Laboratory practice No. 2: Algorithm complexity

Martin Ospina Uribe
Universidad Eafit
Medellín, Colombia
mospinau1@eafit.edu.co

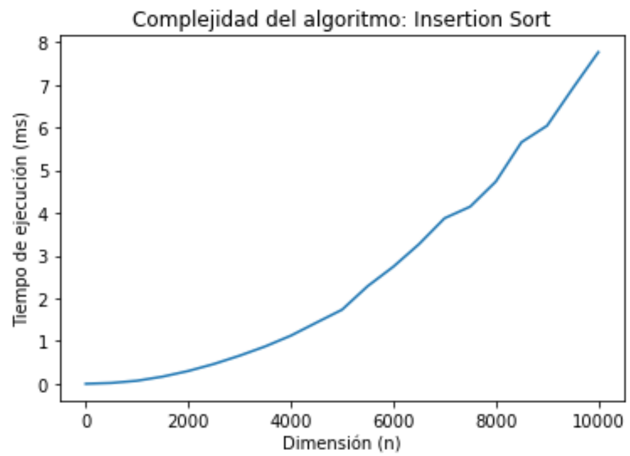
María José Bernal Vélez
Universidad Eafit
Medellín, Colombia
mjbernalv@eafit.edu.co

3) Practice for final project defense presentation

3.1 / 3.2 Insertion sort and merge sort times and graphs

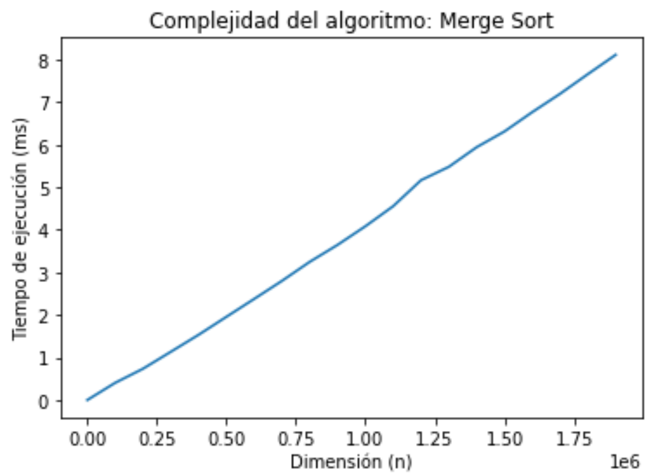
Insertion sort and merge sort are two algorithms, whose main goal is to sort a given array from the smallest to the greatest value. They both use different methods to achieve this, which is portrayed in the time they take to sort (time complexity).

Insertion Sort	
n (array size)	**Execution time (ms)**
1	2.1457672119140625e-06
501	0.019939184188842773
1001	0.07129192352294922
1501	0.16805076599121094
2001	0.29959988594055176
2501	0.4637000560760498
3001	0.6581487655639648
3501	0.8744778633117676
4001	1.1267216205596924
4501	1.4326379299163818
5001	1.7347662448883057
5501	2.287015914916992
6001	2.74155592918396
6501	3.2680959701538086



7001	3.8765981197357178
7501	4.148007154464722
8001	4.73550009727478
8501	5.656491041183472
9001	6.043961763381958
9501	6.918871879577637

Merge Sort	
n (array size)	Execution time (ms)
100	0.00018095970153808594
100100	0.40360593795776367
200100	0.7345011234283447
300100	1.132476806640625
400100	1.5320167541503906
500100	1.9543070793151855
600100	2.3752949237823486
700100	2.7998950481414795
800100	3.247506856918335
900100	3.6458871364593506
1000100	4.082847833633423
1100100	4.55500602722168
1200100	5.165136814117432
1300100	5.479034185409546
1400100	5.940356969833374
1500100	6.310158729553223
1600100	6.764622926712036
1700100	7.194669008255005
1800100	7.657653093338013
1900100	8.110867977142334



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

On both algorithms, the independent variable is the size of the array, which goes in the x axis, while the dependent variable is the time it takes to sort the given array, which goes on the y axis (if the array is bigger, it takes more time).

Both graphs correspond to the complexity of insertion sort and merge sort, which are $O(n^2)$ and $O(nlogn)$ respectively.

3.3 Is it appropriate to use insertion sort for a videogame with millions of elements in a scene and real time demands in rendering?

No. It is not appropriate to use insertion sort because of its complexity, which is $O(n^2)$. This means that it would take a lot of time to process millions of elements, and it would not be able to fulfill real time demands.

3.4 Why is a logarithm in the asymptotic complexity, for the worst case, in merge sort or insertion sort?

A logarithm appears in the asymptotic complexity of merge sort as the array is repeatedly divided by two in order for the algorithm to work.

3.5 For big arrays, how should data be so that insertion sort is faster than merge sort? Organized? All the same? Different? Disorganized?

In order for insertion sort to be faster than merge sort, data must be ordered from smallest to greatest or they must be the same as the second loop of the algorithm would not be used, so complexity would be O(n) (which is better than O(nlogn)).

3.7 / 3.8 Explanation and complexity of online exercises and meaning of variables

Array 2:

Count evens:

This algorithm counts the number of even numbers in an array that is sent to the method as a parameter. In order to do this, we use a cycle to iterate through the array by searching for numbers whose module by two is equal to zero and adding one two a variable we use as a counter. Finally, it returns the counter, which corresponds to the number of even numbers in the array.

- **Recurrence equation:**

$$T(n) = c * n$$

- **Big O notation:**

$$O(n)$$

- **n =** length of the array

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

Big diff:

This algorithm finds the difference between the greatest and the smallest number in a given array. To do this, it iterates through the array, and compares the values, finding the maximum and minimum, until it reaches the end of the array. Finally, it returns the difference between those two numbers, which corresponds to the biggest difference.

- **Recurrence equation:**

$$T(n) = c * (n - 1) * n$$
$$T(n) = cn^2 - cn$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array

Centered average:

This algorithm calculates the "centered average" of an array (given as a parameter), which is the average of its values without counting the largest and the smallest numbers. To do this, it calculates the smallest and biggest numbers in the array with two loops, and then, it finds the total sum of the values in the array. Finally, it finds the average by subtracting the maximum and minimum from the sum and dividing it by its length minus 2 (which are the values that we removed).

- **Recurrence equation:**

$$T(n) = c * (n - 1) * n$$
$$T(n) = cn^2 - cn$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array

Sum 13:

This algorithm calculates the sum of all the digits of an array, except for the number 13 and numbers that come immediately after a 13 also do not count. In order to do this, we use one loop to iterate through the array adding all the elements in the array and a conditional inside of it that skips the position twice when the number 13 appears in the array. Finally, it returns the variable that kept track of the sum of the array.

- **Recurrence equation:**

$$T(n) = c * n$$

- **Big O notation:**

$$O(n)$$

- **n =** length of the array

Has 22:

Given an array, this algorithm finds if there is a 2 immediately followed by another 2 in the array. To achieve this, the function uses a loop to iterate through the whole array, and then uses a conditional to see if both, the number that corresponds to the current

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

position of the loop and the one after it, have a value of 2. If they do, it returns true, else, it returns false.

- **Recurrence equation:**

$$T(n) = c * n$$

- **Big O notation:**

$$O(n)$$

- **n =** length of the array

Array 3:

Series up:

This algorithm receives a number and creates an array with a size of x*(x + 1)/2, where x is the parameter received, that has a pattern that goes from 1 to x in ascending order. For example, if the value is 3, the array would be [1, 1,2, 1,2,3]. To achieve this, it uses two loops to add the corresponding values in the correct position, which is represented by a counter variable (increases every time an element is added). Finally, it returns the array that was created.

- **Recurrence equation:**

$$T(n) = c * n^2$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array created (given by x*(x + 1)/2, where x is the number given)

Linear in:

This algorithm receives two arrays as parameters (outer array and inner array) and returns true if all of the numbers in inner appear in outer. To do this we create two variables that work as counters (“in” and “out”), and we use a conditional inside of a loop so every time that the value of the outer array in the “out” position is the same as the value of the inner array in the “in” position we add one to the variable “in”. Finally, if the value of “in” is the same as the length of the inner array it returns true (which means that all the elements of inner are in outer).

- **Recurrence equation:**

$$T(n) = c * n$$

- **Big O notation:**

$$O(n)$$

- **n =** length of the outer array (longest)

Can balance:

This algorithm checks to see if it is possible to find a position to divide an array in two groups such that the sum of both groups is the same. To do this, it iterates through the array and uses two variables, sum1 and sum2. Then, it uses other 2 loops, one that starts from the beginning (position 0) and adds the values to sum1 until it reaches the current position, and the other that starts from the current position and adds the values to

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

sum2 until the end of the array. Finally, it compares if sum1 and sum2 are the same, and if they are, it returns true, else, it returns false.

- **Recurrence equation:**

$$T(n) = c * n^2$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array

Fix 34:

This algorithm receives an array and returns an array that contains exactly the same numbers as the given array but rearranged so that every 3 is immediately followed by a 4. In order to do this, we implement a conditional inside a loop that iterates through the array so every time that it finds a value in the array equal to 3 it stores the value of the next element in a temporary variable and enters another loop that will search from that position on an element with value equal to 4. Then, it will change the element that follows the 3 with the element 4. Finally, it returns the array with every 3 immediately followed by a 4.

- **Recurrence equation:**

$$T(n) = c * n^2$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array

Fix 45:

This algorithm receives an array and modifies it so that it contains the same values, but with the conditions that every 4 must be immediately followed by a 5, 4's must stay in the same position, and any other number may be moved. To rearrange it, it iterates through the array using 2 variables, and if the number in the second variable is equal to 4 and the one in the first value is equal to 5, it creates a temporary variable that stores the number immediately after the 4. After that, the number after the 4 is changed to a 5, and the value where the 5 was receives the value that was stored in the temporary variable. At last, the modified array is returned.

- **Recurrence equation:**

$$T(n) = c * n^2$$

- **Big O notation:**

$$O(n^2)$$

- **n =** length of the array

4) Practice for midterms

4.1 The algorithm would take 100ms or 0.1s.
4.2 d
4.3 a

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

4.4 1.$O(n*m)$
2.$O(n*m)$
4.5 d
4.6 b
4.7 1, 3 and 4
4.8 a
4.9 c
4.10 c
4.11 c
4.12 a or c

6) Teamwork and gradual progress (optional)

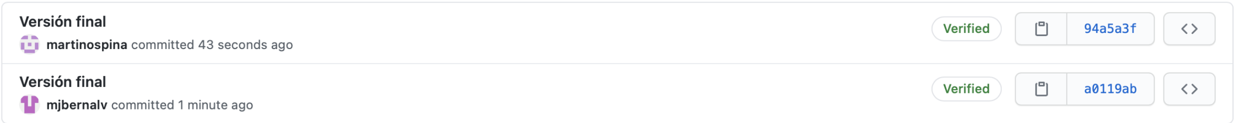
Member	Date	Done	Doing	To do
Martín and Maria José	03/03/2021	Project simulation exercise (1.1), CodingBat Array 2 (2.1), practice for midterms (4.1 - 4.12)	Finding complexities of Array 2 exercises.	CodingBat Array 3.
Martín and Maria José	10/03/2021	CodingBat Array 3 exercises.	Finding complexities of Array 3 exercises.	Write the full report
Martín and Maria José	14/03/2021	Work and finish the report. Upload laboratory on GitHub.		

6.1 Meeting minutes

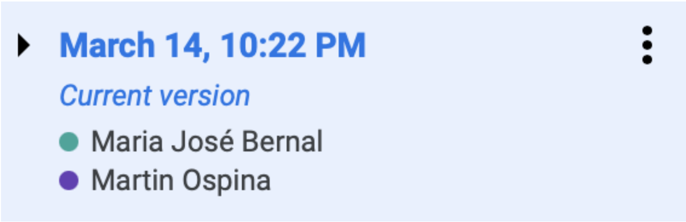
MU	Martin Ospina Uribe	Outgoing	1h 53m	3/10 5:24 PM	...
MU	Martin Ospina Uribe	Outgoing	21m 40s	3/3 7:43 PM	...
MU	Martin Ospina Uribe	Outgoing	1h 56m	3/3 5:45 PM	...
MU	Martin Ospina Uribe	Outgoing	1h 24m	9:07 PM	...

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

6.2 History of changes of the code



6.3 History of changes of the report



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