

Laboratorio Nro. 2: Notación O grande

María Paula Chaparro Muñoz
Universidad Eafit
Medellín, Colombia
mpchaparm@eafit.edu.co

Juan José Parra Díaz
Universidad Eafit
Medellín, Colombia
jjparrad@eafit.edu.co

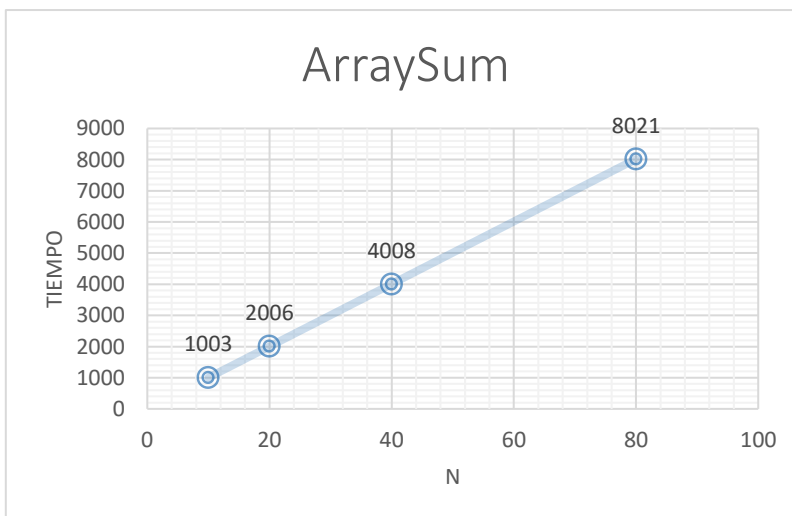
3) Simulacro de preguntas de sustentación de Proyectos

1.

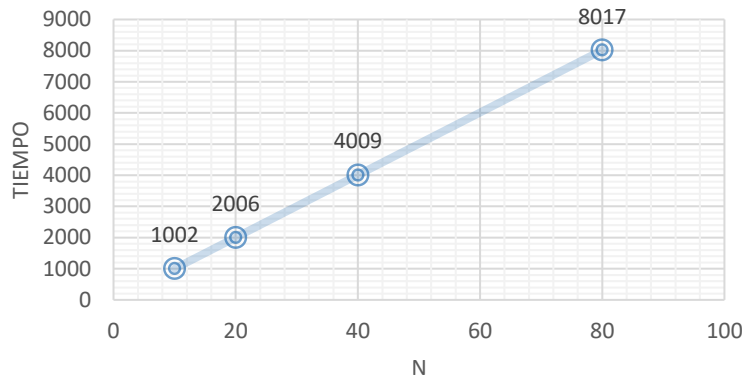
	N = 10*	N = 20*	N = 40*	N = 80*
Array sum	1003	2006	4008	8021
Array maximum	1002	2006	4009	8017
Insertion Sort	3008	9727	39703	149774
Merge Sort	902	1906	3906	7917

*Con un retraso en cada llamado recursivo (para evitar errores de Stack Overflow)

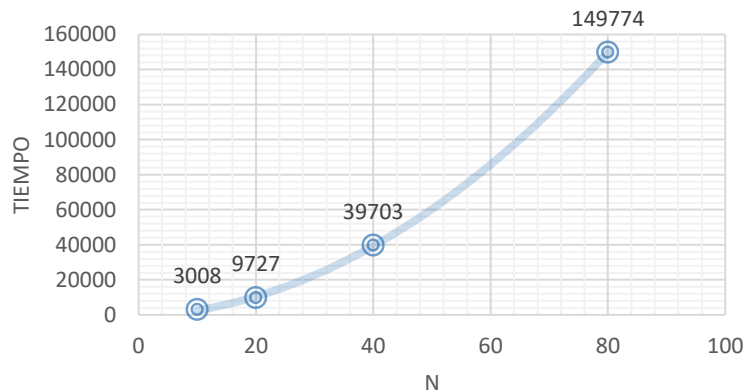
2.

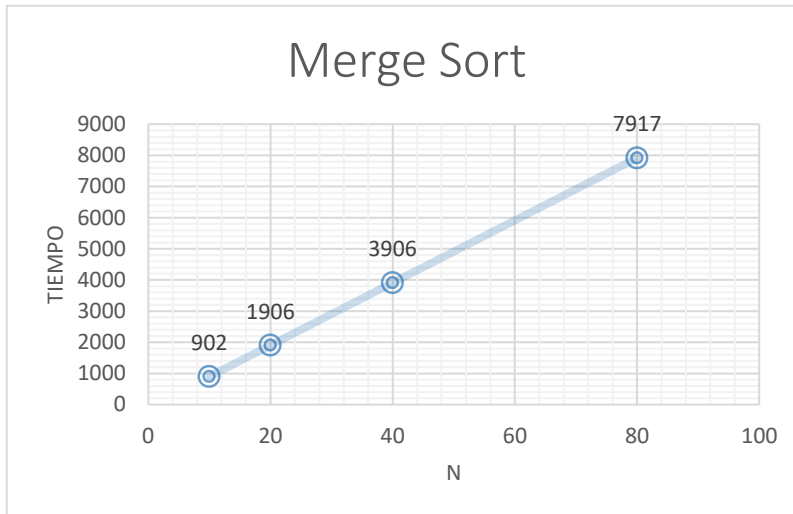


Array Maximum



Insertion Sort





3. Los tiempos obtenidos en el laboratorio han sido concordes con respecto a los teóricos obtenidos a partir de las complejidades de estos, puesto que las gráficas y tiempos de ejecución tardaron más en el punto que tenía la mayor complejidad, y en los otros menos.
4. Cuando se toma los tiempos de InsertionSort con valores n muy grandes, lo que sucede es que se tarda mucho tiempo, ya que es una gráfica que no aumenta linealmente, lo que quiere decir, que cuando se hace con muchos valores, comienza a tender a un tiempo infinito.
5. ArraySum con valores muy grandes, se hará con la misma velocidad que se hace con valores muy pequeños, solamente que tardará más por la cantidad. Esto es porque tiene una complejidad $O(n)$, lo que quiere decir que aumenta su tiempo de ejecución linealmente.
6. Con valores muy grandes MergeSort es mucho más apropiado de usar que InsertionSort, ya que MergeSort en cada paso, reduce el tamaño del problema a la mitad, mientras que InsertionSort no. En cambio, con valores pequeños, InsertionSort es más adecuado puesto que se vuelve más rápido mover un elemento a dividir el problema, muchas veces, porque como no es de un tamaño considerable el arreglo, organizarlo con MergeSort no justifica.
7. El ejercicio maxSpan, funciona parándose en cada elemento de un arreglo. Desde este se compara con cada elemento comenzando desde la izquierda, y

cuando se encuentre a sí mismo halla la diferencia entre su posición con la de su compañero. Hallada esta, la compara con la distancia en el arreglo de sus competidores, y si es mayor, la destrona. Al final se retorna 1 en caso de que todos los elementos sean diferentes o que el arreglo esté vacío, y el maxSpan en caso de que no.

8.

```
public int sum13(int[] nums) {  
    if (nums.length == 0) {           C1  
        return 0;                     C2  
    }  
    int s = 0;                        C3  
    for (int i = 0; i < nums.length; i++){ C4*n  
        if (nums[i] != 13){           C5*n  
            s += nums[i];             C6*n  
        } else {                     C7*n  
            i ++;                     C8*n  
        }  
    }  
    return s;                         C9  
}
```

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(C*n)$$

$$T(n) = O(n)$$

```
public boolean has22(int[] nums) {  
    for (int i = 0; i < nums.length-1; i++){ C1*n  
        if (nums[i] == 2 && nums[i+1] == 2){ C2*n  
            return true; C3*n  
        }  
  
    }  
    return false; C4  
}
```

$T(n) = C' + C*n$

$T(n) = O(C' + C*n)$

$T(n) = O(C*n)$

$T(n) = O(n)$

```
public boolean sum28(int[] nums) {  
    int doss = 0; C1  
    for (int i = 0; i < nums.length; i++){ C2*n  
        if (nums[i] == 2){ C3*n  
            doss += 2; C4*n  
        }  
    }  
  
    if (doss == 8){ C5  
        return true; C6  
    }  
    return false; C7  
}
```

$T(n) = C' + C*n$

$T(n) = O(C' + C*n)$

$T(n) = O(C*n)$

$T(n) = O(n)$

```

public int[] fizzArray(int n) {
    int[] a = new int[n];          C1
    for (int i = 0; i < n; i++){    C2*n
        a[i] = i;                  C3*n
    }
    return a;                      C4
}

T(n) = C' + C*n
T(n) = O(C' + C*n)
T(n) = O(C*n)
T(n) = O(n)

public boolean haveThree(int[] nums) {
    int tres = 0;                  C1
    for (int i = 0; i < nums.length-1; i++){    C2*n
        if (nums[i] == 3 && nums[i+1] != 3){    C3*n
            tres++;                            C4*n
        } else if (nums[i] == 3 && nums[i+1] == 3){    C5*n
            return false;                      C6*n
        }
    }
    if (nums.length > 2){            C7
        if (nums[nums.length-1] == 3 && nums[nums.length-2] != 3){    C8
            tres++;                    C9
        }
    }
    if (tres == 3){                  C10
        return true;                C11
    }
    return false;                    C12
}

T(n) = C' + C*n
T(n) = O(C' + C*n)
T(n) = O(C*n)
T(n) = O(n)

```

```
public int maxSpan(int[] nums) {  
    int maxS = 0;                                C1  
    for (int i = 0; i < nums.length; i++)         C2*n  
        for (int j = nums.length - 1; j > i; j--) C3*n*m  
            if (nums[j]==nums[i])                 C4*n*m  
                if (j - i + 1 > maxS)              C5*n*m  
                    maxS = j-i + 1;               C6*n*m  
    if (maxS == 0 && nums.length != 0){           C7  
        return 1;                                C8  
    }  
    return maxS;                                  C9  
}
```

$$T(n) = C' + C*n + C*n*m$$

$$T(n) = O(C' + C*n + C*n*m)$$

$$T(n) = O(C*n + C*n*m)$$

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

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

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

```

public int[] fix45(int[] nums) {
    int temp = 0;
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] == 4) {
            for (int j = 0; j < nums.length; j++) {
                if (nums[j] == 5) {
                    if (j > 0 && nums[j-1] != 4) {
                        temp = nums[i+1];
                        nums[i+1] = 5;
                        nums[j] = temp;
                    }
                    else if (j == 0) {
                        temp = nums[i+1];
                        nums[i+1] = 5;
                        nums[j] = temp;
                    }
                }
            }
        }
    }
    return nums;
}

```

C1
C2*n
C3*n
C4*n*m
C5*n*m
C6*n*m
C7*n*m
C8*n*m
C9*n*m
C10*n*m
C11*n*m
C12*n*m
C13*n*m
C14

$$T(n) = C' + C*n + C*n*m$$

$$T(n) = O(C' + C*n + C*n*m)$$

$$T(n) = O(C*n + C*n*m)$$

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

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


```
public boolean linearIn(int[] outer, int[] inner) {  
    int numsR = 0;                                C1  
    for (int i = 0; i < inner.length; i++){        C2*n  
        for (int j = 0; j < outer.length; j++){    C3*n*m  
            if (inner[i] == outer[j]){             C4*n*m  
                numsR++;                           C5*n*m  
                break;                             C6*n*m  
            }  
        }  
    }  
    if (numsR == inner.length){                    C7  
        return true;                              C8  
    } else {                                       C9  
        return false;                            C10  
    }  
}
```

$$T(n) = C' + C*n + C*n*m$$

$$T(n) = O(C' + C*n + C*n*m)$$

$$T(n) = O(C*n + C*n*m)$$

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

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

```
public int[] seriesUp(int n) {  
    int[] nums = new int [n*(n + 1)/2]; C1  
    int a = 0; C2  
    for (int i = 0; i <= n; i++){ C3*n  
        for (int j = 1; j <= i; j++){ C4*n*m  
            nums[a] = j; C5*n*m  
            a++; C6*n*m  
        }  
    }  
    return nums; C7  
}
```

$$T(n) = C' + C*n + C*n*m$$

$$T(n) = O(C' + C*n + C*n*m)$$

$$T(n) = O(C*n + C*n*m)$$

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

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


```
public int countClumps(int[] nums) {  
    int c = 0;                                C1  
    for (int i = 1; i < nums.length-1; i++){ C2*n  
        if (nums[i-1] == nums[i]){           C3*n  
  
            } else if (nums[i+1] == nums[i]) { C4*n  
                c++;                          C5*n  
            }  
        }  
    }  
    if (nums.length == 0) return 0;          C6  
    if (nums[0] == nums [1]){                C7  
        c++;                                C8  
    }  
    return c;                                C9  
}
```

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(C*n)$$

$$T(n) = O(n)$$

	<p style="text-align: center;">UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	<p>Código: ST245</p> <p>Estructura de Datos 1</p>
---	---	---

9. En el cálculo de la complejidad, 'n' representa el tamaño del arreglo, es decir que entre más grande sea, más complejidad tendrá el punto, y 'm' es el proceso que se va a hacer con respecto a la longitud del arreglo ('n').

4) *Simulacro de Parcial*

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