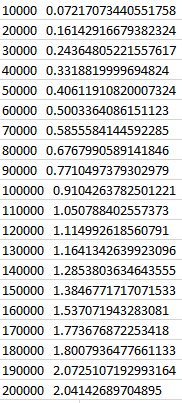
**Laboratory practice No. 2: Complexity**

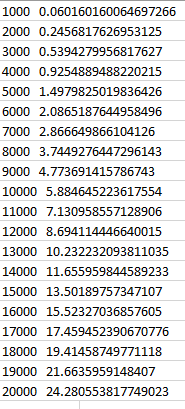
|  |  |
| --- | --- |
| **David Calle Gonzales**  Universidad Eafit  Medellín, Colombia  dcalleg@eafit.edu.co | **Julian Ramirez Giraldo**  Universidad Eafit  Medellín, Colombia  jramirezg@eafit.edu.co |

**3) Practice for final project defense presentation**

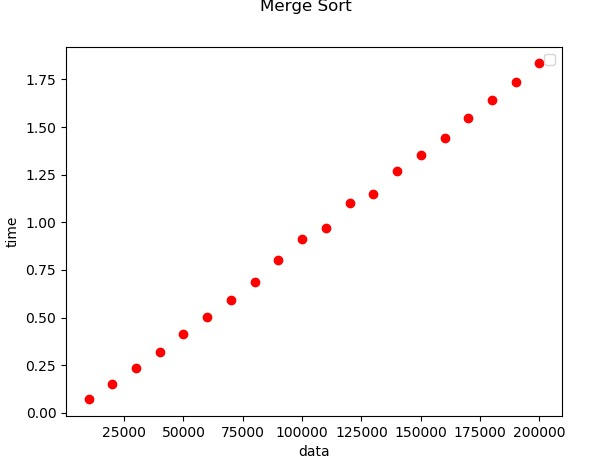
**3.1 MERGE SORT:**

****

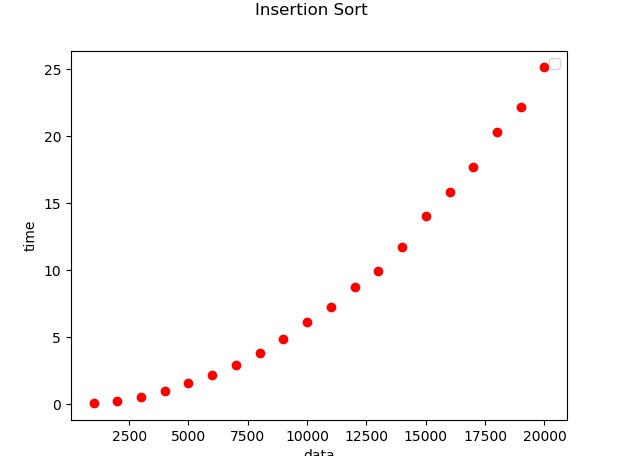
**INSERTION SORT**

****

**3.2 MERGE SORT**

****

**INSERTION SORT:**

****

**3.3** the efficiency of merge sort is incredible higher than insertion sort's due to its complexity, insertion sort has a complexity of n^2 and merge sort n\*log(n), therefore, using merge instead of insertion would be a great decision for big amounts of data

**3.4** taking into account what we explained in numeral 3.3, it is accurate to say that using insertion sort for big amounts of data would not be the best idea because of it's complexity of n^2

**3.5** the only way insertion sort could be faster than merge sort is if the data were already organized, in that case, it would only execute n steps

**3.6** MaxSpan works as follows: First, two variables called "span" and "tmp" are performed, a cycle from 0 to the size of the "nums" array is started, then, in a nested way, another "for" cycle is performed that of the same way that the principal works from 0 to the size of the array.

Then a condition enters, if the element in “i” is equal to the element in “j” then the temporary file will keep the distance between those two ranges with a simple operation that is "j-i + 1", that is, the largest position (j ) the minor position is subtracted (i) and one is added since it is sought to know how many numbers there are among them but not the position.

Then "span" is defined as the maximum value between tmp and span to have which is the size of the longest chain among those that are evaluated. Finally the value of "span" will return

***4) Practice for midterms***

4,1 -> c) O(n+m)

4.2 -> d)O(m\*n)

4.3 -> b)O(ancho)

4.4 -> b)O(n^3)

4.5 -> d) O(n^2)

4.6 -> a) T(n)= T(n-1)+C

4.7.1 -> T(n) = T(n-1)+C

4.7.2 -> O(n)

4.8 -> a) The algorithm execute T(n) = c+T(n-1) steps. The asymptotic complexity is O(n)

4.9 -> d) Run more steps than nxm

4.10 -> The answers are wrong. The complexity of the algorithm is: O(n)

4.11 -> c) T(n)=T(n-1)+T(n-2)+c

4.12 -> b)O(m.n.log(n)+n\*m^2+n^2.log(n)+m^3)

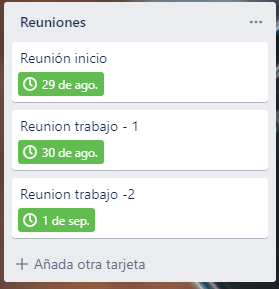
4.13 -> c) T(n) = 2T(n/2)+n

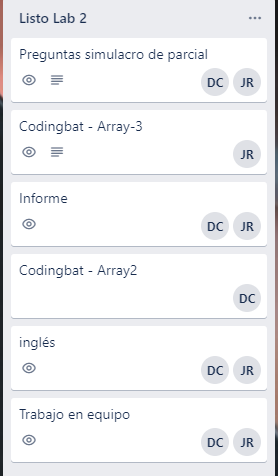
4.14 -> a) O(n^3+n(log(log(m))+m x √(m)

***5) Recommended reading (optional)***

Mapa conceptual

**6)** **Team work and gradual progress (optional)**

****

****