in doing the same?

6. Quicksort + Insertion algorithm

Despite the excellent times we got in the previous section, we are going to try to improve them (at least, we will try). The idea is simple: in the default Quicksort algorithm implementation we call itself recursively until we reach subvectors of size zero (stop case).

It is proposed that when the sizes of these subvectors are less than or equal to a parameter **k**, we should call the Insertion algorithm instead (this idea is based on the fact that these subvectors may already be quite ordered and the insertion works well in that case).

YOU ARE REQUESTED TO:

Reimplement the Insertion sorting algorithm, so that instead of sorting the vector passed to it in all its positions [0 .. length-1], it should work only between the positions [left .. right] that will be passed as parameters.

Implement a **QuicksortInsertion.java** class that orders the elements of the vector as noted above.

It is proposed to take times that, although not conclusive and decisive for comparing them, will be indicative. To do this, implement a class **QuicksortInsertionTimes.java** that will take times (for a size of n=16 million elements initially generated randomly) to fill in the following table:

TABLE 5 = QUICKSORT + INSERTION ALGORITHM (n=16 M and random)

(times in milliseconds and WITHOUT OPTIMIZATION):

We can type "OoT" for times over one minute and "LoR" for times less than 50 milliseconds.

n	*t random*
Quicksort	
Quicksort+Insertion (k=5)	
Quicksort+Insertion (k=10)	
Quicksort+Insertion (k=20)	
Quicksort+Insertion (k=30)	
Quicksort+Insertion (k=50)	
Quicksort+Insertion (k=100)	
Quicksort+Insertion (k=200)	
Quicksort+Insertion (k=500)	
Quicksort+Insertion (k=1000)	

Explain conclusions obtained from the previous table.

7. Work to be done

- An `algstudent.s2` **package** in your course project. The content of the package should be the Java files used and created during this session.
- A `session2.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 deadline is one day before the next lab session of your group.