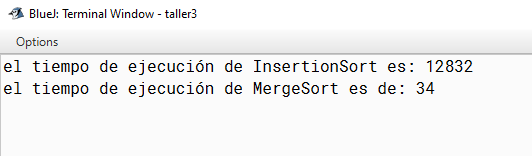
# **Laboratory practice No. 2**

**Algorithm complexity**

Samuel Salazar Salazar Carla Sofía Rendón Baliero

|  |  |
| --- | --- |
| Universidad Eafit  Medellín, Colombia  correoinegrante1@eafit.edu.co | Universidad Eafit  Medellín, Colombia  csrendonb@eafit.edu.co |

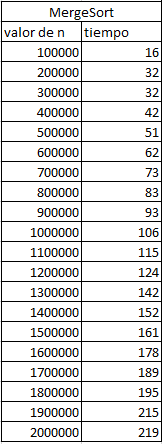
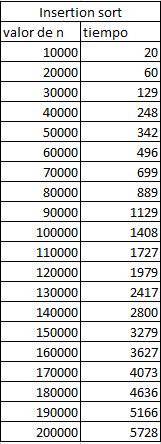
**1.1**



Disclaimer: Both algorithms had an array of 300,000 spaces and all numbers were randomized.

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

**3.1**



**3.2**

**3.3** The complexity of the insert is O (n ^ 2), taking into account the case of applying this algorithm in a game of millions of elements in a scene and demands for 3D rendering time, mainly this algorithm would not be appropriate for this. case and more taking into account that the algorithm having a complexity of n ^ 2 means that it requires the n ^ 2 of steps to order elements, that is, twice, which makes it efficient with small lists but in the case of having a large list would be completely inefficient.

**3.4** The complexity of the marge sort is O (n log n). And this is due to the nature of this algorithm, in which it consists of dividing an array recursively, in short the data vector divides itself until reaching the size '1' of elements, that means that the vector of size 1 is ordered. When the vector is in this part, each sub-vector is recursively ordered 'joining' them and comparing values ​​with the vector next to it, in the case of the complexity that is O (n log n) it is due to the entire input it must be iterated , and this must happen O (log (n)) times (the input can only be halved O (log (n)) times) and therefore n iterated elements log (n) times give O (n log (n )).

**3.5**

**Array 2**

**Exercise 1(countEvens)**

Complexity = O(1)+O(n) // C+T(n)

O (n)

**Exercise 2(sum13)**

Complexity = O(1)+O(n+1) // C+T(n+1)

O (n)

**Exercise 3 (bigDiff)**

Complexity = O(1)+O(n) // C+T(n)

O (n)

**Exercise 4 (bigDiff)**

Complexity = O(1)+O(n) // C+T(n)

O (n)

**Exercise 5(squareUp)**

Complexity = O(1)+O(n) // C+T(n)

O (n)

**Array 3**

**Exercise 1(maxSpan)**

Complexity = O(1)+O(n˄2) // T(n)+C

O (n ^ 2)

**Exercise 2(canBalance) (the for are not nested)**

Complexity = O(1)+O(n) // C+T(n)

O (n)

**Exercise 3 (linearIn)**

Complexity = O(1)+O(n˄2) // T(n)\*T(n)+C

O (n ^ 2)

**Exercise 4 (seriesUp)**

Complexity = O(1)+O(n˄2) // T(n)\* T(n)+C

O (n ^ 2)

**Exercise 5(squareUp)**

Complexity = O(1)+O(n˄2) // T(n)\*T(n)+C

O (n ^ 2)

**3.4 (opcional) (solución del max span)**

public int maxSpan(int[] nums) {

    int max = 0;

Max and min will help us find the maximum interval

    int min = 0;

to check if the array is empty

    if(nums.length == 0){

    return 0;

    }

It is compared if the number furthest to the left and the number furthest to the right are equal, that means that the interval is the size of the array

    if(nums[0] == nums[nums.length-1]){

        max = nums.length;

    }

the first for will allow us to compare the first arrangement position (which will change positions) and the second for will not serve to compare that first arrangement position with the other positions that follow and so on.

for(int i = 0; i < nums.length; i++){

          for(int j = 1; j < nums.length; j++){

and the if will be responsible for making the aforementioned comparison, and if the positions are equal it means that we already have an interval which will be in size of j (position more to the right) and it is stored in the variable min

if(nums[i] == nums[j]){

             min = j;

if(min>max){

                max=min;

And then a comparison is made with the variable max (which at the beginning will be 0) and since the minimum will be greater than the maximum, the minimum will be assigned to the maximum in order to compare the new js that are had and if it allows us to find the maximum interval

             }

            }

         }

      }

      return max;

    }

***4) Practice for midterms***

* 1. 100ms
  2. b) O(m×n×√n)
  3. c) O(n 3× n)
  4. 1 O(n+m+n\*m)= O(n\*m)

2 O(n+m+n\*m)

* 1. 1 (d)T(n)=T(n/10)+c, what is a O(log10 n)

2(a) Yes

***4.7*** 3. Si f=O(g) y g=O(h), then f=O(h) (the correct answer 3)