Laboratory practice No. 2: Big O notation

Juliana Lalinde Velásquez
Universidad Eafit
Medellín, Colombia
jlalindev@eafit.edu.co

Isabel Urrego Gómez
Universidad Eafit
Medellín, Colombia
iurregog@eafit.edu.co

3) Practice for final project defense presentation

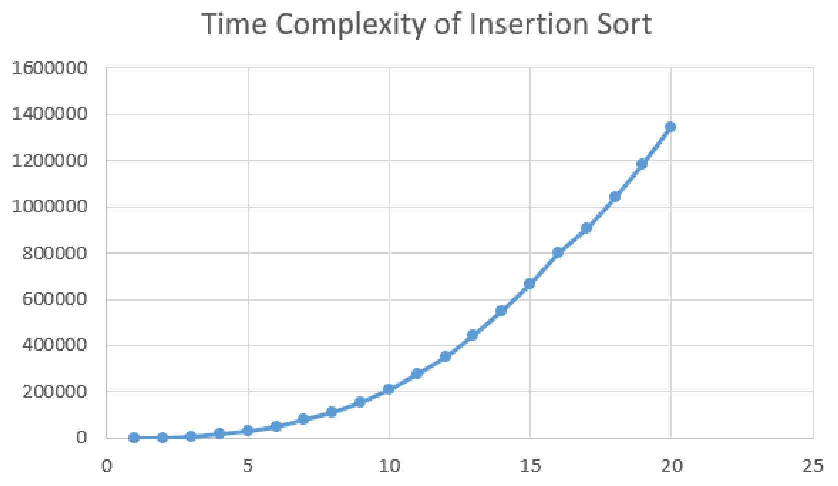
1. Insertion Sort

Size of the Array	Time
1	0
2	2002
3	7003
4	16007
5	30013
6	50025
7	77031
8	112054
9	156070
10	210089
11	275119
12	352169
13	442223
14	546233
15	665338
16	800387
17	903983
18	1039851
19	1185222
20	1340099

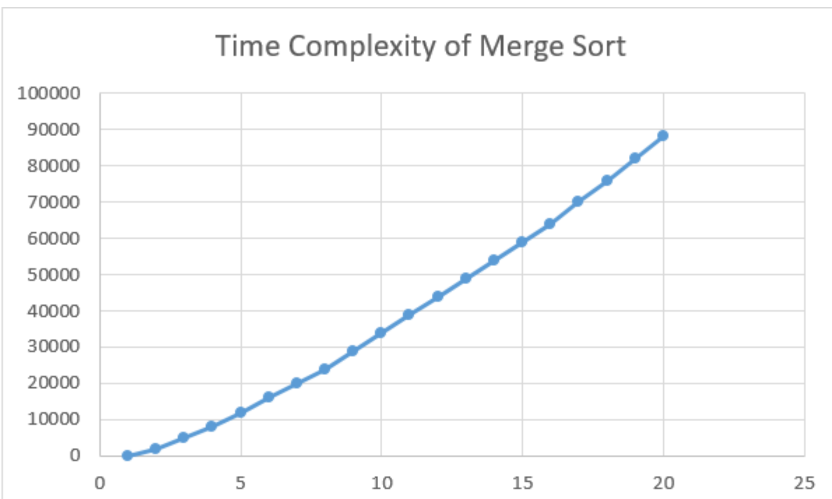
Merge Sort

Size of the Array	Time
1	0
2	2003
3	5002
4	8003
5	12004
6	16006
7	20009
8	24010
9	29014
10	34016
11	39020
12	44036
13	49018
14	54022
15	59047
16	64035
17	70030
18	76029
19	82040
20	88043

2. Graphics



Graphic 1: O(n^2)



Graphic 2: O(n log n)

3. From the graphics and the tables we see that both algorithms have different time complexities. At the beginning sorting small arrays result equally difficult for both algorithms, but as the size of the array gets bigger we notice that the time of the Insertion Sort algorithm grows much faster in comparison to the Merge Sort algorithm. Therefore, we can conclude that the second solution is more efficient when we must work with a million elements data base and working with Insertion Sort leads to a waste of time in those cases.
4. Our solution of the exercise maxSpan, from Codingbats Array-3 problems, works by using two cycles to find what is the biggest span (the number of elements between two

specific elements inclusive) between the leftmost and rightmost appearance of a number on an array given. The first cycle allows us to move one position on the array after we have used the second cycle to check what is that numbers span. In order to check what the span is we first need to check if the element on the position we are at is the same as the number in the main position (given by the first cycle), in case they are we calculate the span between them by subtracting x from y+1, this span value is saved under the name “comparar”. To continue the process, we then check if “comparar” is bigger than “maximo”, which is a integer that is replaced by the biggest span. After we check every position with the main position we end the second cycle and start the entire process again which a new main index. Once we have checked all of the array we return “maximo”.

5. Complexity

5.1. Array 2

_countEvens:

$$T(n) = \sum_{i=0}^{n} c_1$$

Solution:

$$T(n) = n$$

O(n)

_bigDiff:

$$T(n) = c_1 + \sum_{i=0}^{n} c_2$$

Solution:

$$T(n) = c_1 + n$$

O(n)

_centeredAverage:

$$T(n) = c_1 + \sum_{i=0}^{n} c_2$$

Solution:

$T(n) = c_1 + n$

O(n)

_sum13:

$$T(n) = \sum_{i=0}^{n} c_1$$

Solution:

$T(n) = n$

O(n)

_has22:

$$T(n) = \sum_{i=0}^{n} c_1$$

Solution:

$T(n) = n$

O(n)

5.2. Array 3

_maxSpan:

Complexity of the internal cycle:

$$\sum_{y=0}^{n} c_1 = n$$

$$T(n) = \sum_{x=0}^{n} n + c_1$$

Solution:

$$T(n) = n^2 + n$$

O(n^2)

__fix34:__

Complexity of the internal cycle:

$$\sum_{j=i+2}^{n} c_1 = n - i - 1$$

$$T(n) = \sum_{i=0}^{n} n - i - 1$$

Solution:

$$T(n) = \frac{1}{2}(n-2)(n+1)$$

O(n^2)

__fix54:__

Complexity of the internal cycle:

$$\sum_{y=0}^{n} c_1 = n$$

$$T(n) = \sum_{x=0}^{n} n + c_1$$

Solution:

$$T(n) = n^2 + n$$

O(n^2)

_linearIn:

$$T(n) = c_1 + \sum_{x=0}^{n} c_2$$

Solution:

$$T(n) = c_1 + n$$

O(n)

_seriesUp:

Complexity of the internal cycle:

$$\sum_{y=1}^{x} c_1 = x$$

$$T(n) = \sum_{x=1}^{n} x$$

Solution:

$$T(n) = \frac{n(n+1)}{2}$$

O(n^2)

6. Explanation of the variables

6.1. Array 2

- countEvens is O(n), where n is the number of elements in the array
- bigDiff is O(n), where n is the number of elements in the array
- centeredAverage is O(n), where n is the number of elements in the array
- sum13 is O(n), where n is the number of elements in the array
- has22 is O(n), where n is the number of elements in the array

6.2. Array 3

- maxSpan is O(n^2), where n is the number of elements in the array
- fix34 is O(n^2), where n is the number of elements in the array
- fix54 is O(n^2), where n is the number of elements in the array
- linearIn is O(n), where n is the number of elements in the array in the outer array

- seriesUp is O(n^2), where n is the final number that you want to be part of the array following a series pattern that consists of adding the numbers from 1 to n, starting with n=1 and adding a number every cycle until n equals the number n entered by the user.

4) Practice for midterms

1. c
2. d
3. b
4. b
5. d
6. a
7.
   7.1. T(n)= c+ T(n-1)
   7.2. O(n)
8. b
9. d
10. c
11. c
12. b
13. a

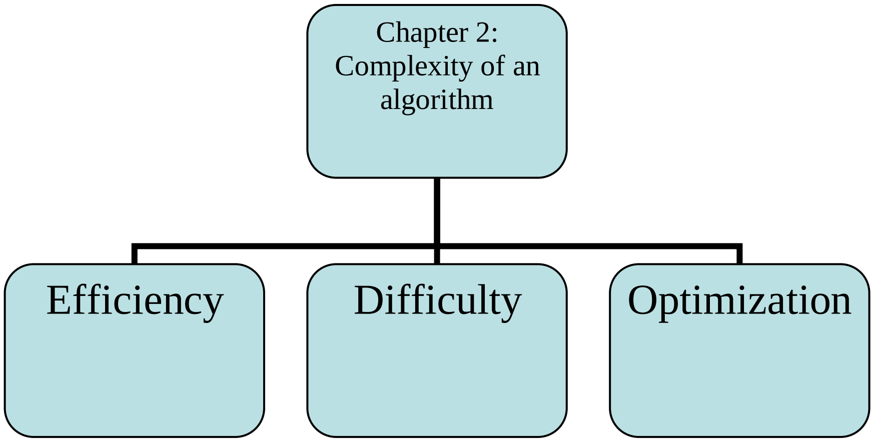
5) Recommended reading (optional)

a) Introduction to design and analysis of algorithms. Chapter 2

b) When people talk about an efficient algorithm, they mainly take into account the time complexity. This aspect is based on the number of steps that the algorithm does while running. That is how programmers are able to compare algorithms without worrying about the velocity of the machine and other aspects.

The big O notation is used to represent the complexity of an algorithm in the worst cases. Therefore, in some cases, the program might do less steps than the ones described by the notation but never more. Besides, time complexity can also be seen as a warning to look for simpler solutions or choose the most efficient one when we have more than one solution for the problem.

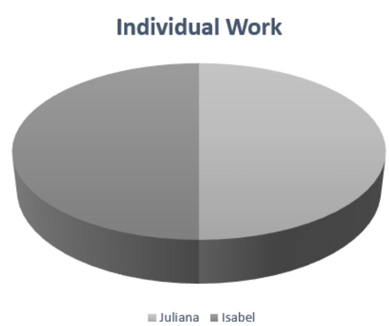
c) Concept map



6) Team work and gradual progress (optional)

a) Individual Work

Member	Part of the Laboratory	Description
Juliana	1	Insertion Sort
	2	Array 3
	3	3.4 maxSpan Explanation 3.5 Complexity 3.6 Description of the complexity
	4	Even excercises (2,4,6,8,10,12)
	6 and 7	Team work and english
Isabel	1	Merge Sort and Insertion Sort
	2	Array 2
	3	3.1 Tables 3.2 Graphics 3.3 Comparison
	4	Odd excercises (1,3,5,7,9,11,13)
	5	Lecture
	6 and 7	Team work and english



Team work(Meeting Minutes):

Date	Time	Description
31.08.18	1 hour	Discuss answers of part 4 Tell the other why the answer was chosen and correct posible mistakes
8.09.18	1 hour	Discussion of other parts
9.09.18	20 minutes	Collect answers of both parts (Document)

b) History of changes of the code

Version	Includes	Left	Status
1.0	Array 2	Array 3 Insertion Sort MergeSort	
2.0	Array 2 Insertion Sort	Array 3 Merge Sort	
3.0	Array 2 Insertion Sort MergeSort		Complete

c) History of changes of the report

Version	Includes	Left	Status
1.0	Practice for Midterms	Part 1 Part 2 Part 3 Part 5 Part 6	
2.0	Parts 1, 2(half), 3 (half),4, 5(half)	2,3,5,6	
3.0	Parts 1, 2, 3 (half),4, 5(half),6	3,5	
4.0	Parts 1,2,3,4,5,6		Complete