Laboratorio Nro. 2: Notación O grande

Isabela Muriel Roldán
Universidad Eafit
Medellín, Colombia
imurielr@eafit.edu.co

Mateo Flórez Restrepo
Universidad Eafit
Medellín, Colombia
mflorezr@eafit.edu.co

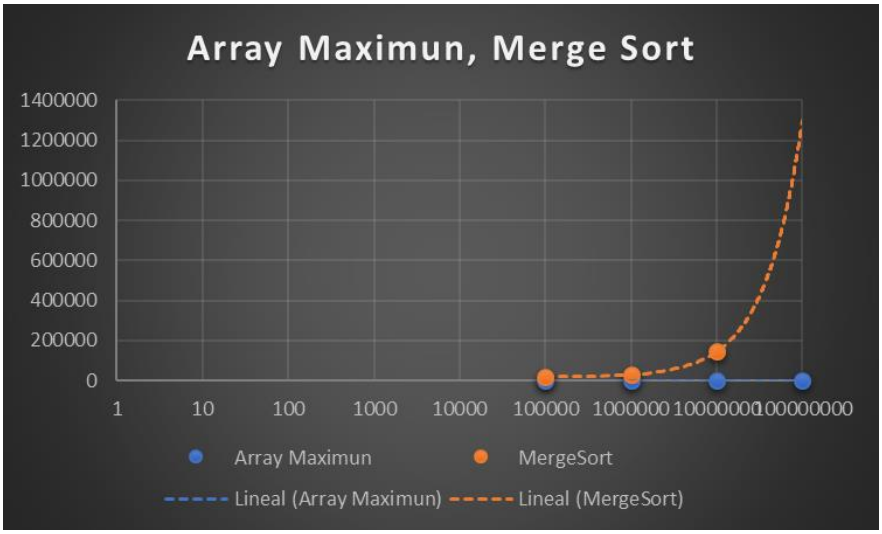
3) Simulacro de preguntas de sustentación de Proyectos

1.

	N=100.000	N=1’000.000	N=10’000.000	N=100’000.000
Array sum	Mas de 5 min	Mas de 5 min	Mas de 5 min	Mas de 5 min
Array maximun	11	17	39	47
Insertion sort	126628	Mas de 5 min	Mas de 5 min	Mas de 5 min
Merge Sort	19483	28368	145123	Mas de 5 min

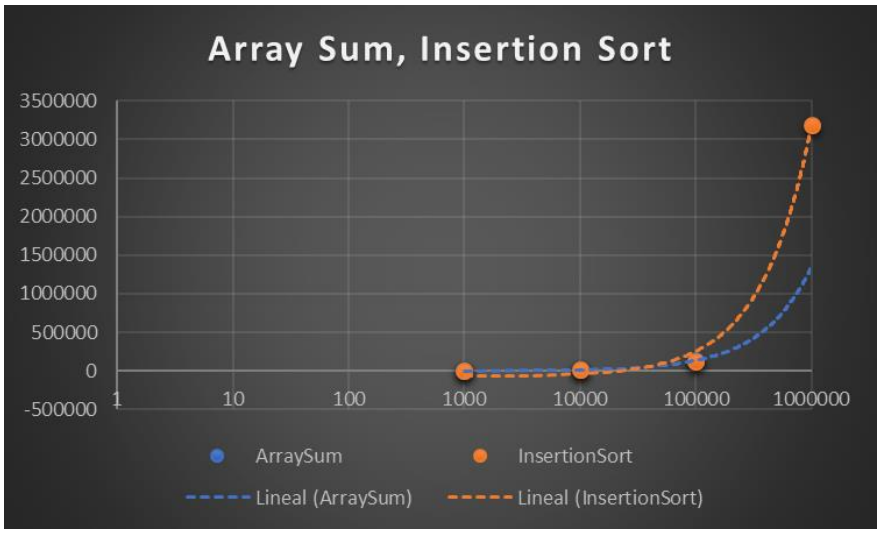
2.

	N=100.000	N=1’000.000	N=10’000.000	N=100’000.000
Array maximun	11	17	39	47
Merge Sort	2385	28368	145123	Mas de 1 min



N	1.000	10.000	100.000	1'000.000
Array Sum	1509	12800	134426	
InsertionSort	1316	12340	126628	3185376

Nota: Fueron evaluados con valores más pequeños ya que su complejidad es más alta al momento de ejecutar n de valores muy grandes.



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

3. Conclusión de resultados

Según los tiempos obtenidos en el laboratorio se puede apreciar la coherencia tanto en los datos como en las gráficas representativas frente a los resultados teóricos dados por la notación O (su complejidad), aunque pueda parecer confuso ya que no todos se evaluaron bajo los mismos parámetros, pues debido a que sus complejidades son diferentes unos algoritmos pueden tardar más tiempo en ejecutar valores n más grandes que otros.

Array Sum	O(n)
Array Maximum	O(n)
Insertion Sort	O(n^2)
Merge Sort	O(nlog(n))

***Tabla 1**. Resultados teóricos de Notación O de los algoritmos.*

Los resultados teóricos coinciden con las gráficas que representan cierta complejidad, sin embargo, en Array Maximum es difícil verla claramente, ya que, al momento de evaluarlo, aun con valores muy grandes arrojaba valores de tiempo muy pequeños, pero no constantes. Quizás al ser evaluado con valores más grandes se pueda notar mejor su complejidad.

4. Insertion Sort

La notación O del algoritmo “Insertion sort” contiene la complejidad más grande según la jerarquía de complejidad respecto a los otros algoritmos del laboratorio, la cual es O(n^2), por cierta razón habría más dificultad y poca eficiencia al calcular valores N demasiados grandes, pues tardaría demasiado tiempo en ejecutar la cantidad de acciones necesarias para generar un resultado. Además, este algoritmo recorre todas las posiciones de un arreglo y lo ordena según el orden que haya posteriormente respecto a los demás elementos, y mientras más elementos haya, será más difícil llegar al resultado.

5. Array Sum

Este algoritmo es más eficiente al momento manejar valores grandes de N, pues la cantidad de acciones que tiene que hacer para arrojar el resultado son mínimas relativamente a otro tipo de algoritmos, debido a que tiene un complejidad simple y manejable para valores muy grandes, O (n), por esta razón los tiempos de ejecución no crecen tanto como los de insertion sort al momento

de ejecutar un N completamente grande, pues n^2 > n , además es posible que haya mayor complejidad en el tiempo de ejecución en un algoritmo de ordenamiento que en uno de suma de elementos (ambos referidos a los arreglos de grandes valores de n).

6. ¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos grandes?

Merge sort es más eficiente en cuanto arreglos con una cantidad grande de elementos, esta es más utilizada para manejarlos debido a que su complejidad es menor frente a la del algoritmo de insertion sort (nlog(n) < n^2), y esto toma mucha importancia cuando la n toma valores realmente grandes, pues ambos son algoritmos de ordenamiento, sin embargo, insertion sort tiende a crecer demasiado en cuanto a su tiempo de ejecución, haciéndolo más inestable e ineficiente cuando se trata de ciertos valores n. Además, el algoritmo merge sort se encarga de dividir y mezclar sucesivamente esos grandes valores, hasta el punto en que sea más simple de ordenar el arreglo, evitando recorrer todas las posiciones como lo hace insertion sort.

¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos pequeños?

En el caso de arreglos con una cantidad pequeña elementos la eficiencia cambia a Insertion sort, merge sort también es eficiente, pero relativamente a un N de valores pequeños Insertion sort tiende a ser más veloz en cuanto el tiempo de ejecución, y sus pasos para llegar al resultado son menos. En este caso al usar merge sort sería muy innecesario dividir y mezclar valores pequeños sucesivamente hasta llegar a algo más simple, mientras que haciendo el insertion sort solo tendría que hacer el recorrido de ordenamiento y al ser un valor n pequeño de elementos no tardaría su ejecución y los pasos se simplificarían.

7. Max Span

El span según la descripción del ejercicio es la cantidad de elementos que hay entre dos posiciones del arreglo, incluyendo los valores de ciertas posiciones, pero esto pasa siempre y cuando los valores de los limites, tanto izquierda como derecha, sean iguales. El ejercicio lo que hará básicamente es verificar entre dos valores iguales, la cantidad de elementos que haya y devolverá el valor de la cantidad máxima que encuentre. En caso de que no haya elementos iguales en el arreglo, devolverá 1. Ejemplo:

Entrada:

1	4	2	1	4	1	4
Nº de elementos:	1	2	3	4	5	6

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

Salida (maxSpan): 6

8. Complejidad de ejercicios en línea.

NOTACION O EJERCICIOS CODINGBAT:

Array2:

- public int countEvens(int[] nums){

```
int cont = 0                            //C1
for(int i=0; i<nums.length; ++i){       //C2*n
      if(nums[i]%2 == 0){               //C3*n
            ++coun;                     //C4*n
      }
}
return cont;
}
```

T(n) = C1+(C2+C3+C4)*n
T(n) = O(C1+C'*n)
T(n) = O(C'*n)
T(n) = O(n)

- public int sum13(int[] nums){

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

T(n) = C1+C2+C7+C8+(C3+C4+C5+C6)*n
T(n) = O(C'+C''*n)
T(n) = O(C'*n)
T(n) = O(n)

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

- public int matchUp(int[] nums1, int[] nums2){

```
int cont = 0;                                                          //C1
for(int i=0; i<nums1.length; ++i){                                     //C2*n
      if(nums[i]!=nums[2[i]){                                          //C3*n
         if((nums1[i]-nums2[i]<=2 && nums2[i]-nums1[i]<=2){   //C4*n
               ++cont                                                  //C5*n
         }
      }
}
return cont;                                                           //C6
}
```

T(n) = C1+C6+(C2+C3+C4+C5)*n
T(n) = O(C'+C''*n)
T(n) = O(C''*n)
T(n) = O(n)

- public boolean isEverywhere(int[] nums, int val){

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

T(n) = C4+(C1+C2+C3)*n
T(n) = O(C+C'*n)
T(n) = O(C'*n)
T(n) = O(n)

- public int[] evenOdd(int[] nums){

```
int temp = 0;                                          //C1
for(int i=0; i<nums.length; ++i){                      //C2*n
      for(int j=0; j<nums.length; ++j){                //C3*n
            if(nums[i]%2==0){                          //C4*n
                  temp=nums[i];                        //C5*n
                  nums[i]=nums[j];                     //C6*n
                  nums[j]=temp;                        //C7*n
            }
      }
}
return nums;                                           //C8
}
```

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

T(n) = C1+C8+(C2+C3+C4+C5+C6+C7)*n
T(n) = O(C+C'*n)
T(n) = O(C'*n)
T(n) = O(n)

Array 3

- public int maxSpan(int[] nums) {

```
if (nums.length > 0) {                                  //C1
    int maxSpan = 1;                        //C2
    for (int i = 0; i < nums.length; i++){              //C3*n
        for (int j = nums.length - 1; j > i; j--){      //(C4*n)n
            if (nums[j] == nums[i]) {                   //C5*n^2
                int count = (j - i) + 1;                //C6*n^2
                if (count > maxSpan){                   //C7*n^2
                    maxSpan = count;                    //C8*n^2
                    break;
                }
                }
            }
        }
     return maxSpan;                                     //C9
    } else  {
        return 0;                                       //C10
    }
}
```

T(n)= C1+C2+C9+C10+C3*n+(C4+C5+C6+C7+C8)*n^2
T(n)= O(C+C'n+C"n^2)
T(n)= O(C"n^2)
T(n)= O(n^2)

- public static int[] fix34(int[] nums) {

```
for(int i = 0; i < nums.length; ++i ){              //C1*n
    if (nums[i]==3){                                //C2*n
        int temp = nums[i+1];                       //C3*n
        nums[i+1]= 4;                               //C4*n
        for(int j=i+2; j <nums.length; ++j){        //(C5*n) n
            if(nums[j]==4){                         //C6*n^2
                nums[j] = temp;                     //C7*n^2
            }
        }
    }
}
    return nums;                                    //C8
}
```

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

T(n)= (C1+C2+C3+C4)*n+(C5+C6+C7)*n^2+C8
T(n)= O(C'n+ C"n^2+C)
T(n)= O(C"n^2)
T(n)= O(n^2)

- public int[] fix45(int[] nums) {

```
for (int i = 0; i < nums.length; i++){                    //C1*n
    if (nums[i] == 5 && i == 0                            //C2*n
            || nums[i] == 5 && nums[i - 1] != 4) {
      int pos5 = i;                                       //C3*n
      for (int j = 0; j < nums.length; j++){              //(C4*n)n
         if (nums[j] == 4 && nums[j + 1] != 5) {          //C5*n^2
            int temp = nums[j + 1];                       //C6*n^2
            nums[j + 1] = 5;                              //C7*n^2
            nums[pos5] = temp;                            //C8*n^2
            break;
         }
      }
     }
    }
    return nums;                                          //C9
}
```

T(n)= (C1+C2+C3)*n + (C4+C5+C6+C7+C8)*n^2+C9
T(n)= O(C'n+C"n^2+C)
T(n)= O(C"n^2)
T(n)= O(n^2)

- public boolean canBalance(int[] nums) {

```
for (int i=0; i < nums.length; ++i){                      //C1*n
    int sum =0;                                           //C2*n
    for(int j=0; j<i ; ++j){                              //(C3*n)n
       sum+=nums[j];                                      //C4*n^2
    }
    for(int k=i; k < nums.length; ++k){                   //(C5*n)n
       sum-=nums[k];                                      //C6*n^2
    }
    if (sum==0)                                           //C7*n
    return true;                                          //C8*n
  }
  return false;                                           //C9
}
```

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

T(n)= (C1+C2)*n+(C3+C4+C5+C6+C7+C8)*n^2+C9
T(n)= O(C'n+C"n^2+C)
T(n)= O(C"n^2)
T(n)= O(n^2)

- public boolean linearIn(int[] outer, int[] inner) {

```
int indexInner = 0;                                                  //C1
int indexOuter = 0;                                                  //C2
while (indexInner < inner.length && indexOuter < outer.length) {     //C3*n^2
    if (outer[indexOuter] == inner[indexInner]) {                    //C4*n^2
        indexOuter++;                                                //C5*n^2
        indexInner++;                                                //C6*n^2
    } else indexOuter++;                                             //C7*n^2
}
return (indexInner == inner.length);                                 //C8
}
```

T(n)= C1+C2+C8+(C3+C4+C5+C6+C7)*n^2
T(n)= O(C+C'n^2)
T(n)= O(C'n^2)
T(n)= O(n^2)

9. n y m

En los cálculos de la complejidad anterior, estas variables se refieren a la cantidad de veces que el algoritmo tiene que ejecutar algún paso o acción al momento de compilarse para llegar al resultado esperado. En los ejercicios anteriores usamos en la gran mayoría solo la variable n ya que en los códigos solo se trabajaba con un solo tipo de entrada igual para cada caso. La variable m la usaríamos si hubiera más de una entrada para poder diferenciar el alcance de la complejidad de los códigos.

4) Simulacro de Parcial

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