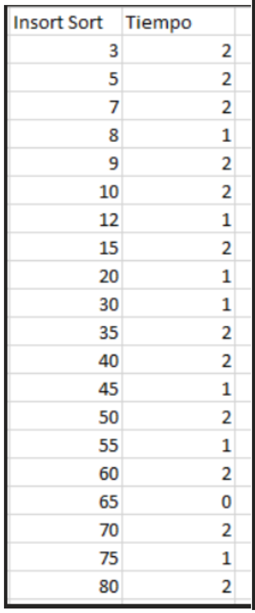
Laboratory practice No. X: Complexity

Isabella Pérez Serna
Universidad Eafit
Medellín, Colombia
iperezs2@eafit.edu.co

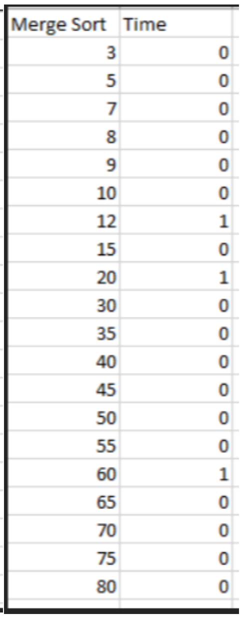
Mariana Quintero Montoya
Universidad Eafit
Medellín, Colombia
mquintero3@eafit.edu.co

3) Practice for final project defense presentation

3.1

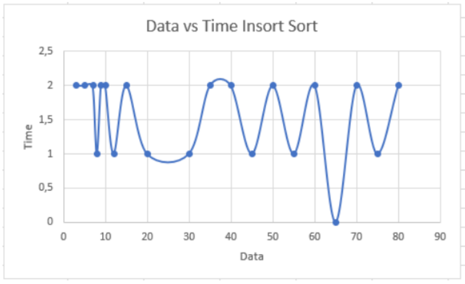


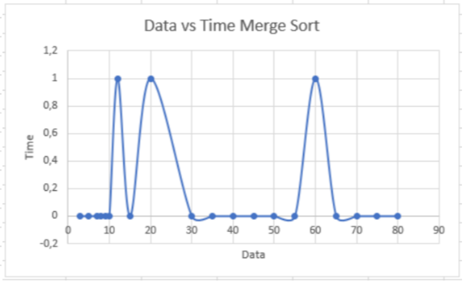
Insort Sort	Tiempo
3	2
5	2
7	2
8	1
9	2
10	2
12	1
15	2
20	1
30	1
35	2
40	2
45	1
50	2
55	1
60	2
65	0
70	2
75	1
80	2



Merge Sort	Time
3	0
5	0
7	0
8	0
9	0
10	0
12	1
15	0
20	1
30	0
35	0
40	0
45	0
50	0
55	0
60	1
65	0
70	0
75	0
80	0

3.2





Insertion Sort: According to the experimental information of the table where the volume of data varied from 3 to 80, and the execution time had a high increase, but remained in a range of 0 to 2 milliseconds can be stated that with this algorithm if it is possible to sort large volumes of data and the execution time will have a large increase.

Merge Sort: According to the experimental information of the table where the volume of data varied from 3 to 80, and the execution time did not have a high increase, but remained in a range of 0 to 1 milliseconds it can be stated that with this algorithm if it is possible to sort large volumes of data and the execution time will not have a large increase, it will be proportional; it is more appropriate to use it when they are large amounts unlike the Insertion Sort.

3.3
Due to its complexity and considering that Merge sort has less complexity, it is not suitable for using Insertion Sort in jobs involving large amounts of data. For a set of 60 million elements, the algorithm will take approximately 1 hour or more to sort through all the elements.

3.4 The algorithm in whose complexity a logarithm appears is in the *MergeSort* and this is because the size of the problem, that is, the length of the array, is divided in this algorithm, that division action is what causes a logarithm in its complexity.

3.5 [opc] The Insert Sort and Merge Sort perform the same function, sorting an arrangement from minor to major, but in different ways, the Insert Sort compares element by element of the arrangement of numbers, and the Merge Sort is based on the "divide and conquer", this divides as much as possible the arrangement in pairs and returns the order. Now, in order for Inserton Sort to be faster than Merge Sort, the data in this array must be in order, so Insert Sort will not make any changes and will only quickly compare the elements, while Merge Sort will do its normal process of dividing and returning the order.

3.4 [opc] The code begins by analyzing the length, if this is less than or equal to 1 returns the length as requested by the statement.Then the counter is initialized to 1, this is the one that will contain the length of the largest interval found, the first For takes the element in the position i that begins at 0 and runs the array when the second cycle ends its function, then the second For takes the elements in the position j that begins at i + 1 and runs the array, within this cycle is the first If that compares the elements i and j, if these two elements of the array are equal then the second If gives a new value to the counter only if the length of the spaces between i and j are greater than the counter at that time; finally, when i goes through the whole array, except the last position, the value of the counter is returned, that is, the length of the longest interval.

3.5 The variables n and m can be understood as those that represent the extension, length or size of the program that is performed, I mean, on them depends if the complexity of the algorithm that is being implemented increases or decreases.

3.6
sum13

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

T(n) = c1, c1= 3
T(n) = c2 + T(n+2), c2= 4
T(n) = c3 + T(n+1), c3= 3

fizzArray
T(n) = n

notAlone
T(n) = n
T(n) = c1 * n, c1 = 4
T(n) = c2 * (c1 * n), c2 = 6
T(n) = c3 * (c1 * n), c3 = 0
T(n) = c4 * n, c4 = 0
T(n) = c5, c5= 1

lucky13
T(n) = n
T(n) = c1 * n, c1 = 5
T(n) = c2, c2 = 1

countEvens
T(n) = n
T(n) = c1 * n, c1 = 3
T(n) = c2, c2 = 1

countClumps
T(n) = n
T(n) = c1 * n, c1 = 3
T(n) = c2 * (c1 * n), c2 = 5
T(n) = c3, c3 = 1

canBalance
T(n) = n
T(n) = n
T(n) = c1 * n, c1 = 3
T(n) = c2, c2 = 1

linearIn
T(n) = n
T(n) = c1 * n, c1 = 4
T(n) = c2, c2 = 3
T(n) = c3, c3 = 1

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

seriesUp
T(n) = n
T(n) = n * n
T(n) = c1, c1 = 1

maxSpan
T(n) = c1, c1 = 3
T(n) = n
T(n) = n * n
T(n) = c1 * (n * n), c1 = 2
T(n) = c2 * (c1 * (n * n)), c2 = 4
T(n) = c3, c3 = 1

4) Practice for midterms

4.1 C
4.2 D
4.3 [opc] B
4.4 [opc] B
4.5 A
4.6 100 Segundos
4.7 Todas las anteriores
4.8 [opc] A
4.9 A
4.10 [opc] C
4.11 [opc] C
4.12 [opc] B
4.13 [opc] C
4.14 A-C

5) Recommended reading (optional)

Complexity of the algorithms and lower levels of problems:

Time complexity of an algorithm

It is often said that an algorithm is good if it takes little time to run and requires little memory space. However, by tradition, a more important factor to determine the effectiveness of an algorithm is the time needed to run it. To measure the temporal complexity of an algorithm, it is tempting to write a program for this algorithm and see how fast it runs. This is not appropriate because there are many factors unrelated to the algorithm that affect the performance of the program.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

In algorithm analysis, you will always choose a particular type of operation that occurs in the algorithm, and perform a mathematical analysis to determine the number of operations needed to complete the algorithm. According to such a circumstance, it seems that data exchange, rather than comparison, should be used to measure the temporal complexity of this particular ordering algorithm. It is often said that the cost of running an algorithm depends on the size of the problem. Suppose that steps are required to execute an algorithm.

It would often be said that the temporal complexity of this algorithm is of the order of n3.

Definition: f(n)= O (g(n)) if and only if there are two positive constants c and n0 such that |f(n)| < c|g(n)| for all > n0.
Suppose you have two algorithms A1 and A2 that solve the same problem. A common mistake is to think that the program for A2 will always run faster than the program for A1. In other words, even if the number of steps needed to run A2 is less than those required for A1, in some cases A1 runs faster than A2. The reader now understands the importance of the constant that appears in the definition of the function O g.

However, it does not influence how big the constant is, its importance decreases as n grows. Another point to remember is that it is always possible, at least theoretically, to code any algorithm in hardware. That is, it is always possible to design a custom circuit to implement an algorithm. If two algorithms are implemented in hardware, the time needed to run a step of one of the algorithms can be equated to the time required in the other algorithm.

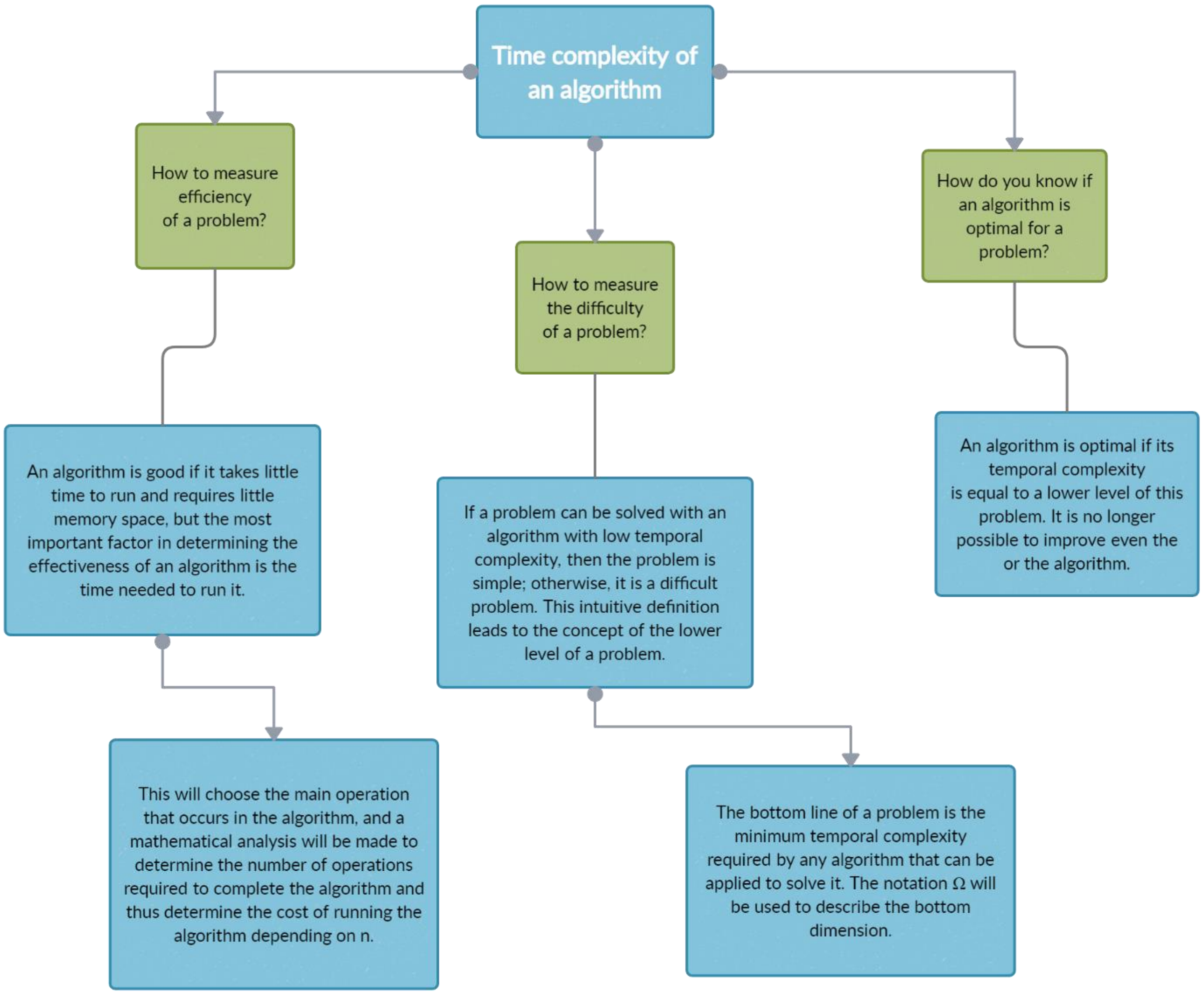
If the temporal complexities of A1 and A2 are O and O , respectively, then it is known that A2 is better than A1 if both are implemented in hardware. Of course, the above analysis makes sense only if the ability to implement the algorithms in hardware is perfectly mastered. The importance of the order of magnitude can be seen by studying Table 2-1.
Usually the best case analysis is the easiest, the worst case analysis is the second easiest, and the most difficult is the average case analysis. In fact, there are still many open problems involving the average case analysis. If xi is the largest of all the i numbers, then the inner cycle is not executed and within this inner cycle there is no data movement at all. If xi is the second largest of all i numbers, there will be a data exchange, and so on.

The binary search algorithm:
The binary search is a famous search algorithm. After ordering a number set in an increasing or decreasing sequence, the binary search algorithm starts from the middle of the sequence. If the test point is equal to the midpoint of the sequence, the algorithm ends; otherwise, depending on the result of comparing the test element and the center point of the sequence, the search is recurrently performed to the left or to the right of the sequence.

The bottom line of a problem:
In the previous section, many examples of algorithms to find temporal complexities taught us how to determine the efficiency of an algorithm. In this section, the problem of temporal complexity will be addressed from a completely different point of view. Consider the problem of finding ranges, or the problem of the traveling agent. The question is: how to measure the difficulty of a problem? This question can be answered in a very intuitive way. If a problem can be solved with an algorithm with low temporal complexity, then the problem is simple; otherwise,

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

it is a difficult problem. This intuitive definition leads to the concept of the lower level of a problem.


Time complexity of an algorithm
How to measure efficiency of a problem?
How to measure the difficulty of a problem?
How do you know if an algorithm is optimal for a problem?
An algorithm is good if it takes little time to run and requires little memory space, but the most important factor in determining the effectiveness of an algorithm is the time needed to run it.
If a problem can be solved with an algorithm with low temporal complexity, then the problem is simple; otherwise, it is a difficult problem. This intuitive definition leads to the concept of the lower level of a problem.
An algorithm is optimal if its temporal complexity is equal to a lower level of this problem. It is no longer possible to improve even the or the algorithm.
This will choose the main operation that occurs in the algorithm, and a mathematical analysis will be made to determine the number of operations required to complete the algorithm and thus determine the cost of running the algorithm depending on n.
The bottom line of a problem is the minimum temporal complexity required by any algorithm that can be applied to solve it. The notation Ω will be used to describe the bottom dimension.

References:

Barnes, H (2019) Platzi/Cuerso Básico de Algoritmos (Version 1.0) [Source code]. https://platzi.com/tutoriales/1469-algoritmos/4260-merge-sort-en-java/

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473