Laboratorio Nro. 2: Notación O grande

Luis Carlos Rodríguez Zúñiga
Universidad Eafit
Medellín, Colombia
lcrodriguz@eafit.edu.co

Laura Sánchez Córdoba
Universidad Eafit
Medellín, Colombia
lsanchezc@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1.

ArraySum

datos en el array	Tiempo en nanosegundos
10000	165611
100000	132955
1000000	1380402
10000000	22140542

ArrayMax

datos en el array	Tiempo en nanosegundos
10000	172142
100000	142286
1000000	1497496
10000000	15225938

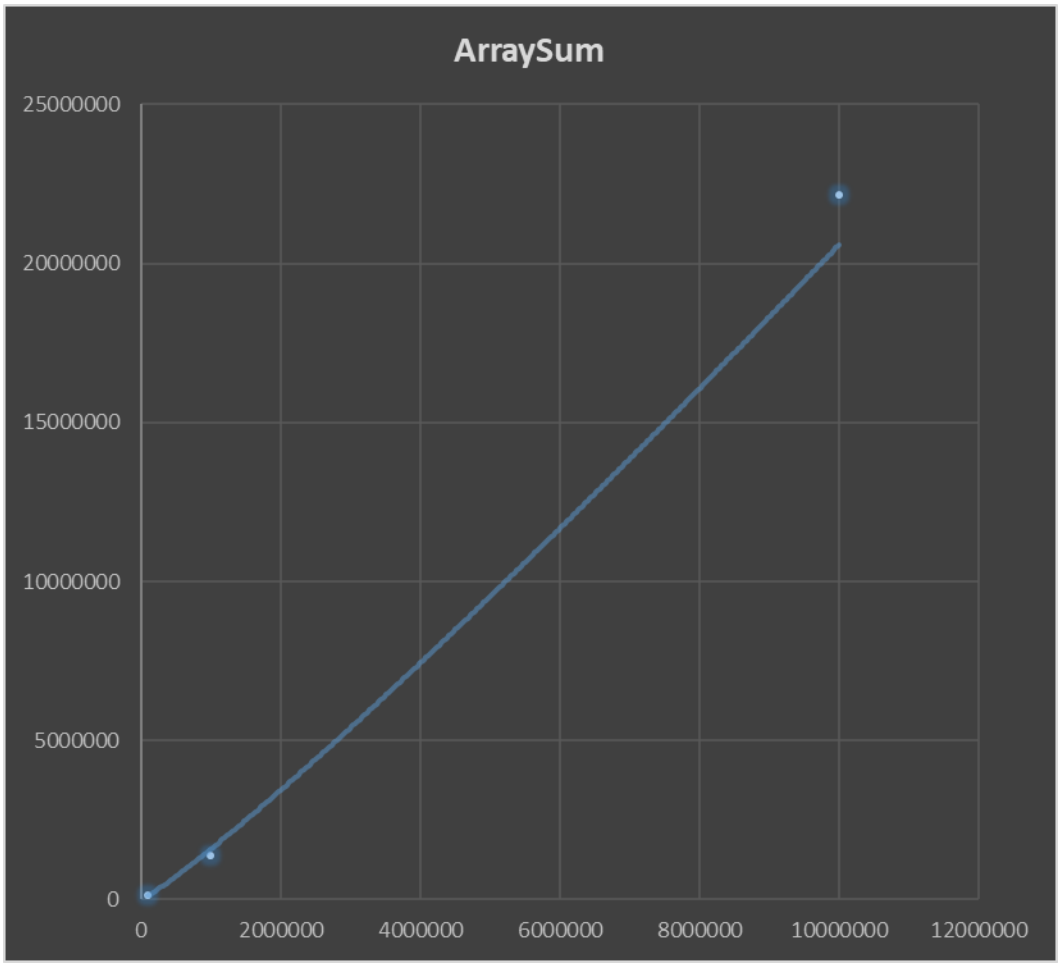
InsertionSort

datos en el array	Tiempo en nanosegundos
1000	28448683
10000	433351885
100000	11023583922
1000000	7,58983E+11

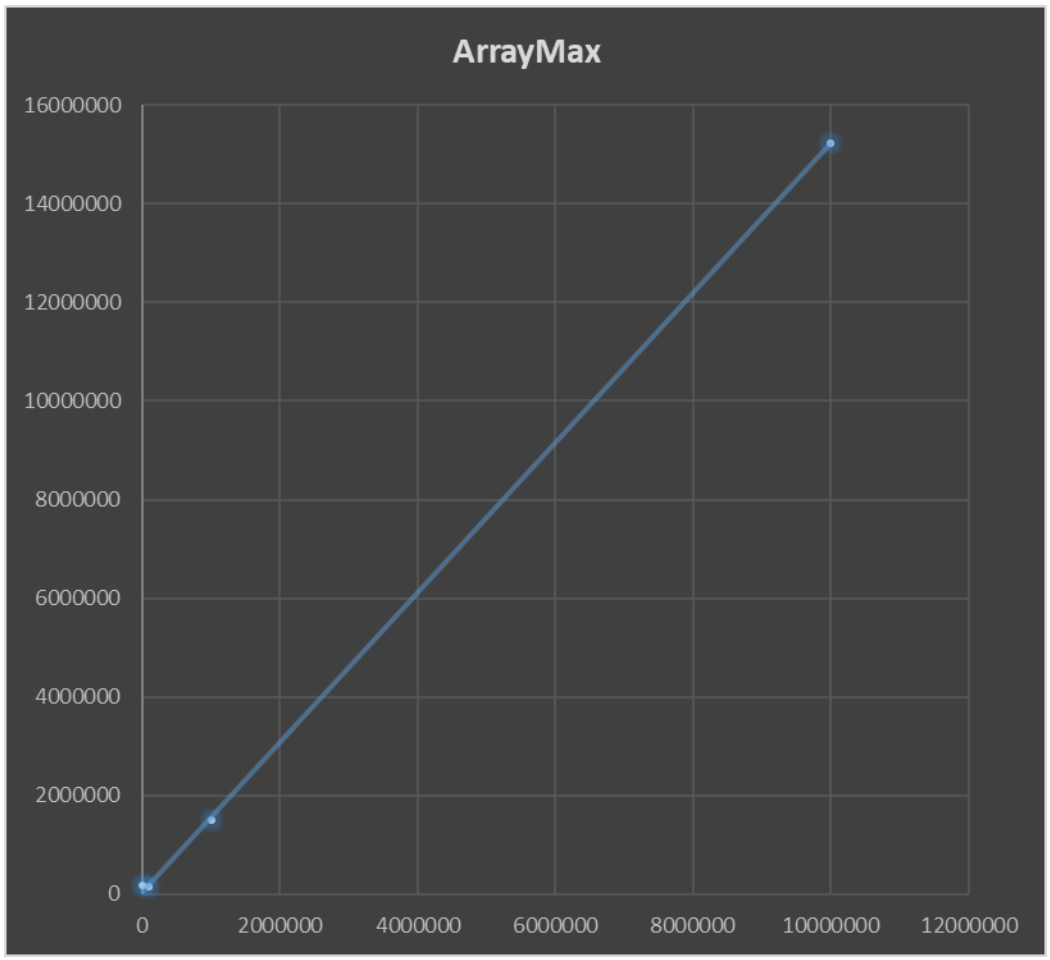
MergeSort

datos en el array	Tiempo en nanosegundos
1000	751546
10000	2505623
100000	23978589
1000000	388403226
10000000	3285212254

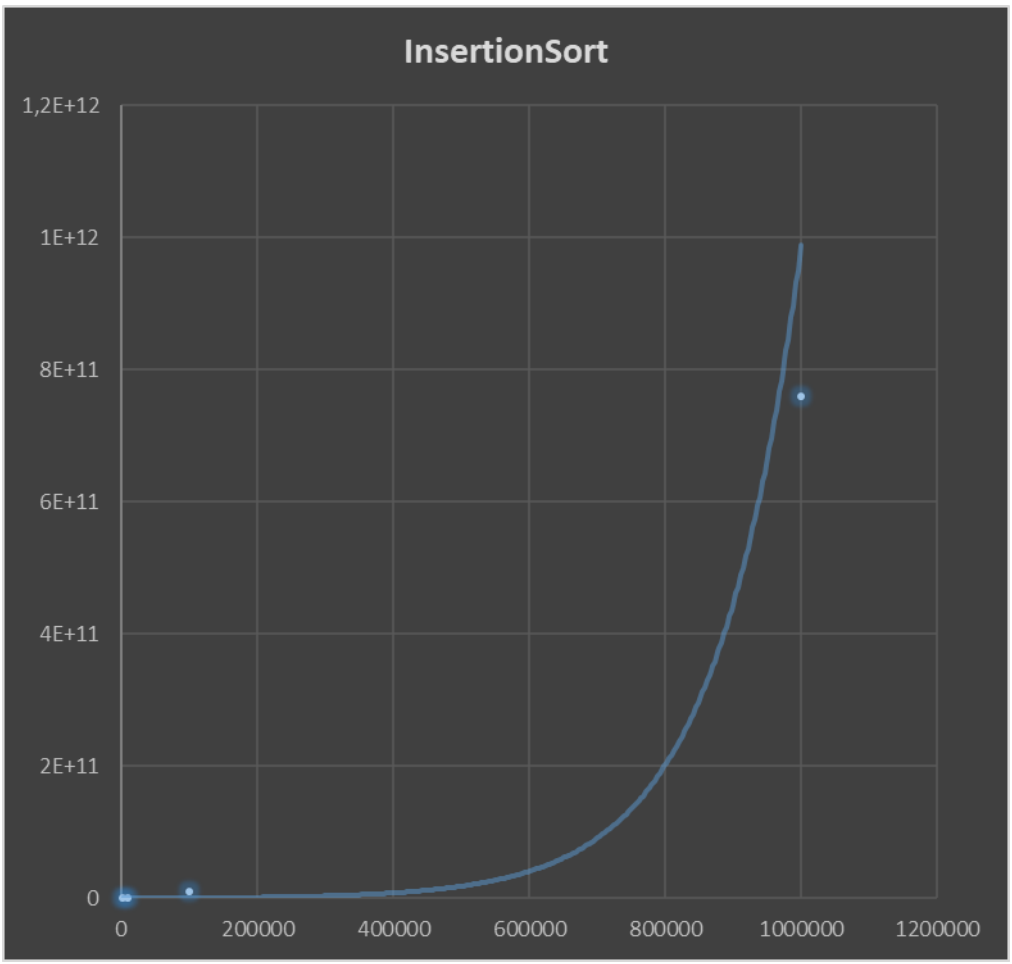
2.



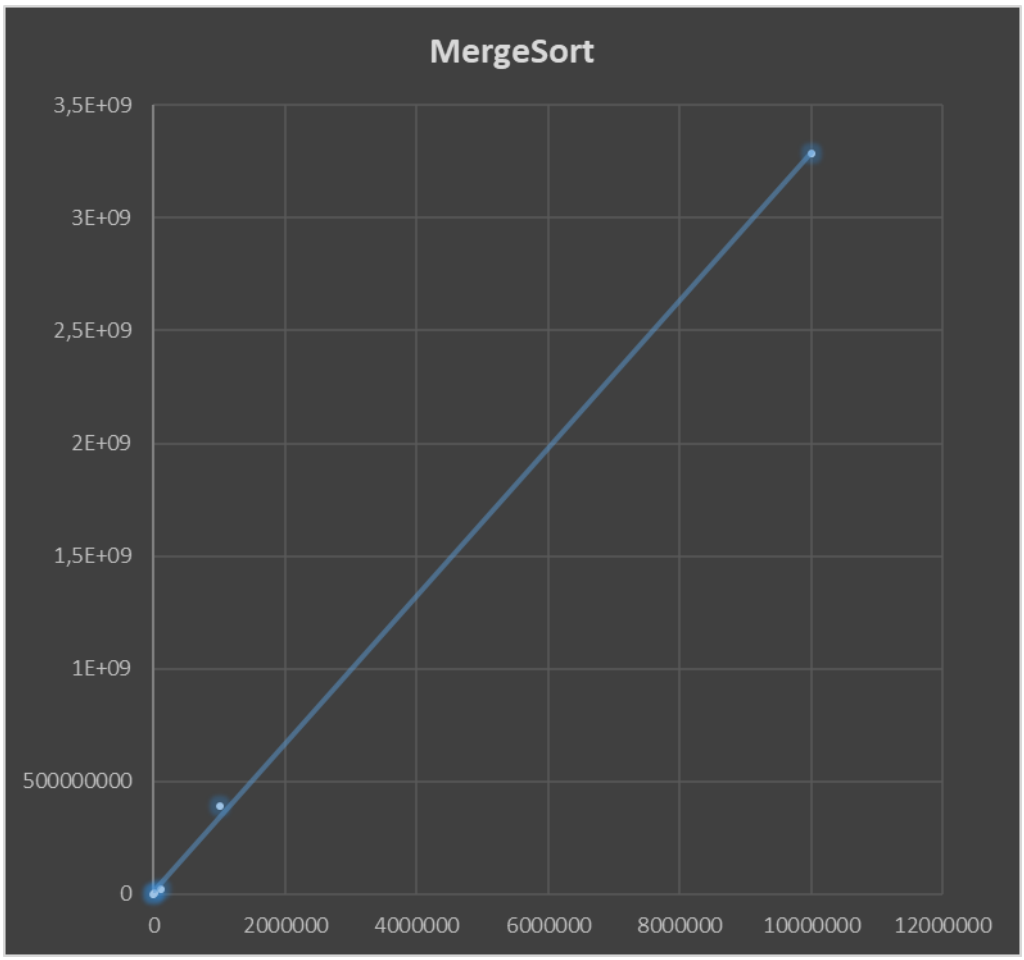
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co



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



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



3. Pueden variar según el cálculo de las constantes las cuales dependerán de los procesos y servicios que esté desarrollando el procesador (independientes de programa) y la cantidad de datos que sea capaz de analizar el procesador de la máquina, por esto habrán maquinas que ejecuten el programa más rápido que otras que hasta el tiempo sea difícil de diferenciar en milisegundos los cuales es mejor tomar en nanosegundo y las gráficas (las cuales se esperan que sean en log) pueden variar de los factores anteriores.

4. Empecemos que el algoritmo de InsertionSort se encarga de recibir un array y de recorrerlo organizando cada dato de manera creciente (< a >), al realizar este procedimiento con un array con una cantidad de 100.000.000 de datos, donde cada dato está dentro del rango 0 – 500.000.000, va a necesitar demasiado tiempo analizar cada dato y decidir en qué ubicación asignarlo,

esto implica realizar una mayor cantidad de cálculos por segundos en el procesador (teniendo en cuenta que se realizó en una maquina con una velocidad de 2.2 GHz en el procesador), esto implica un crecimiento exponencial en el tiempo relacionado a la cantidad de datos.

5. En el algoritmo de ArraySum se puede denotar un tiempo de O(n), por lo tanto el tiempo que se tome en ejecutar va a ser dependiente de la cantidad de datos que se encuentre en el array, por lo tanto ejecutarlo con valores demasiado grandes no va a ser problema, pero, en comparación con el algoritmo InsertionSort en el cual el tiempo es de O(n*n) donde los tiempos crecen demasiado respecto a la cantidad de datos, en comparación es más rápido sumar datos de un array que organizarlo (ya sea de < a > o > a <).

6. Primero hablemos de los dos algoritmos de ordenamiento, mergeSort se encarga de dividir un array en dos hasta llegar a una mínima expresión y empezar a ordenar de menor a mayor su tiempo es de O(n log n) es útil y eficiente para array de artos datos. Por el otro lado tenemos al algoritmo InsertionSort el cual consiste de recorrer el arreglo y organizando cada dato de menor a dato, es eficiente para array de bajos datos ya que su tiempo es de O(n log n) al comienzo su crecimiento en tiempo es casi despreciable pero al momento de darle como parámetro un array de una longitud mayor a 1’000,000 se tomara demasiado tiempo. En comparación de los dos algoritmos es mas eficiente con pocos datos el mergeSort.

7. MaxSpan pretende hallar la cantidad de números que hay entre la ocurrencia más a la izquierda y más a la derecha de un número y funciona con dos ciclos que no están anidados que se encargan de: en el caso del primero, comparar si un número en la posición i es igual al de la última posición y en caso se ser afirmativo, le asigna a un contador el número de posiciones entre éstos (incluidos los extremos) y se sale del ciclo para entrar al segundo ciclo. El segundo ciclo funciona de manera inversa, comparando la primera posición con la de la variable i que parte desde la última y si encuentra una ocurrencia de el mismo número a la izquierda y a la derecha; le asigna a otro contador el valor de i+1 (la posición en la que va del arreglo +1 para incluir el extremo). Finalmente se comparan ambos contadores, para ver cual el mayor Span (cantidad de números entre los extremos iguales) y éste es el que devuelve el programa Y funciona así porque con ambos ciclos permiten que dado el caso de que haya más de un caso en el que se hallen 2 extremos iguales, se pueda comparar cuál de éstos es el mayor.

8.
Array 2

CountEvens:

```
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
    cont++;    //c4
  return cont;
}
```

T(n)=c1+c2*n+c3+c4

Notación O
O(n)=c1+c2*n+c3+c4
O(n)=c2*n R.S
O(n)=n R.P

FizzArray:

```
public int[] fizzArray(int n) {
        int[] a = new int[n];       //c1
        for (int i=0 ; i < a.length; i++) {       //c2*n
            a[i]=i;      //c3
        }
        return a;
    }
```

T(n)=c1+c2*n+c3

Notación O
O(n)=c1+c2*n+c3
O(n)=c2*n R.S
O(n)=n R.P

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

Has22

```
public boolean has22(int[] n) {
        boolean r=false;     //c1
        for (int i = 0; i < n.length-1; i++){     //c2*n
            if (n[i] == 2 && n[i + 1] == 2)      //c3
                r=true;     //c4
        }
        return r;
    }
```

T(n)=c1+c2*n+c3+c4

Notación O
O(n)=c1+c2*n+c3+c4
O(n)=c2*n R.S
O(n)=n R.P

modThree

```
public boolean modThree(int[] n) {
        boolean r=false;     //c1
        for (int i = 0; i < n.length -2; i++) {        //c2*n
            if(n[i]%2==0 && n[i+1]%2==0 && n[i+2]%2==0)      //c3
                r=true;       //c4
            if(n[i]%2==1 && n[i+1]%2==1 && n[i+2]%2==1)       //c5
                r=true;     //c6
        }
        return r;
    }
```

T(n)=c1+c2*n+c3+c4+c5+c6

Notación O
O(n)=c1+c2*n+c3+c4+c5+c6
O(n)=c2*n R.S
O(n)=n R.P

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

TripleUp

```
public boolean tripleUp(int[] n) {
      boolean r=false;    //c1
      for(int i=0;i<n.length-2;i++){     //c2*n
         if(n[i+1]==n[i]+1 && n[i+2]==n[i]+2)       //c3
         r=true;  //c4
      }
      return r;
   }
```

T(n)=c1+c2*n+c3+c4

Notación O
O(n)=c1+c2*n+c3+c4
O(n)=c2*n R.S
O(n)=n R.P

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

Array 3

CountClumps

Título: CodingBat: Java. Array-3
Autor: Gregor Ulm
Fecha: Marzo 2013
Disponible: http://gregorulm.com/codingbat-java-array-3/

```
public int countClumps(int[] nums) {

    int count = 0;       //c1
    for (int i = 0; i < nums.length - 1; i++)      //c2*n
        if (nums[i] == nums[i + 1]) {          //c3
            count++;     //c4
            for (int j = i + 1; j< nums.length; j++)      //c5*n
                if (nums[j] == nums[i])     //c6
                    i++;     //c7
                else break;    //c8
        }
    return count;
}
```

T(n)= c+c2*n*(c5*n)

Notación O
O(n)= c+c2*n*(c5*n)
O(n)= c2*n*(c5*n) R.S
O(n)= n*(c5*n) R.P
O(n)= n*n R.P
O(n)= n^2

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

SquareUp:

Título: Java. Array-3
Autor: Mateo Agudelo
Fecha: 11 de septiembre de 2017

```
public int[] squareUp(int n) {
  int k = 0;    // c1
  int[] res = new int[n*n];      //c2
  for(int i = 1; i <= n; ++i)     //n*c3
    for (int j = n; j >= 1; --j)    //n*c4
      res[k++] = j <= i ? j : 0;  //c5
  return res;
}
```

T(n)= c1+c2+c3*n*(c4*n)+c5

Notación O
O(n)= c1+c2+c3*n*(c4*n)+c5
O(n)= c3*n*(c4*n) R.S
O(n)= n*(c4*n) R.P
O(n)= n*n R.P
O(n)= n^2

LinearIn

Titulo: CodingBat: Java. Array-3
Autor: Gregor Ulm
Fecha: Marzo 2013
Disponible: http://gregorulm.com/codingbat-java-array-3/

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

    int indexInner = 0;     //c1
    int indexOuter = 0;    //c2
    while (indexInner < inner.length && indexOuter < outer.length) {      //c3*n+c4*n
        if (outer[indexOuter] == inner[indexInner]) {    //c5
            indexOuter++;   //c6
            indexInner++;   //c7
        } else indexOuter++;   //c8
    }
    return (indexInner == inner.length);
}
```

T(n)= c1+c2+c3*n+(c4*n)+c

Notación O
O(n)= c1+c2+c3*n+(c4*n)+c
O(n)= c3*n+(c4*n) R.S
O(n)= n*(c4+c3)
O(n)= n R.P
O(n)= n

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

MaxSpan

Titulo: CodingBat: Java. Array-3
Autor: Gregor Ulm
Fecha: Marzo 2013
Disponible: http://gregorulm.com/codingbat-java-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
                if (nums[j] == nums[i]) {           //c5
                    int count = (j - i) + 1;    //c6
                    if (count > maxSpan) maxSpan = count;   //c7
                    break;
                }
        return maxSpan;
    } else return 0;
}
```

T(n)= c1+c2+c3*n*(c4*n)+c

Notación O
O(n)= c1+c2+c3*n*(c4*n)+c
O(n)= c3*n*(c4*n) R.S
O(n)= n*(c4*n) R.P
O(n)= n*n R.P
O(n)= n^2

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

SeriesUp:

Titulo: CodingBat: Java. Array-3
Autor:Gregor Ulm
Fecha:Marzo 2013
Disponible:http://gregorulm.com/codingbat-java-array-3/

```
public int[] seriesUp(int n) {
    int[] result = new int[n * (n + 1) / 2];   //c1
    int pos = 0;                               //c2
    for (int i=1;i<= n + 1;i++) {              //c3*n
        for (int j = 1; j< i; j++)             //c4*n
        result[pos++] = j;                     //c5

    }
    return result;

}
```

T(n)= c1+c2+c3*n*(c4*n)+c5

Notación O
O(n)= c1+c2+c3*n*(c4*n)+c5
O(n)= c3*n*(c4*n) R.S
O(n)= n*(c4*n) R.P
O(n)= n*n R.P
O(n)= n^2

9. Las variables n y m usadas en el cálculo de las complejidades (en el caso del punto previo, solo n) representan el tamaño de la entrada que se da en los parámetros del algoritmo y en función de la cual se mide la eficiencia de éste, pues permite ver la cantidad de tiempo que le tomará realizar cierta cantidad de pasos para emitir la respuesta correspondiente a la entrada.

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