Laboratory practice No. 2: Big O notation

Pablo Alberto Osorio Marulanda
Universidad Eafit
Medellín, Colombia
paosoriom@eafit.edu.co

Verónica Mendoza Iguarán
Universidad Eafit
Medellín, Colombia
vmendozai@eafit.edu.co

3) Practice for final project defense presentation
3.1)

Time's table for Insertion sort algorithm

n	Time
0	0
1	0
2	1001
3	3001
4	6001
5	10004
6	15007
7	21009
8	28011
9	36013
10	45025

11	**55022**
12	**66030**
13	**78030**
14	**91042**
15	**105042**
16	**120047**
17	**136056**
18	**153066**
19	**171094**
20	**190085**

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

Time's table for Merge sort algorithm

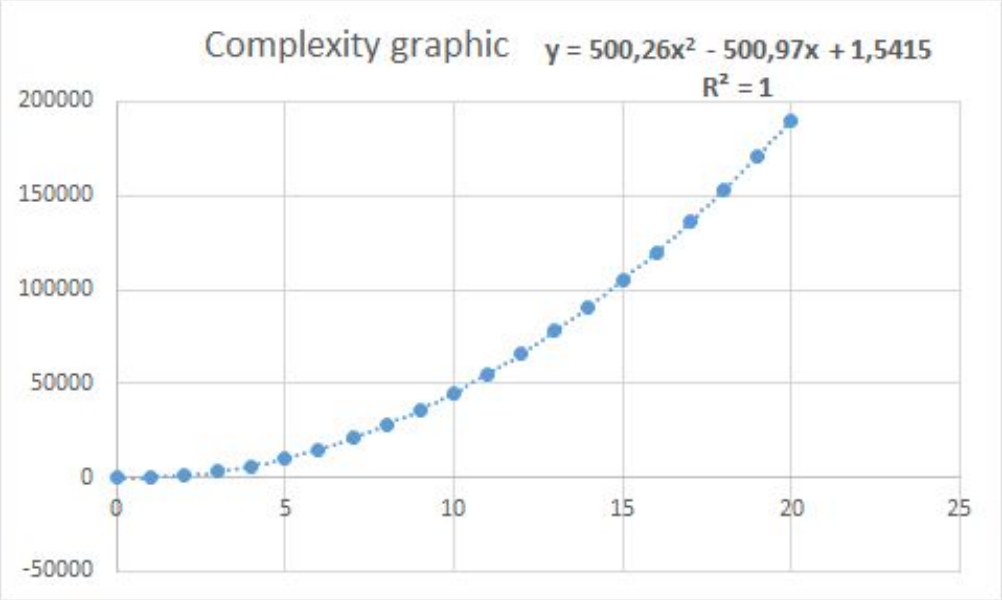
n	time
0	0
1	0
2	1002
3	3001
4	4001
5	7003
6	9003
7	11003
8	12005
9	16003
10	19004
11	22006
12	24005
13	27007
14	29005
15	31007
16	32006
17	37008
18	41011

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

19	**45010**
20	**48011**

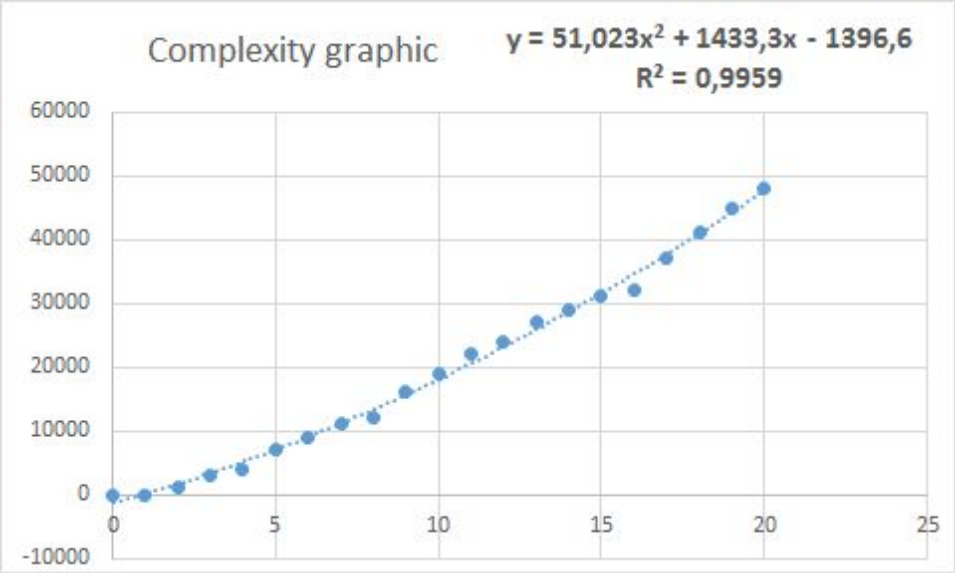
3.2)

Size-Time graphic for insertion sort algorithm [(n)(ms)]



PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

Size-Time graphic for Merge sort algorithm [(n)(ms)]



3.3)

If you want to sort a really big array, you must use the Merge Sort algorithm. Why? The Merge Sort algorithm have a complexity equal to O(n logn), while the insertion sort algorithm have a comexity equal to O(n^2). But if we did not have the last information, we could know that just seeing the graphics size-time in the previous point. In the insertion sort graphic the executing time grows faster than the MergeSort graphic executing time. So, in order to this, it is not appropriate to use the insertion sort algorithm to sort a big data base (A data base with so many elements)

3.4)

Initially it is examined if the array is zero size, if so, zero is returned. If not, the algorithm continues. There is a first cycle that is responsible for taking the first position of the array, the second cycle takes the last position, these two are compared, if they are equal, the variable **cont** is responsible for counting how many positions separate the first from the last, it means, the **span** between the two positions that have equal values. Then it is examined if the **span** that has just been calculated is the greatest among all the possible ones that can be given in the array, so that there is a variable **max** that stores it; this is done during the second cycle because this is where the distances between the equal values are measured; if the **span** that has just been calculated is greater than the variable **max**, then **max** is updated and takes the value of the calculated **span**. In this way, at the end of the algorithm, the variable **max** is left with the largest **span**.

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

3.5)

Array-2

countEvens

T(n)= c_1+c_2+(c_3*(n+1)+c_4*n+c_5+c_6
T(n) is O(c_1+c_2+(c_3*(n+1)+c_4*n+c_5+c_6) By big-O definition
T(n) is O(c_3*(n+1)+c_4*n) By rule of the sum
T(n) is O(n+1+n) by rule of the product
T(n) is O(n) By rule of the sum

more14

T(n)=c_1+c_2+c_3+(c_4*(n+1)+c_5*n+c_6+c_7*n+c_8+c_9
T(n) is O(c_1+c_2+c_3+(c_4*(n+1)+c_5*n+c_6+c_7*n+c_8+c_9) By Big-O definition
T(n) is O(c_4*(n+1)+c_5*n+c_6+c_7*n) By rule of the sum
T(n) is O(c_4*(n+1)) By rule of the sum
T(n) is O(n+1) By rule of the product
T(n) is O(n+1) By rule of the sum

sum13

T(n)=c_1+c_2+c_3+c_4+(c_5*(n+1)+c_6*n+c_7+c_8*n+c_9+c_10+c_11*n+c_12+c_13
T(n) is O(c_1+c_2+c_3+c_4+(c_5*(n+1)+c_6*n+c_7+c_8*n+c_9+c_10+c_11*n+c_12+c_13) By Big-O definition
T(n) is O(c_5*(n+1)+c_6*n+c_8*n+c_11*n) By rule of the sum
T(n) is O(c_5*(n+1)) By the rule of sum
T(n) is O(n+1) By rule of the product
T(n) is O(n) By rule of the sum

s**ameEnds**

T(n)=c_1+c_2+c_3+(c_4*(n+1))+c_5+(c_6*n)+c_7+(c_8*n)+c_9
T(n) is O(c_1+c_2+c_3+(c_4*(n+1))+c_5+(c_6*n)+c_7+(c_8*n)+c_9) By Big-O Definition
T(n) is O(c_4*(n+1)+(c_6*n)+(c_8*n)) By rule of the sum
T(n) is O(c_4*(n+1)) By rule of the sum
T(n) is (n+1) By the rule of the product
T(n) is O(n) By the rule of the sum

fizzBuzz

T(n)=c_1+c_2+c_3+(c_4*(n+1))+c_5+c_6*n+c_7*n+c_8*n+c_9+c_10*n+c_11*n+c_12*n+c_13+c_14*n+c_15*n+c_16*n+c_18*n+c_19*n+c_20*n+c_21
T(n) is O(c_1+c_2+c_3+(c_4*(n+1))+c_5+c_6*n+c_7*n+c_8*n+c_9+c_10*n+c_11*n+c_12*n+c_13+c_14*n+c_15*n+c_16*n+c_18*n+c_19*n+c_20*n+c_21) By big-O definition
T(n) is O(c_4*(n+1)) By the rule of the sum
T(n) is O(n+1) By the rule of the products
T(n) is O(n) By the rule of the sum

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

Array-3
maxSpan:
T(n)= c_1+c_2*n+c_3*m*n
T(n) is O(c_1+c_2*n+c_3*m*n) by definition of O
T(n) is O(c_3*n*m) by rule of the sum
T(n,m) is O(n*m) by rule of the product

fix34:
T(n,m)= c_1+c_2*n+n+c_3*m*n
T(n,m) is O(c_1+c_2*n+n+c_3*m*n) by definition of O
T(n,m) is O(n*m) by rule of the sum
T(n,m) is O(n*m) by rule of the product

fix45:
T(n)= c_1+c_2*n+c_3*n*n
T(n) is O(c_1+c_2*n+c_3*n*n) by definition of O
T(n) is O(c_3*n^2) by rule of the sum
T(n) is O(n^2) by rule of the product

canBalance:
T(n,m)= c_1+c_2*n+c_3*m*n
T(n,m) is O(c_1+c_2*n+c_3*m*n) by definition of O
T(n,m) is O(c_3*n*m) by rule of the sum
T(n,m) is O(n*m) by rule of the product

linearIn:
T(n,m)= c_1+c_2*n+c_3*n*m
T(n,m) is O(c_1+c_2*n+c_3*n*m) by definition of O
T(n,m) is O(c_3*n*m) by rule of the sum
T(n,m) is O(n*m) by rule of the product

seriesUp:
T(n,m)= c_1+c_2*n+c_3*n*m
T(n,m) is O(c_1+c_2*n+c_3*n*m) by definition of O
T(n,m) is O(c_3*n*m) by rule of the sum
T(n,m) is O(n*m) by rule of the product

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

3.6)

Array-2

The variable n, for all the array-2 exercises wrote before this, means the same thing. So, the n variable represent the positions in the inicial array that must be visited (from left to right in all the cases), it means, the positions in the array that have not still been processed by the program.

Array-3

maxSpan: n represents the positions in the array that must be visited (from left to right) and **m** represents the positions of the array to be visited (from right to left) except for the **n** positions already visited.

fix34: n represents the number of positions in the array that must be visited and **m** represents the positions of the array to be visited except two, the position of number **3** and **4**, since these two are already ordered.

fix45: n represents the number of positions in the array that are still to be processed, that is, the number of missing iterations.

canBalance: for this case, **n** represents the number of positions in the array that must be visited (from left to right), and **m** represents the positions of the array that will be visited (from right to left), except for the **n** positions already visited.

linearIn: the variable **n** represents the size of the **inner** array, that is, the number of positions that must be visited in the array. The variable **m** is the number of positions of the **outer** array.

seriesUp: for this case, **n** is the number that the user enters, where **n * (n + 1) / 2** represents the number of times the algorithm will be executed, this being the number of positions to fill in the new array. The variable **m** is (n + 1)/2

4) Practice for midterms

1. *c*
2. d
3. b
4. b
5. d
6. a
7.

7.1. T(n) = c_1 + c_2 + T(n-1)

7.2. O(n)

8. b
9. d
10. c
11. c
12. b
13. a

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

5) Recommended reading (optional)

5.1)

a) Complexity of the algorithms and lower bounds of the problems

b) Main ideas:

An important factor in the analysis of the algorithms is the execution **time**, this depends on the **size of the problem** (n), which by definition has constants, whom has an importance that decreases as n grows. At this way, the constants are important only for small values in the problem.

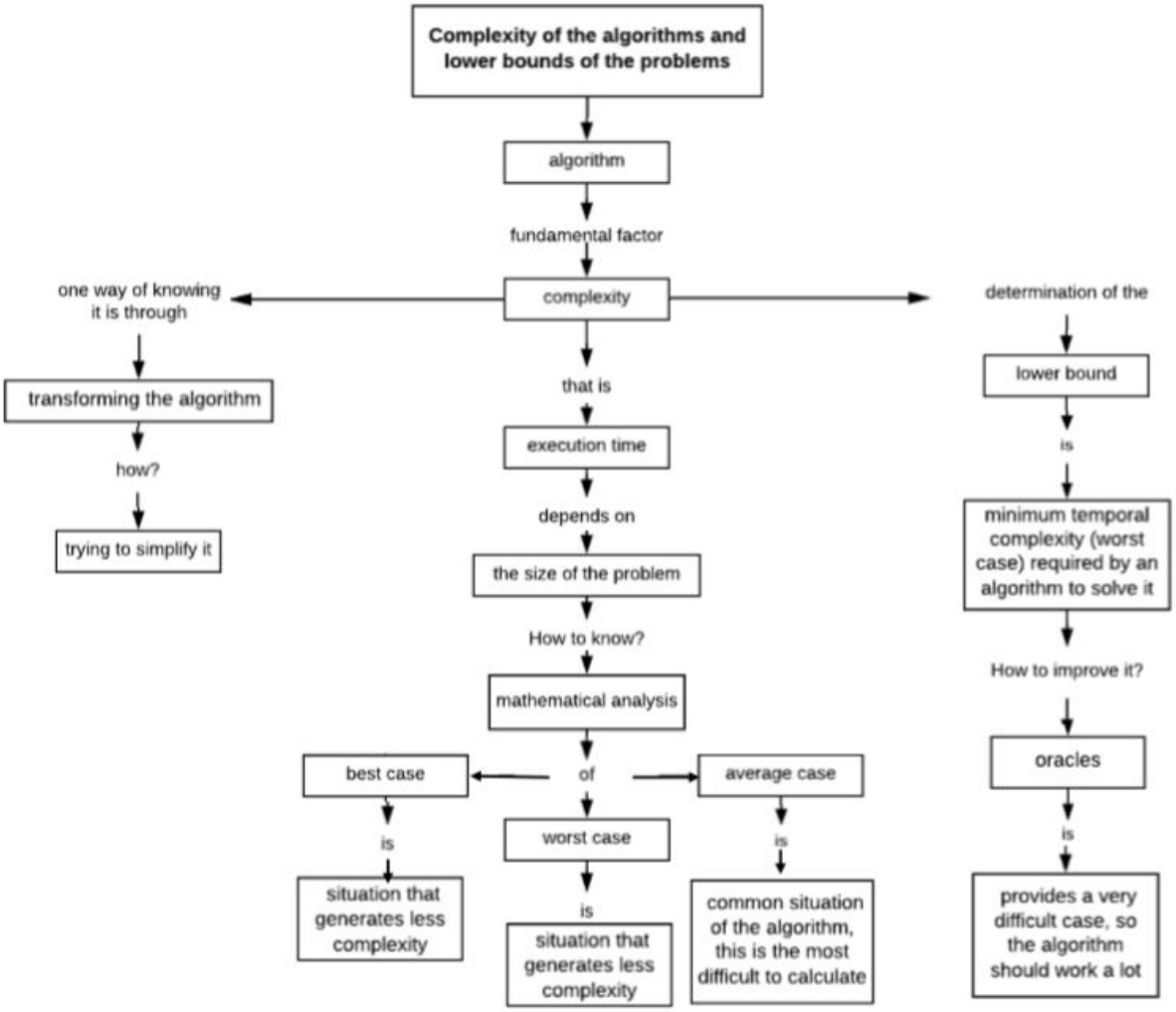
It is important to find an algorithm with less temporal **complexity**, because if the size of the problem increases, the algorithm will take a longer executing . To solve that problem, a mathematical analysis is carried out, in order to determine the number of operations necessary to complete the algorithm: analysis of the **best case** (situation that generates less complexity), analysis of the **worst case** (situation that generates a worse complexity) and analysis of the **average case** (common situation of the algorithm, this is the most difficult to calculate).

Another way to measure the difficulty of an algorithm is through the **lower bound**, that is, the minimum temporal complexity (worst case) required by an algorithm to solve it. Where the best lower bound is the highest.

One way to improve the height is through **oracles**. An oracle provides a very difficult case, so the algorithm should work a lot. Another way is transforming the algorithm.

c) Concept map:

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co



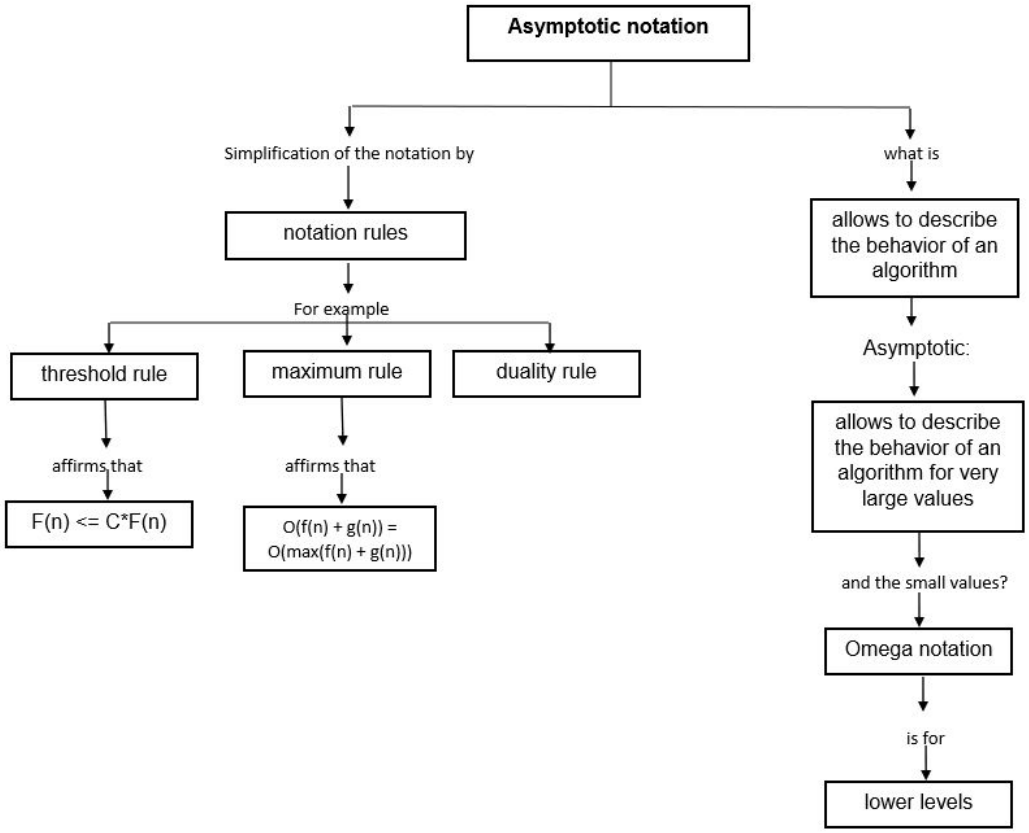
PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

5.2)

a) Asymptotic notation

b) Main ideas:
This notation allows simplifications to be made when describing and analyzing an algorithm. It is asymptotic because it deals with the behavior of functions for very large values (in the limit). This does not work for small values. So the Omega notation has been designed, which serves for lower levels.
The order of a function is the set of all the functions greater than this; this represents the worst-case execution time, which can be simplified with rules such as: the threshold rule that states that f (n) <= cf (n) with c equal to an arbitrary constant, or the rule of the maximum that states that O (f (n) + g (n)) = O (max (f (n) + g (n)))

c) Concept map:



PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

6) Team work and gradual progress (optional)

Member	Date	Done	Doing	To do
Verónica	28/08/2018 Start time: 3pm	· Online exercises · solution developmentt o partial simulation exercises	· Online exercises developmen t	
Pablo	28/08/2018 Start time: 3pm	· Online exercises · solution development to partial simulation exercises	· Online exercises developmen t	· calculate complexity of online exercises
Verónica	30/08/2018 Start time: 4:35pm	· calculate complexity of online exercises		
Pablo	30/08/2018 Start time: 4:35pm	· implementati on of Insertion sort and Merge sort	· run-time tables for different sizes of problems	· graphs of the execution times for different sizes of the problems

Verónica	06/09/2018 Start time: 3:15pm	· Analysis and explanation of the complexity calculation variables · online exercises continuation		· explanation of the exercise of Array3 (maxspan)
Pablo	06/09/2018 Start time: 3:15pm	· analysis of Insertion sort and Merge sort · creation of graphics		· explanation of the exercise of Array3 (maxspan)
Verónica	08/09/2018 Start time: 1:00pm	· analysis of Insertion sort and Merge sort		
Pablo	08/09/2018 Start time: 1:00pm	· continuation of calculate complexity of online exercises		