Laboratorio Nro. 2: Notación O grande

Alejandra Martínez Vega
Universidad Eafit
Medellín, Colombia
amartinezv@eafit.edu.co

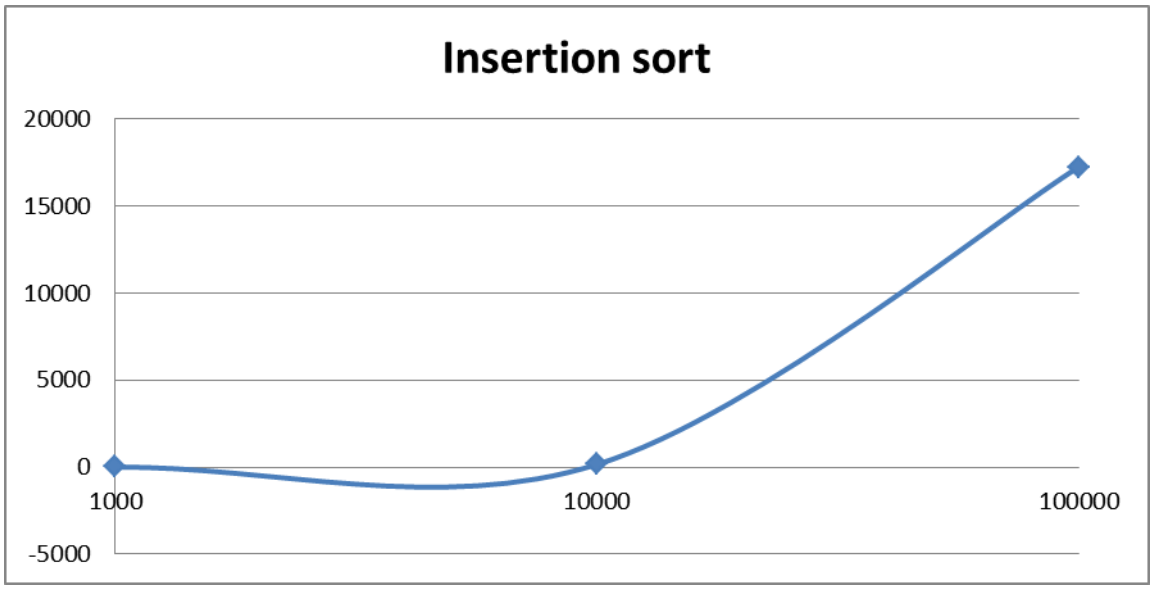
3) Simulacro de preguntas de sustentación de Proyectos

1.

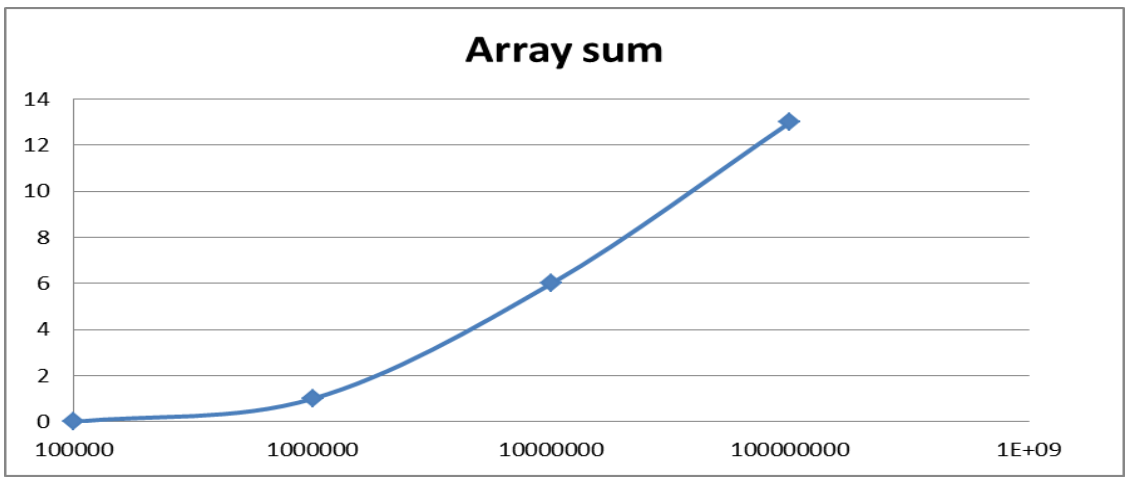
	N = 100.000	*N = 1'000.000*	*N = 10'000.000*	*N = 100'000.000*
Array sum	***0***	***1***	***6***	***13***
Array máximum	***0***	***1***	***10***	***21***
Insertion sort	***17246***	***Más de 5 min***	***Más de 5 min***	***Más de 5 min***
Marge sort	***16***	***174***	***1930***	***5442***

2.

Insertion sort	
Entrada	***Tiempo***
1000	***3***
10000	***156***
100000	***17246***

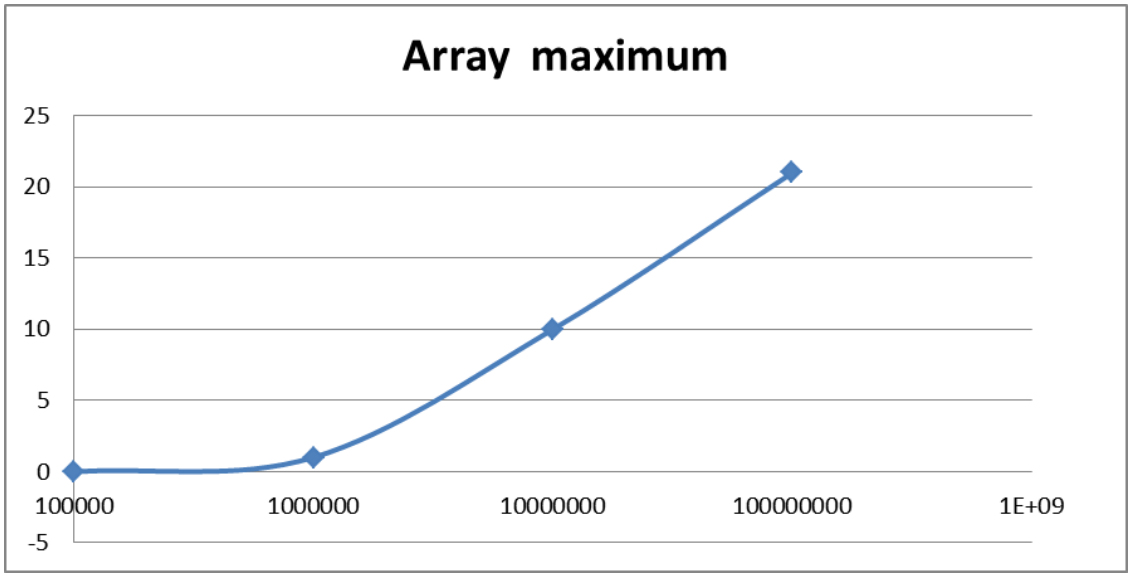


Array sum	
Entrada	***Tiempo***
100000	***0***
1000000	***1***
10000000	***6***
100000000	***13***



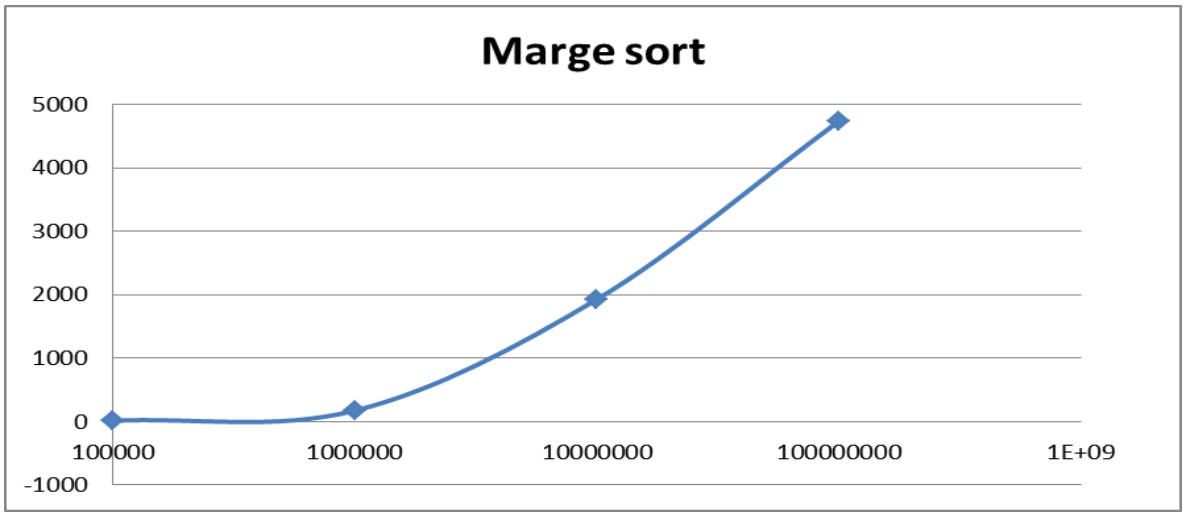
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Array maximum	
Entrada	***Tiempo***
100000	***0***
1000000	***1***
10000000	***10***
100000000	***21***



Marge sort	
Entrada	***Tiempo***
100000	***16***
1000000	***174***
10000000	***1930***
100000000	***4742***

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



3. ***Los tiempos obtenidos con el laboratorio se relacionan con la teoría obtenida de la notación O, con Array sum, Array máximum y Marge sort no hay problema porque es de orden lineal en cambio la complejidad de Insertion sort es de orden cuadrático y por eso con valores mayores el tiempo de ejecución es mucho más grande.***

4. ***Siendo n el número de elementos en el arreglo y estos generados de forma aleatorio con un valor máximo de cinco mil, el problema con Insertion sort para valores grandes de n es que recorre y compara cual elemento es mayor devolviéndose en el arreglo y comparándolo con el elemento a insertar hasta llegar a la posición correcta corriendo todos los elementos hacia la derecha y esto hace que la complejidad del problema con valores grandes sea mayor.***

5. ***Los tiempos no crece tan rápido con Array sum porque su complejidad es de orden lineal y solo se recorre el arreglo una vez y se va sumando todos los elementos hasta que se acabe el arreglo, en cambio Insertion sort recorre el arreglo y va moviendo todos sus elementos hacia la derecha hasta que el arreglo este ordenado lo que aumenta la complejidad y hace que los tiempos crezcan mucho más.***

6. ***Teniendo en cuenta los tiempos obtenidos en el laboratorio es más eficiente Marge sort con respeto a Insertion sort para arreglos grandes y es más eficiente Insertion sort para arreglos pequeños con respecto a Marge sort.***

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

7. ***El ejercicio maxSpan evalúa el primer y último elemento y se va devolviendo hasta encontrar un número que sea diferente al primero del arreglo y ahí calcula el span que es el número de elementos entre el mas a la izquierda y más a la derecha anteriormente en el arreglo.***

8.

1.) Array 1

i.

```
public boolean only14(int[] nums) {
        int cont = 0;                                   //c1
        for(int i = 0; i < nums.length; i++){           //c.n
          if(nums[i] == 1 || nums[i] == 4)cont++;       //c2
         }
        if(cont == nums.length)return true;             //c3
        return false;                                   //c4
    }
```

ii.

```
public boolean isEverywhere(int[] nums, int val) {
      for(int i = 0; i < nums.length-1; i++){                   //c.n
         if(nums[i] != val && nums[i+1] != val) return false;   //c1
       }
         return true;                                           //c2
      }
```

iii.

```
public int matchUp(int[] nums1, int[] nums2) {
    int cont = 0;                                       //c1
     for(int i = 0; i < nums1.length; i++){             //c.n
       int a = nums1[i] - nums2[i];                     //c2
          if(Math.abs(a) <= 2 && Math.abs(a) > 0 )cont++;   //c3
     }
        return cont;                                    //c4
  }
```

iv.

```
public boolean has12(int[] nums) {
    boolean comprovar = false;                          //c1
        for(int i = 0; i < nums.length; i++)            //c.n
       {
          if (nums[i] == 1)                             //c2
             comprovar = true;                          //c3
          if (comprovar && nums[i] == 2)                //c4
             return true;                               //c5
       }
       return false;                                    //c6
  }
```

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

v.
```
public boolean haveThree(int[] nums) {
    int cont3 = 0;                                              //c1
      if (nums.length < 5)                                      //c2
        return false;                                           //c3
      for (int i = 0; i < nums.length-1; i++){                  //c.n
        if (nums[i] == 3 && nums[i+1] != 3)                     //c4
          cont3++;                                              //c5
      }
      if (nums[nums.length-1] == 3 && nums[nums.length-2] != 3)//c6
        cont3++;                                                //c7
      return cont3 == 3;                                        //c8
  }
```

La complejidad en todos los casos es T(n) = O(n)

2.) Array 3

I.
```
public int maxSpan(int[] nums) {
    int max = 0;
     int retador = 0;
     for(int i = 0; i < nums.length; i++){                      //c.n
        int j = nums.length-1;
        while(nums[j] != nums[i]){                              //c.n.n
           j--;
        }
        retador = j - i + 1;
        if(retador > max) max = retador;
     }
     return max;
  }
```

T(n) = c + c.n + c.n.n
T(n) = O(n^2)

II.
```
public int[] fix34(int[] nums) {
    int temp = 0;
    boolean three = false;
      for (int i = 0; i < nums.length-1; i++){                  //c.n
        if (nums[i] == 3)
```

```
                three = true;
              if (three == true){
                  three = false;
             if (nums[i+1] != 4){
             for (int j = 1; j < nums.length; j++){                    //c.n.n
                 if (nums[j] == 4 && nums[j-1] != 3){
                     temp = nums[i+1];
                nums[i+1] = nums[j];
               nums[j] = temp;
           }
       }
    }
  }
 }
 return nums;
}
```

$T(n) = c + c.n + c.n.n$
$T(n) = O(n^2)$

III.

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

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

T(n) = c+ c.n + c.n.n
T(n) =O(n^2)

IV.

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

T(n) = c + c.n + c.n.m
T(n) = O(n.m)

V.

```
public int[] seriesUp(int n) {
        int[] arr = new int[n*(n+1)/2];
       int p = 0;
       for(int i = 1; i <= n; i++){                  //c.n
          for(int j = 1; j <= i; j++, p++)           //c.n.n
             arr[p] = j;
       }
       return arr;
     }
```

T(n) = c + c.n + c.n.n
T(n) = O(n^2)

9.

1. *Array 1*
   N es nums.length
2. *Array 2*
   N es nums.length
   N es inner.length, M es outer.length

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

4) Simulacro de Parcial

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

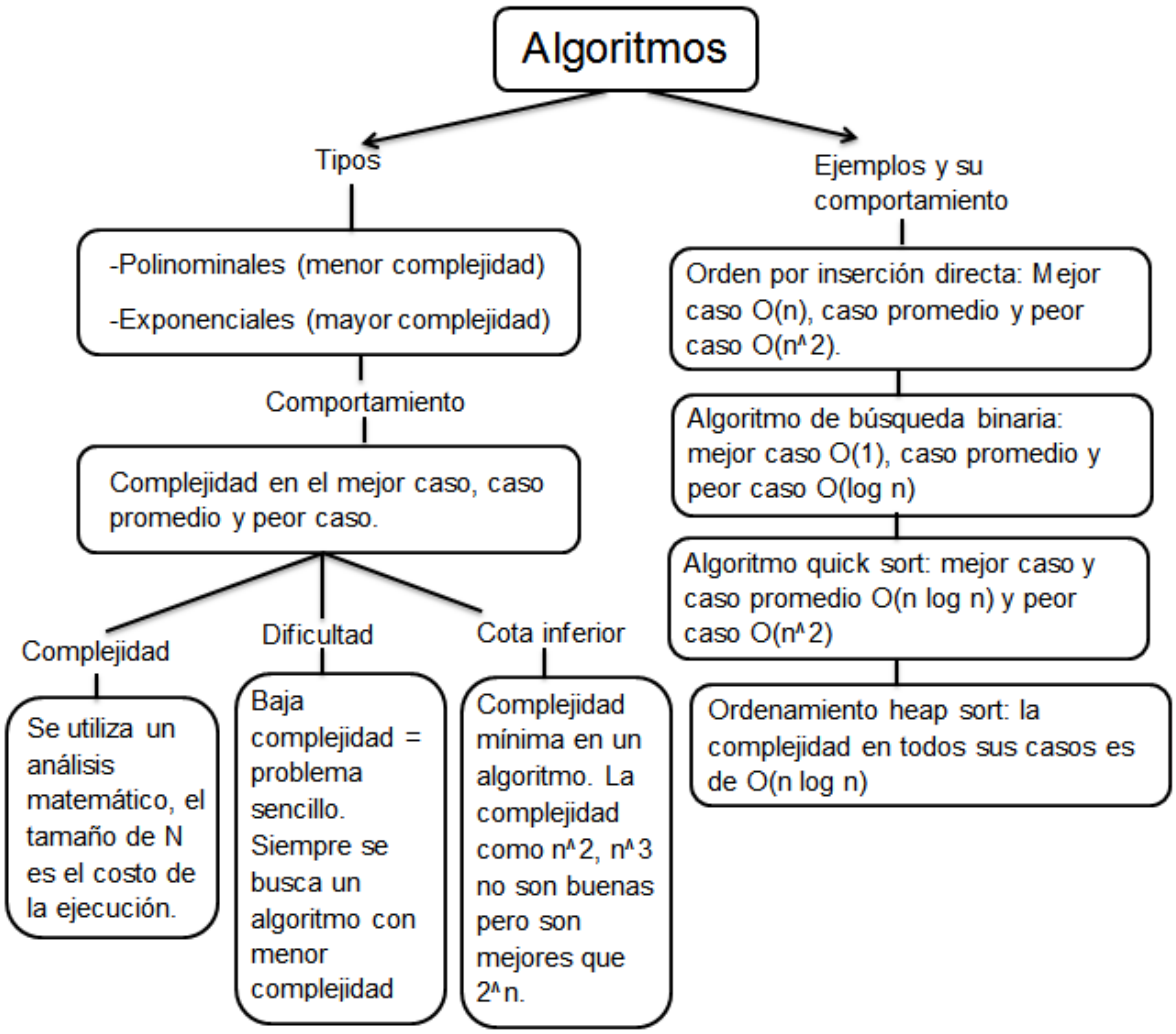
5. Lectura recomendada (opcional)

a) R.C.T Lee, Introducción al análisis y diseño de algoritmos, Capítulo 2. 2005

b) Un algoritmo es bueno según su eficiencia, es decir, su complejidad, que se haya con un análisis matemático donde el tamaño N depende de su ejecución. Es importante si se puede encontrar un algoritmo con menor orden de complejidad, pero también es claro que la complejidad como n^2, n^3 pueden ser no deseados pero siguen siendo tolerados a comparación de 2^n. La cota inferior de un problema es la complejidad temporal mínima requerida por cualquier algoritmo. Existen los algoritmos polinominales y los algoritmos exponenciales Para cualquier algoritmo se está interesado en su comportamiento en tres situaciones: mejor caso, caso promedio y peor caso. Algunos ejemplos de algoritmos son: el ordenamiento por inserción directa, búsqueda binaria, ordenamiento por selección directa, quik sort, kout sort y ordenamiento heap sort. Para medir la dificultad de un problema se dice que si este se puede resolver con un algoritmo de baja complejidad es sencillo.

c) Mapa de Conceptos

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



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