ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 2
Arrays y Notación O grande

Santiago Ochoa Castaño
Universidad Eafit
Medellín, Colombia
sochoac1@eafit.edu.co

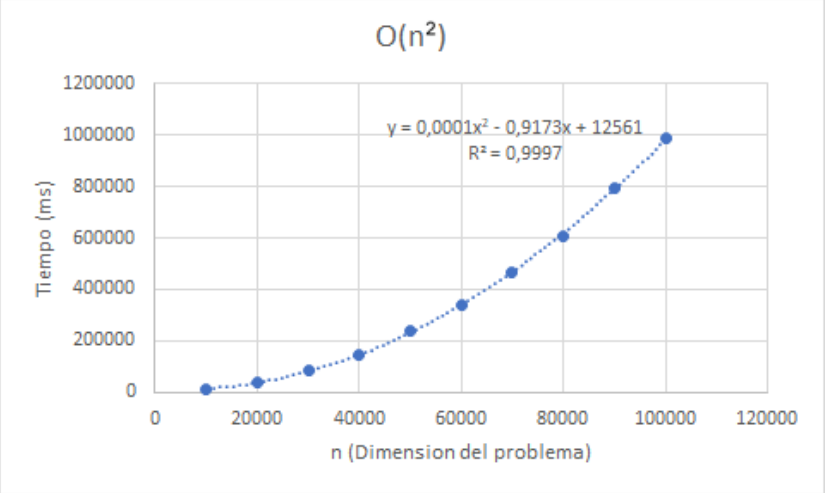
Miguel Ángel Zapata Jiménez
Universidad Eafit
Medellín, Colombia
mazapataj@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.2. Grafiquen los tiempos que tomó en cada algoritmo para los 20 tamaños del problema. Grafiquen el Tamaño de la Entrada Vs. Tiempo de Ejecución.

INSERTION SORT:

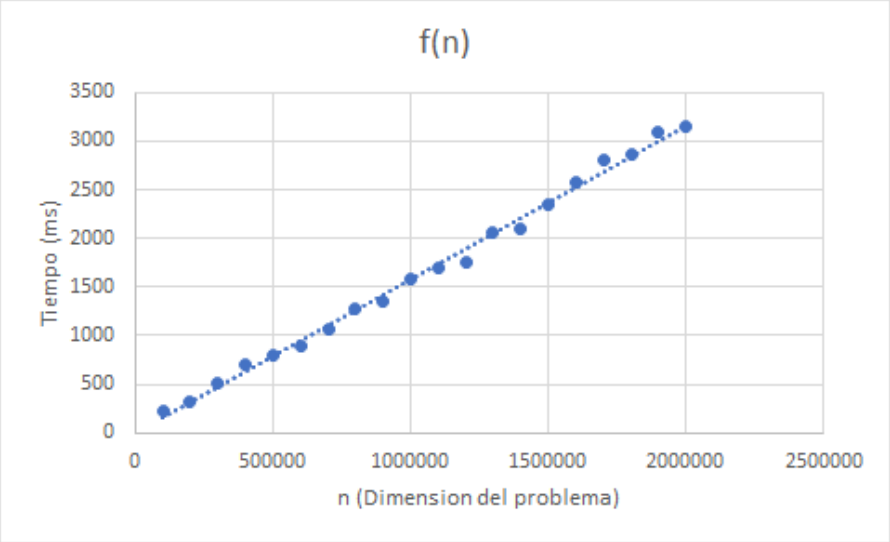
n	f(n)
100000	9039
200000	38456
300000	84571
400000	142180
500000	230959
600000	220161
700000	310533
800000	375927
900000	365546
1000000	446058



MERGE SORT:

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

n	f(n)
100000	231
200000	311
300000	501
400000	707
500000	794
600000	884
700000	1057
800000	1282
900000	1349
1000000	1584
1100000	1691
1200000	1760
1300000	2059
1400000	2106
1500000	2347
1600000	2570
1700000	2806
1800000	2855
1900000	3099
2000000	3141



3.3. ¿Es apropiado usar insertion sort para un videojuego con millones de elementos en una escena y demandas de tiempo real en la renderización?

- El insertion sort, pese a poseer una simple y estable implementación eficiente para conjuntos de datos pequeños, a la hora de enfrentarse a conjuntos de muchos elementos, por ejemplo, en el caso de un videojuego durante el renderizado de texturas en tiempo real, tiende a tardarse mucho. El algoritmo compara cada elemento con su predecesor, y suponiendo en el peor de los casos que el arreglo se encuentre en orden decreciente para ordenarlo de menor a mayor sería muy lento. Además, tiene una complejidad que tiende a comportarse de orden cuadrático($O(n^2)$) lo cual no es nada óptimo para grandes cantidades de elementos.

3.4. ¿Por qué aparece un logaritmo en la complejidad asintótica, para el peor de los casos, de merge sort o insertion sort?

- Para el peor de los casos, solo para el método de ordenamiento merge sort se obtiene una complejidad de O(nlog(n)). El motivo por el cual adquiere esta complejidad se debe a que para el ordenamiento se divide el arreglo en dos mitades y así sucesivamente comportándose como un logaritmo. De esta manera se puede concluir que el merge sort es más eficiente que el insertion sort ya que el insertion sort tarda más tiempo en realizar sus operaciones debido a que posee una complejidad $O(n^2)$.

3.7. Calculen la complejidad de los Ejercicios en Línea de los numerales 2.1 y agréguenla al informe PDF

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

ESTRUCTURA DE DATOS 1
Código ST0245

Array 2

- **fizzArray3**

```
public static int [] fizzArray3(int start, int end){
      int tamaño = end - start;
      int[] nums = new int[tamaño];
      for(int i = 0; i < tamaño; i++){
         nums[i] = start;
         start = start + 1;
      }
      return nums;
   }
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2
T4(n): C3 + C4*n
T5(n): C5*n
T6(n): C6*n
T8(n): C7

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

O(C1 + C2 + C3 + C4*n + C5*n + C6*n + C7)

O(C4*n + C5*n + C6*n)

O(n(C4 + C5 +C6))

O(n)

- **Only14**

```
public boolean only14(int[] nums) {
      boolean x=true;
      for(int i=0; i < nums.length;i++){
         if(nums[i]==1||nums[i]==4){
            x=true;
         }else{
            x=false;
            break;
         }
      }
      return x;
   }
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2 + C3*n
T4(n): C4*n
T5(n): C5*n
T11(n): C6

T(n): C1 + C2 +C3*n C4*n + C5*n + C6

O(C1 + C2 +C3*n C4*n + C5*n + C6)

O(C3*n C4*n + C5*n)

O(n(C3 + C4 + C5))

O(n)

- **CenteredAvarage**

```
public int centeredAverage(int[] nums) {
      int n = nums.length;
      int pos_menor, temp;
      int promedio = 0;
      int cont = 0;
      for (int i = 0; i < n - 1; i++) {
         pos_menor = i;
         for (int j = i + 1; j < n; j++) {
            if (nums[j] < nums[pos_menor]){
               pos_menor = j;
            }
```

Complejidad Asintótica:

T2(n): C1
T4(n): C2
T5(n): C3
T6(n): C4 + C5*n
T7(n): C6*n
T8(n): (C7 + C8*n) *n
T9(n): (C9*n) *n
T10(n): (C10*n) *n

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
        }
        temp = nums[i];
        nums[i] = nums[pos_menor];
        nums[pos_menor] = temp;
    }
    for (int i = 1; i < nums.length - 1; i++){
        if(nums[i] == nums[i+1] && i +1 < nums.length-1){
            continue;
        }else{
            promedio = promedio + nums[i];
            cont++;
        }
    }
    return promedio/cont;
}
```

T11(n): C11*n
T12(n): C12*n
T13(n): C13*n
T14(n): C14+C15*n
T15(n): C16*n
T17(n): C17*n
T19(n): C18*n
T20(n): C19*n
T23(n): C20

T(n): C1+C2+C3+C4+C5*n+C6*n+(C7+C8*n) *n+(C9*n) *n+(C10*n) *n+C11*n+C12*n+C13*n+C14+C15*n+C16*n+C17*n+C18*n+C20

O(C1+C2+C3+C4+C5*n+C6*n+(C7+C8*n) *n+(C9*n) *n+(C10*n) *n+C11*n+C12*n+C13*n+C14+C15*n+C16*n+C17*n+C18*n+C20)

O(C5*n+C6*n+(C7+C8*n) *n+(C9*n) *n+(C10*n) *n+C11*n+C12*n+C13*n+C15*n+C16*n+C17*n+C18*n)

$O(C5*n+C6*n+C7*n+C8n^2+C9*n^2+C10*n^2 +C11*n+C12*n+C13*n+C15*n+C16*n+C17*n+C18*n)$

$O(n(C5+C6+C7+C11+C12+C13+C15+C16+C17+C18) + n^2 (C8+C9+C10))$

$O(n+n^2)$

$O(n^2)$

- **FizzArray**

```
public int[] fizzArray(int n) {
    int[] nums= new int[n];
    for(int i=0; i<nums.length;i++){
        nums[i]=i;
    }
    return nums;
}
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2 + C3*n
T4(n): C4*n
T6(n): C5

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

O(C3*n + C4*n)

O(n(C3 + C4))

O(n)

- **Shiftleft**

```
public int[] shiftLeft(int[] nums) {
    if(nums.length == 0)return nums;
    int num2[] = new int[nums.length];
    int temp = nums[0];
    int cont = 0;
    for(int i = 1;i< nums.length; i++){
        num2[cont] = nums[i];
        cont++;
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2
T4(n): C4
T5(n): C5 + C6*n
T6(n): C7*n
T7(n): C8*n
T9(n): C9
T10(n): C10

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

}
num2[cont] = temp;
return num2;
}
	T(n): C1 + C2 + C3 + C4 + C5 + C6*n + C7*n + C8*n + C9 + C10
	
	O(C1 + C2 + C3 + C4 + C5 + C6*n + C7*n + C8*n + C9 + C10)
	
	O(C6*n + C7*n + C8*n)
	
	O(n(C6 + C7 + C8))
	
	O(n)

Array 3

- **Fix34**

```
public int[] fix34(int[] nums) {
    for(int i = 0; i < nums.length; i++){
        if(nums[i] == 3){
            for(int j = 0; j < nums.length; j++){
                if(nums[j] == 4&&nums[j-1]!=3){
                    nums[j] = nums[i+1];
                    nums[i+1] = 4;
                }
            }
        }
    }
    return nums;
}
```

Complejidad Asintótica:

T2(n): C1 + C2*n
T2(n): C3*n
T4(n): (C4 + C5*n)*n
T5(n): (C6)*n*n
T6(n): (C7)*n*n
T7(n): (C8)*n*n
T12(n): C9

$T(n): C1 + C2*n + C3*n + C4*n + C5*n^2 + C6*n^2 + C7*n^2 + C8*n^2 + C9$

$O(C1 + C2*n + C3*n + C4*n + C5*n^2 + C6*n^2 + C7*n^2 + C8*n^2 + C9)$

$O(C2*n + C3*n + C4*n + C5*n^2 + C6*n^2 + C7*n^2 + C8*n^2)$

$O(n(C2 + C2 + C4) + n^2(C5 + C6+ C7 + C8))$

$O(n + n^2)$

$O(n^2)$

- **LinearIn**

```
public boolean linearIn(int[] outer, int[] inner) {
    int cont = 0;
    boolean res = false;
    if(inner.length == 0) return true;
    for(int i = 0; i < inner.length; i++){
        for(int j = 0; j < outer.length; j++){
            if(inner[i] == outer[j]){
                cont++;
                break;
            }
        }
        if(cont == inner.length){
            res = true;
        }
    }
    return res;
}
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2
T4(n): C3
T5(n): C4 + C5*n
T6(n, m): (C6 + C7*m) *n
T7(n, m): (C8*m) *n
T8(n, m): (C9*m) *n
T12(n): C11*n
T13(n): C12*n
T16(n): C13

T(n, m): C1 + C2 + C3 + C4 + C5*n + (C6+C7*m)*n +(C8*m)*n + (C9*m)*n + C11*n + C12*n + C13

O(C1 + C2 + C3 + C4 + C5*n + (C6+C7*m)*n +(C8*m)*n + (C9*m)*n + C11*n + C12*n + C13)

O(C5*n + (C6+C7*m)*n +(C8*m)*n + (C9*m)*n + C11*n + C12*n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

	O(C5*n + C6*n + C7*mn+ C8*mn + C9*mn + C11*n +C12*n) O(n(C5+C6+C11+C12)+mn(C7+C8+C9)) O(n + nm) O(nm)

- **CountClumps**

```
public int countClumps(int[] nums) {
    int cont = 0;
    boolean res = true;
    for (int i = 0; i < nums.length-1; i++ ) {
       if(nums[i] == nums[i+1] && res) {
          cont++;
          res = false;
       } else if(nums[i] != nums[i+1]) {
          res = true;
       }
    }
    return cont;
}
```

Complejidad Asintótica:

T2(n): C1
T3(n): C2
T4(n): C3 + C4*n
T5(n): C5*n
T6(n): C6*n
T7(n): C7*n
T8(n): C8*n
T12(n): C9

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

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

O(C4*n + C5*n + C6*n + C7*n + C8*n)

O(n(C4 + C5 + C6 + C7 + C8)

O(n)

- **Fix45**

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

Complejidad Asintótica:

T2(n): C_1 + C_2*n
T6(n): (C_3+C_4+C_5+C_6) *n
T8(n): (C_7+C_8*n) *n
T9(n): (C_9) *n
T10(n): (C_10) *n
T11(n): (C_11) *n
T12(n): C_12

T(n):C_1+C_2+(C_3+C_4+C_5+C_6)*n (C_7+C_8*n)*n+(C_9)*n+(C_10)*n+(C_11)*n+C_12

O(C_1+C_2+(C_3+C_4+C_5+C_6)*n+(C_7+C_8*n)* n+(C_9)*n+(C_10)*n+(C_11)*n+C_12)

O(C_1+C_2+C_3*n+C_4*n+C_5*n+C_6*n+C_7*n+C _8*n^2 +C_9*n+C_10*n+C_11*n+C_12)

O(C_3*n+C_4*n+C_5*n+C_6*n+C_7*n+C_8*n^2 +C_9*n+C_10*n+C_11*n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

	O(n(C_3+C_4+C_5+C_6+C_7+C_9+C_10+C_11) +C_8*$n^{2)}$ O(n + n^2) O(n^2)

- **CanBalance**

public boolean canBalance(int[] nums) {
int sumLeft=0;
int sumRigth=0;
for(int i=0; i < nums.length; i++){
sumLeft=sumLeft+nums[i];
for(int j=nums.length-1; j > i; j--){
sumRigth=sumRigth+nums[j];
}
if(sumLeft==sumRigth){
return true;
}
sumRigth=0;
}
return false;
}
	Complejidad Asintótica:
	
	T1(n): C1
	T2(n): C2
	T3(n): C3 + C4*n
	T4(n): C5*n
	T5(n): (C6 + C7*n)*n
	T6(n): C8*n^2
	T9(n): C9*n
	T11(n): C10*n
	T12(n): C11
	
	T(n): C1 + C2 + C3 + C4*n + C5*n + C6*n + C7* n^2 + C8* n^2 + C9*n + C10*n + C11
	
	O(C1 + C2 + C3 + C4*n + C5*n + C6*n + C7* n^2 + C8* n^2 + C9*n + C10*n + C11)
	
	O(C4*n + C5*n + C6*n + C7* n^2 + C8* n^2 + C9*n + C10*n)
	
	O(n(C4 * C5 + C6 + C9 + C10) + n^2(C7 + C8))
	
	O(n + n^2)
	
	O(n^2)

3.8. Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del ejercicio anterior.

Array 2

- **FizzArray3:** En el cálculo la variable n representa el tamaño del arreglo que es formado a partir de una variable start (int) y una variable end (int) que son recibidas a través de los parámetros establecidos de la función. El tamaño es formado a partir de la resta de la variable end y la variable start. La n también nos indica las veces que se debe ejecutar el ciclo "for" antes de que el índice deje de cumplir la condición de ser menor al tamaño.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

- **Only14:** En el cálculo la variable n representa el tamaño del arreglo que es mandado por el usuario al método y a su vez se puede interpretar que la n es las veces que se debe repetir el ciclo for antes de que el índice deje de cumplir la condición establecida de ser menor al tamaño del arreglo.

- **CenteredAvarage:** En el cálculo se puede apreciar que la variable n representa la cantidad de elementos que posee el arreglo enviado a la función y también nos indica las veces que se deben repetir los ciclos “for”.

- **FizzArray:** En el cálculo de la complejidad asintótica se puede apreciar que la n es un valor dado por el usuario a la función y que posteriormente se convertirá en el tamaño del arreglo que se debe retornar. De igual manera la n indica las veces que se debe repetir el ciclo “for” hasta que el índice sea mayor al tamaño del arreglo que en este caso es n.

- **Shiftleft:** En el cálculo de complejidad de este algoritmo encontramos que la n representa el tamaño del arreglo brindado por el usuario y las veces que debe ser ejecutado el ciclo “for”.

Array 3

- **Fizz34:** Al realizar el cálculo de complejidad se obtuvo que la n representa el tamaño de arreglo dado por el usuario a la función. También indican las veces que deben ser ejecutados los “for” hasta que los índices dejen de ser menores a la n (tamaño del arreglo) que en otras palabras es la condición que se establece en los ciclos “for” presentes.

- **LinearIn:** En la realización del cálculo para la complejidad asintótica de este algoritmo fue de O(nm). La n representa el tamaño del inner o arreglo a verificar si sus elementos se encuentran dentro de outer. De esta manera, m representa el tamaño de outer.

- **CountClumps:** Cuando realizamos la ecuación de complejidad nos damos cuenta de que la n representa el tamaño del arreglo que es ingresado por el usuario y las veces que debe ser repetido el ciclo “for”.

- **Fix45:** Al calcular la ecuación de complejidad nos damos cuenta de que la n representa el tamaño del arreglo que es ingresado por parte el usuario y también representa las veces que se debe repetir el ciclo “for” y hasta que el índice deje de cumplir la condición de ser menor al tamaño del arreglo.

- **CanBalance:** En el cálculo la variable n representa el tamaño del arreglo proporcionado por el usuario y también representa las veces que se deben repetir los ciclos “for” presentes en el algoritmo dado.

4) Simulacro de Parcial.

4.2. *B)*

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4.5.1. *D) /* ***4.5.2.*** *A)*

4.6. *Primero encontramos el valor de la constante C:*
Dado que T(100) = 1 seg, entonces:
$10000 = C \times 100^2$
C = 0.1
Sabiendo el valor de c, hallamos T(10000):
$T(10000) = (0.1)(10000)^2$
T(10000) = 10´000.000 ms
Lo pasamos a segundos:
$10´000.000ms * 1x10^{-4} = 10.000$ *seg*
Por lo tanto, T (10000) tardaría 10.000 seg

4.7.

- *La 1, 3 y 4 son verdaderas ya que hacen parte de las reglas de la notación O, pero cuando observamos la 2 nos damos cuenta de que no hace parte de las reglas. Cuanto tenemos O(n*m) hacemos referencia a una matriz y no a dos funciones separadas como lo muestra el ejemplo por lo que es falsa la segunda preposición.*

1. $O(f+g)=O(max(f,g))$
2. $O(f\times g)=O(f)\times O(g)$
3. Si $f=O(g)$ y $g=O(h)$, entonces $f=O(h)$
4. $O(c.f)=O(f)$, donde c es una constante

4.9. *A)*

4.14. *A)*

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473