Laboratorio Nro. 2: Notación O grande

Agustín Rico
Universidad EAFIT
Medellín, Colombia
aricop@eafit.edu.co

Santiago Cano
Universidad EAFIT
Medellín, Colombia
scanof@eafit.edu.co

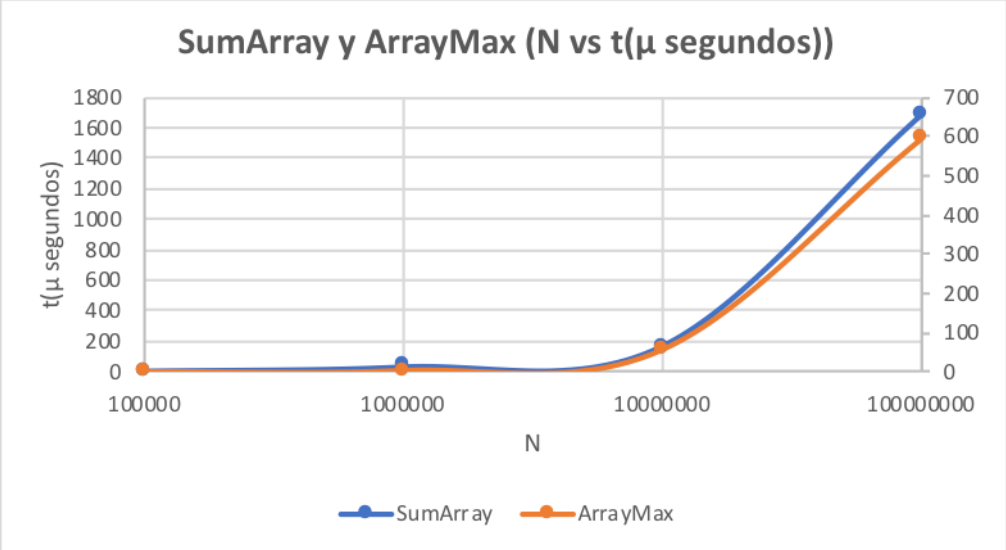
3) Simulacro de preguntas de sustentación de Proyectos

1.

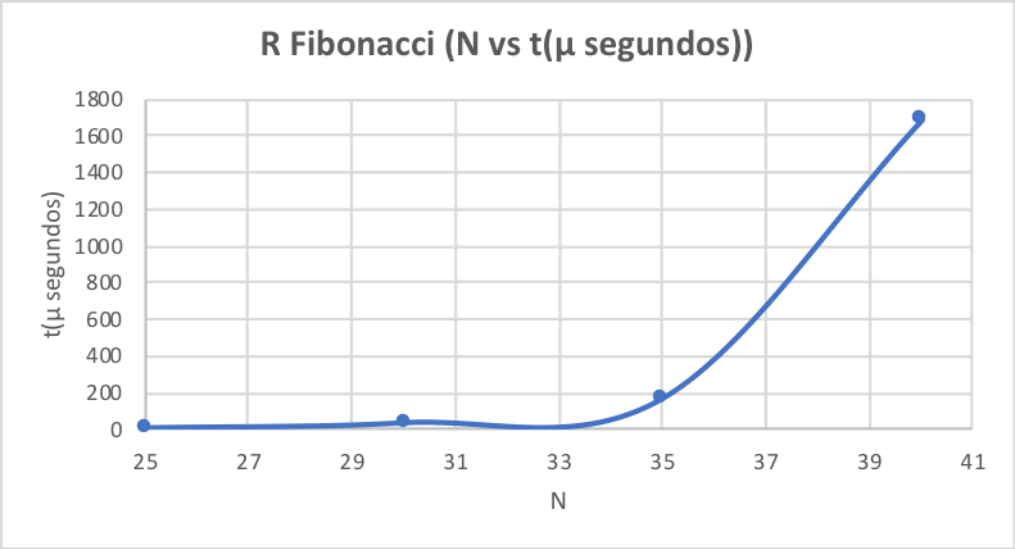
	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
R ArraySum	**4**	**32**	**166**	**1686**
R ArrayMax	**2**	**5**	**57**	**593**

	N = 25	N = 30	N = 35	N = 40
R Fibonacci	**1**	**4**	**38**	**413**

2.



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

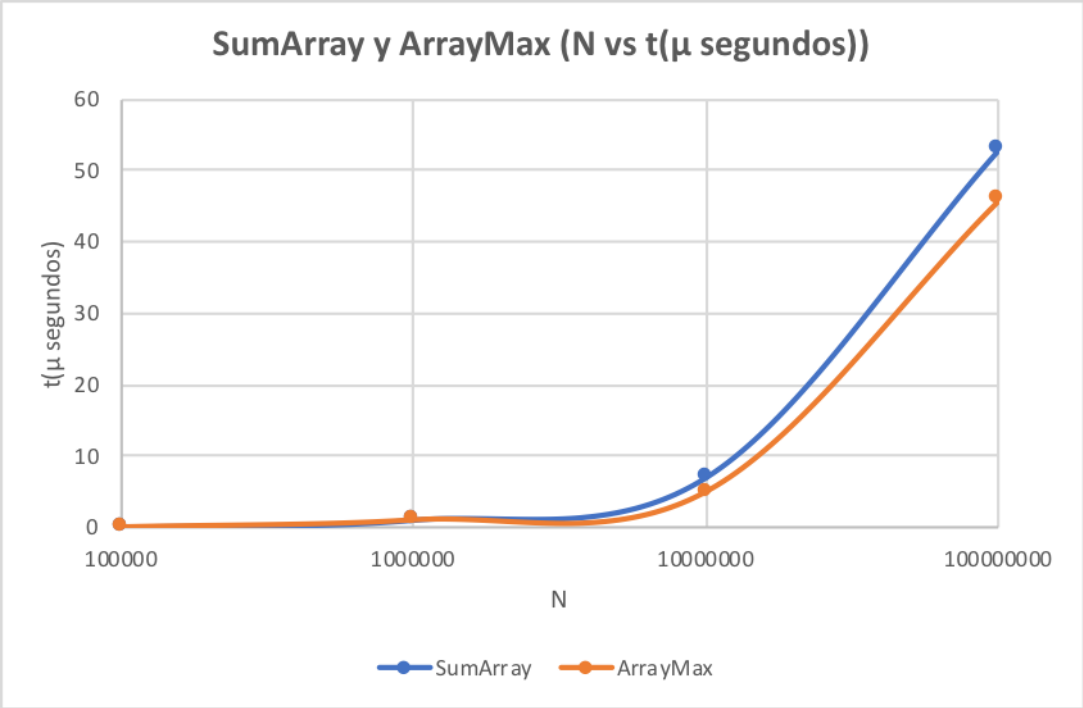


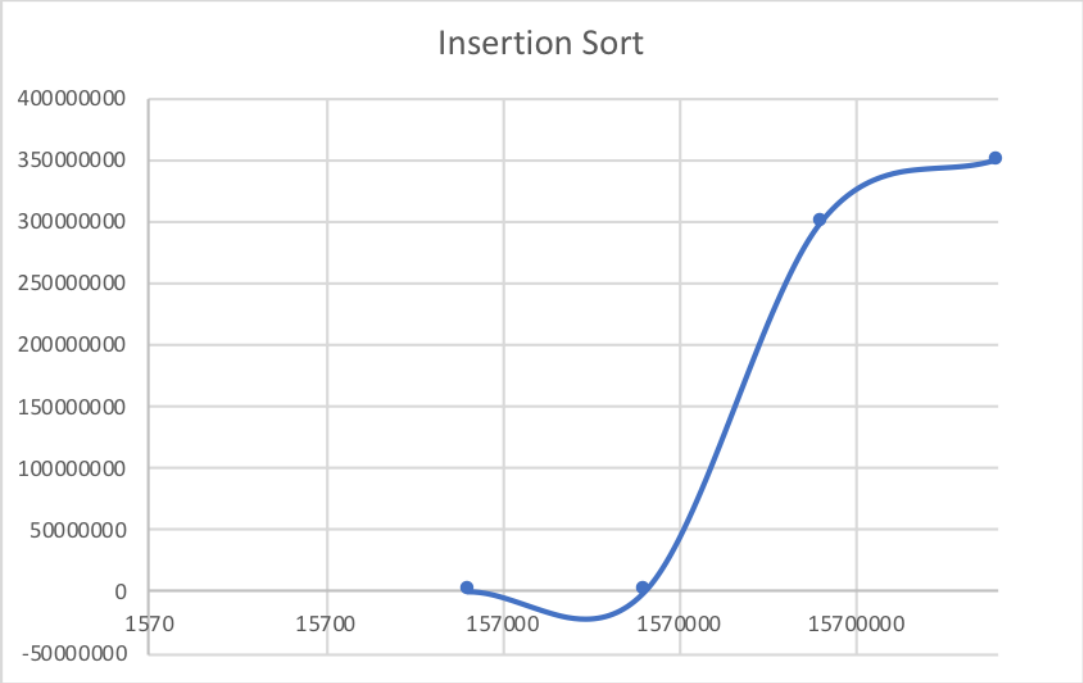
3. Se puede concluir que los resultados obtenidos son los esperados de acuerdo con los resultados teóricos, en los intervalos que se tomaron los resultados se puede observar que concuerdan con los valores encontrados con la notación O.

4.

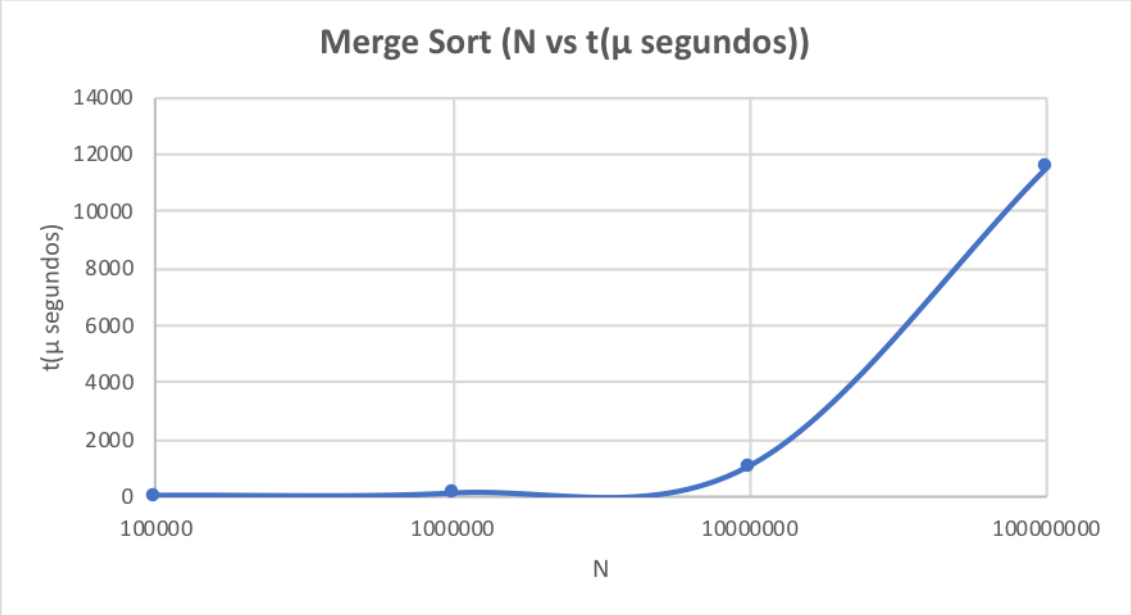
	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
ArraySum	**0**	**1**	**7**	**53**
ArrayMax	**0**	**1**	**5**	**46**
Insertion Sort	**1578**	**167704**	**Más de cinco minutos**	**Más de cinco minutos**
Merge Sort	**16**	**99**	**1078**	**11538**

5. Debido a que los tiempos del Insertion Sort y Merge Sort son mucho más grandes que los de los otros dos métodos, sus gráficas se hicieron aparte, pues de lo contrario, gráficamente parecía que para todos los valores de ArraySum y ArrayMax el tiempo fuera cero.





DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co



6. Los resultados experimentales obtenidos son los esperados comparados con los resultados teóricos.
7. El algoritmo Insertion Sort con valores muy grandes, tarda muchísimo tiempo en llevar acabo todo el ordenamiento, lo que es lógico debido a su complejidad que en el peor de los casos es n^2, donde n es el número de elementos que tiene el arreglo que se esta ordenando.
8. El algoritmo arraysum con valores grandes, tarda poco tiempo en procesar el resultado.
   Si lo comparamos con el algoritmo de ordenamiento Insertion Sort, se puede notar que el tiempo cambia de manera abismal debido a que la complejidad de arraysum es O(n) y la de Insertion Sort es O(n^2)
9. Para arreglos grandes es mucho más eficiente el algoritmo Merge Sort, pero si miramos arreglos pequeños es más eficiente Insertion Sort. Para valores de "n" menores que 100 es mucho más eficiente Insertion Sort, de lo contrario será mucho más eficiente Merge Sort.
10. El ejercicio maxSpan consiste en encontrar el máximo número de elementos entre dos apariciones del mismo elemento en un arreglo. El algoritmo comienza preguntándose si la longitud del *Array* es menor o igual a 1, porque en ese caso, no podría haber dos apariciones del mismo elemento, entonces retornaría la longitud del elemento. En el caso contrario, de longitud del *Array* mayor que 1, lo que hace el algoritmo es ejecutar dos ciclos, uno dentro del otro. El ciclo anidado lo que hace es contar cuántos elementos se hay desde la aparición del elemento que se está evaluando en el primer ciclo, hasta encontrarlo de nuevo en el *Array*, en ese caso, se detiene el contador y se compara si ese contador fue mayor al anterior, es decir, cuando el primer ciclo estaba evaluando el elemento anterior. De ser mayor, se reemplaza; y de esta manera hasta el último elemento del *Array*, para al final retornar el contador máximo.
11. **ARRAY 2**

```
public boolean lucky13(int[] nums) {
  for(int i = 0; i < nums.length; i++) {        //C1 * n
    if(nums[i] == 1 || nums[i] == 3) {          //C2 * n
      return false;                             //C3 * n
    }                                           //C4 * n
  }
  return true;                                  //C5
```

```
}
```

Donde n es el número de elementos en el arreglo nums
Complejidad:
En el mejor de los casos es constante
T(n) = C1 + C2 + C3 + C4 ó T(n)= C5

En el peor de los casos es lineal.
T(n)= C1* n+ C2* n+ C3* n+ C4* n+ C5
T(n)= n * (C1 + C2 +C3 + C4)
T(n) = n O(n)

```
public boolean sum28(int[] nums) {
    int sum8 = 0;                       //C1
    for(int i = 0; i < nums.length; i++) {      //C2*n
        if(nums[i] == 2) {                  //C3*n
            sum8 += nums[i];                //C4*n
        }                                   //C5*n
    }                                       //C6*n
    if(sum8 == 8) {                         //C8
        return true;                        //C9
    }                                       //C10
    return false;                           //C11
}
```

Donde n es el número de elementos en el arreglo nums.
Complejidad:
En el mejor de los casos es constante
T(n) = C1 + C8 + C11

En el peor de los casos es lineal.
T(n)= C1 + C2* n+ C3* n+ C4* n+ C5*n +C6*n+C8+C9+C10+C11
T(n)= n * (C2 + C3 +C4 + C5+ C6)
T(n) = n O(n)

```
public boolean more14(int[] nums) {
    int count1 = 0; int count4 = 0;         //C1
    for(int i = 0; i < nums.length; i++) {      //C2*n
        if(nums[i] == 1) {                  //C3*n
            count1++;                   //C4*n
        } else if(nums[i] == 4) {               //C5*n
            count4++;                   //C6*n
        }                                   //C7*n
    }                                       //C8*n
```

```
    if(count1 > count4) {                   //C9
        return true;                        //C10
    }                                       //C11
    return false;                           //C12
}
```

Donde n es el número de elementos en el arreglo nums.

Complejidad:

En el mejor de los casos es constante

T(n) = C1 + C9 + C12

En el peor de los casos es lineal.

T(n)= C1 + C2* n+ C3* n+ C4* n+ C5*n +C6*n+C7*n+C8*n+C9+C10+C11+ C12

T(n)= n * (C2 + C3 +C4 + C5+ C6+ C7 +C8)

T(n) = n O(n)

```
public int[] fizzArray(int n) {
    int [] nums = new int [n];              //C1
    for(int i = 0; i < nums.length; i++) {  //C2*n
        nums[i] = i;                        //C3*n
    }                                       //C4*n
    return nums;                            //C5
}
```

Donde n es el número de elementos en el arreglo nums, lo que es igual al número que se pasa por parametro.

Complejidad:

En el mejor de los casos es constante

T(n) = C1 + C5

En el peor de los casos es lineal.

T(n)= C1 + C2* n+ C3* n+ C4* n+ C5

T(n)= n * (C2 + C3 +C4)

T(n) = n O(n)

```
public boolean only14(int[] nums) {
    for(int i = 0; i < nums.length; i++) {          //C1*n
        if(nums[i] != 4 && nums[i] != 1) {          //C2*n
            return false;                           //C3*n
        }                                           //C4*n
    }                                               //C5*n
    return true;                                    //C6
}
```

Donde n es el número de elementos en el arreglo nums.

Complejidad:

En el mejor de los casos es constante

T(n) = C6

En el peor de los casos es lineal.
T(n)= C1*n + C2* n+ C3* n+ C4* n+ C5*n + C6
T(n)= n * (C1+ C2 + C3 +C4+ C5)
T(n) = n O(n)

ARRAY 3

```
public static int maxSpan(int[] nums) {
    if(nums.length <= 1) {                      //C1
        return nums.length;                     //C2
    } else {                                    //C3
        int maxSpan = 0;                        //C4
        for(int i = 0; i < nums.length; i++) {          //C5*n
            int span = 0;                       //C6*n
            for(int j = i; j < nums.length; j++) {      //C7*n*n
                span++;                         //C8*n*n
                if(nums[j] == nums[i]) {                //C9*n*n
                    if(span > maxSpan) {                //C10*n*n
                        maxSpan = span;         //C11*n*n
                    }                           //C12*n*n
                }                               //C13*n*n
            }                                   //C14*n*n
        }                                       //C15*n
        return maxSpan;                         //C16
    }
}
```

Donde n es el número de elementos en el arreglo nums.
Complejidad:
En el mejor de los casos es constante
T(n) = C1+C2
En el peor de los casos es cuadrática.
T(n)=C1+C3+C4+C5*n+C6*n+C7*n*n+C8*n*n+C9*n*n+C10*n*n+C11*n*n+C12*n*n+C13*n*n+ C14*n*n+ C15*n +C16
T(n)= n*(C5+C6+C15+ n*(C7+ C8 + C9 +C10+C11+C12 +C13 +C14 +C15))
T(n) = n*n O(n^2)

```
public int[] fix34(int[] nums) {
    for(int i = 0; i < nums.length; i++) {              //C1*n
        for(int j = nums.length - 1; j >= 0; j--) {             //C2*n*n
            if(nums[i] == 3 && nums[i + 1] != 4) {          //C3*n*n
                if(nums[i] == 3 && nums[j] == 4) {          //C4*n*n
                    int temp = nums[i + 1];             //C5*n*n
                    nums[i + 1] = -4;                   //C6*n*n
                    nums[j] = temp;                     //C7*n*n
                    break;                              //C8*n*n
                }
            }
        }
    }
    for(int i = 0; i < nums.length; i++) {                  //C9*n
        if(nums[i] == -4) {                             //C10*n
```

```
            nums[i] = 4;                        //C11*n
        }
    }
    return nums;                        //C12
}
```

Donde n es el número de elementos en el arreglo nums.

Complejidad:

En el mejor de los casos es constante

T(n) = C12

En el peor de los casos es cuadrática.

T(n)=C1*n+C2*n*n+C3*n*n+C4*n*n+C5*n*n+C6*n*n+C7*n*n+C8*n*n+C9*n+C10*n+C11*n+C12

T(n)= n*(C1+C9+C10+C11+ n*(C2+ C3 + C4 +C5+C6+C7 +C8))

T(n) = n*n O(n^2)

```
public static int[] fix45(int[] nums) {
    for(int i = 0; i < nums.length; i++) {          //C1*n
        for(int j = 0; j < nums.length; j++) {      //C2*n*n
            if(nums[i] == 4 && nums[j] == 5) {      //C3*n*n
                int temp = nums[i + 1];             //C4*n*n
                if(temp != 5) {                     //C5*n*n
                    nums[i + 1] = -5;               //C6*n*n
                    nums[j] = temp;                 //C7*n*n
                    break;                          //C8*n*n
                } else {                        //C9*n*n
                    nums[i + 1] = -5;               //C10*n*n
                    break;                          //C11*n*n
                }
            }
        }
    }
    for(int i = 0; i < nums.length; i++) {          //C12*n
        if(nums[i] == -5) {                     //C13*n
            nums[i] = 5;                        //C14*n
        }
    }
    return nums;                        //C15
}
```

Donde n es el número de elementos en el arreglo nums.

Complejidad:

En el mejor de los casos es constante

T(n) = C15

En el peor de los casos es cuadrática.

T(n)=C1*n+C2*n*n+C3*n*n+C4*n*n+C5*n*n+C6*n*n+C7*n*n+C8*n*n+C9*n*n+C10*n*n +C11*n*n +C12*n+C13*n+C14*n+C15

T(n)= n*(C1+C12+C13+C14+ n*(C2+ C3 + C4 +C5+C6+C7 +C8+C9+C10+C11))

T(n) = n*n O(n^2)

```
public boolean canBalance(int[] nums) {
    for(int i = 0; i < nums.length; i++) {          //C1*n
        int sum1 = 0; int sum2 = 0;             //C2*n
```

```
        for(int j = 0; j <= i; j++) {                    //C3*n*n
            sum1 += nums[j];                             //C4*n*n
        }
        for(int k = i + 1; k < nums.length; k++) {       //C5*n*n
            sum2 += nums[k];                             //C6*n*n
        }
        if(sum1 == sum2) {                               //C7*n
            return true;                                 //C8*n
        }
    }
    return false;                                        //C9
}
```

Donde n es el número de elementos en el arreglo nums.

Complejidad:

En el mejor de los casos es constante

T(n) = C9

En el peor de los casos es cuadrática.

T(n)=C1*n+C2*n+C3*n*n+C4*n*n+C5*n*n+C6*n*n+C7*n+C8*n+C9

T(n)= n*(C1+C2+C7+C8+ n*(C3 + C4 +C5+C6))

T(n) = n*n O(n^2)

```
public boolean linearIn(int[] outer, int[] inner) {
    for(int i = 0; i < inner.length; i++ ) {             //C1*n
        boolean appears = false;                         //C2*n
        for(int j = 0; j < outer.length; j++) {          //C3*m*n
            if(outer[j] == inner[i]) {                   //C4*m*n
                appears = true;                          //C5*m*n
            }
        }
        if(appears == false) {                           //C6*n
            return false;                                //C7*n
        }
    }
    return true;                                         //C8
}
```

Donde n es el número de elementos en el arreglo inner; y m es el número de elementos del arreglo outer.

Complejidad:

En el mejor de los casos es constante

T(n) = C8

En el peor de los casos es cuadrática.

T(n)=C1*n+C2*n+C3*n*m+C4*n*m+C5*n*m+C6*n+C7*n

T(n)= n*(C1+C2+C6+C7+ m*(C3 + C4 +C5))

T(n) = n*m O(m*n)

4) Simulacro de Parcial

1. *c*
2. *d*
3. *b*
4. *b*
5. *d*

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

6. *a*

7. *7.1: T(n) = T(n-1) + C* *7.2:O(n)*

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co