Práctica de Laboratorio No. 2: Notación O

Nicolás Restrepo López
Universidad Eafit
Medellín, Colombia
nrestrepol@eafit.edu.co

Mateo Restrepo Sierra
Universidad Eafit
Medellín, Colombia
mrestrepos@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos.

1. **Extiendan el código existente para medir los tiempos de ArraySum, ArrayMax, InsertionSort y MergeSort para arreglos generados aleatoriamente con diferentes tamaños.**

   Véase Anexo: Laboratorio2.

   Algoritmo para MergeSort basado en el video Algorithms: Merge Sort *por* HackerRank: *https://www.youtube.com/watch?v=KF2j-9iSf4Q&t=490s*

2.1 **Resuelvan mínimo 5 ejercicios del nivel Array 2 de la página CodingBat: http://codingbat.com/java/Array-2.**

Estos ejercicios fueron realizados en la cuenta de CodingBat mateo.restrepo20171g062@hotmail.com. *Véase Anexo: Array2.*

2.2 Resuelvan mínimo 5 ejercicios del nivel Array 3 de la página CodingBat: http://codingbat.com/java/Array-3.

Estos ejercicios fueron realizados en la cuenta de CodingBat mateo.restrepo20171g062@hotmail.com. *Véase Anexo: Array3.*

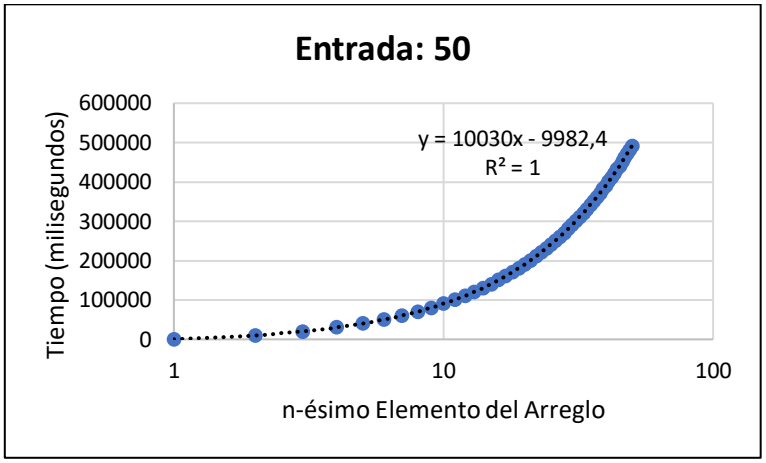
3.1 En la vida real, para analizar la eficiencia de los algoritmos de forma experimental, debemos probar con problemas de diferente tamaño. El tamaño del problema lo denominamos N. Como un ejemplo, para un algoritmo de ordenamiento, el tamaño del problema es el tamaño del arreglo. Como otro ejemplo, para calcular la serie de Fibonacci, N es el número de términos a calcular. Teniendo en cuenta lo anterior, completen la siguiente tabla con tiempos en milisegundos.

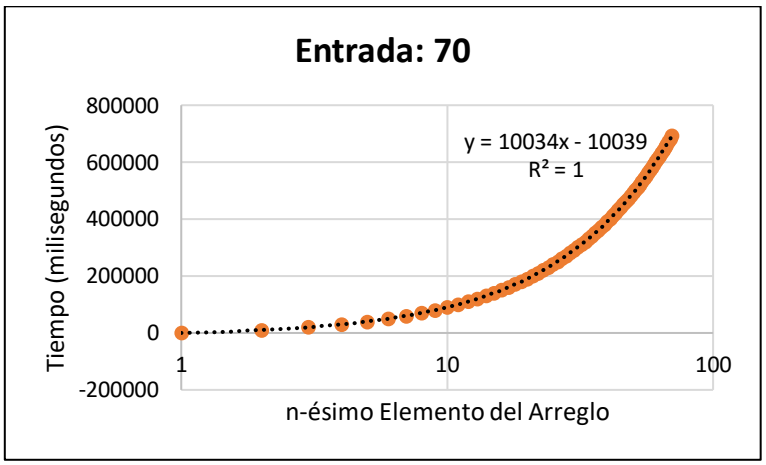
	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
ArraySum	1.003.290.112	10.032.990.112	100.329.990.112	1.003.299.990.112
ArrayMax	1.003.200.049	10.032.000.049	100.320.000.049	1.003.200.000.049
InsertionSort	50.152.502.461.325	5.015.295.024.601.320	501.529.950.246.001.000	50.152.999.502.460.000.000
MergeSort	2.006.590.064	20.065.990.064	200.659.990.064	2.006.599.990.064

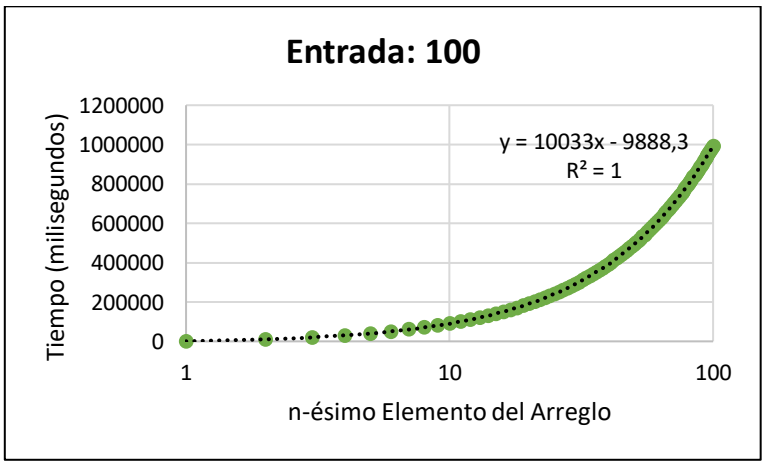
Los valores anteriores se obtuvieron mediante la extrapolación utilizando la ecuación de las gráficas en 3.2 con entradas de 100 elementos para todos los casos. Nótese que la complejidad de MergeSort resulta ser lineal y no logarítmica. Se hablará de eso en 3.3.

3.2 Grafiquen los tiempos para ArraySum, ArrayMax, InsertionSort y MergeSort con entradas de diferentes tamaños.

ArraySum

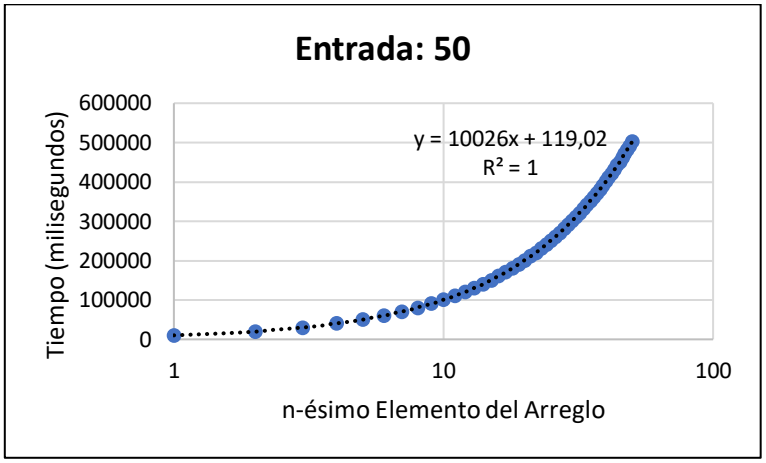


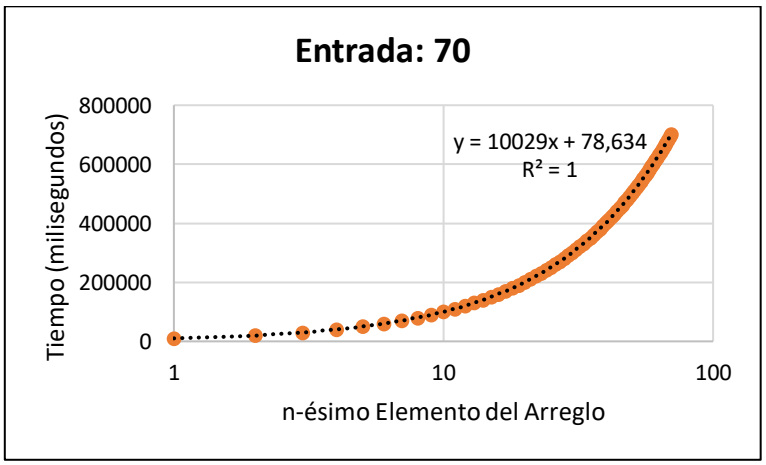


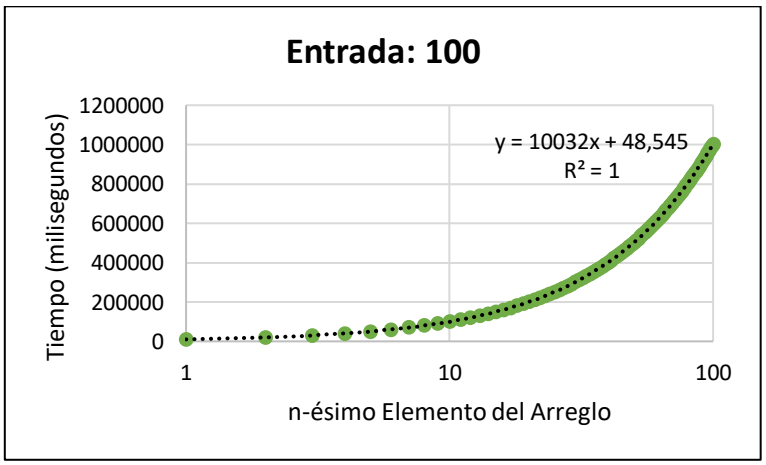


DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

ArrayMax

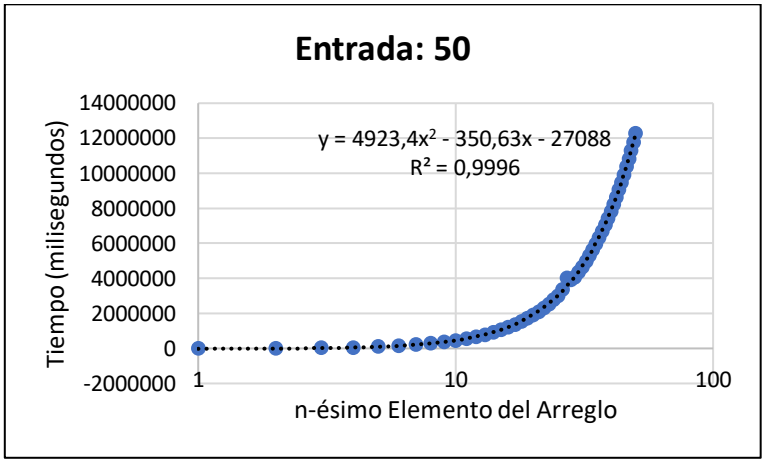


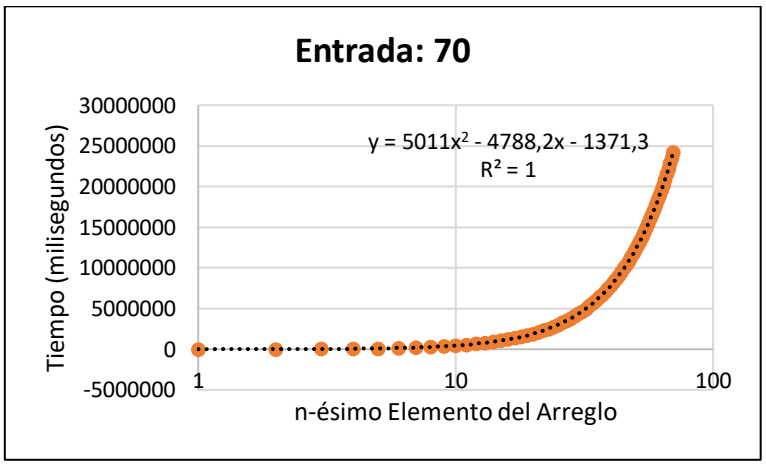


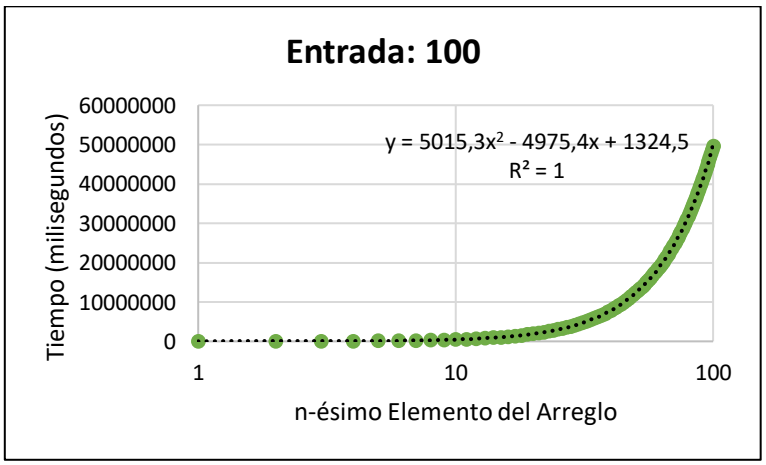


DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

InsertionSort

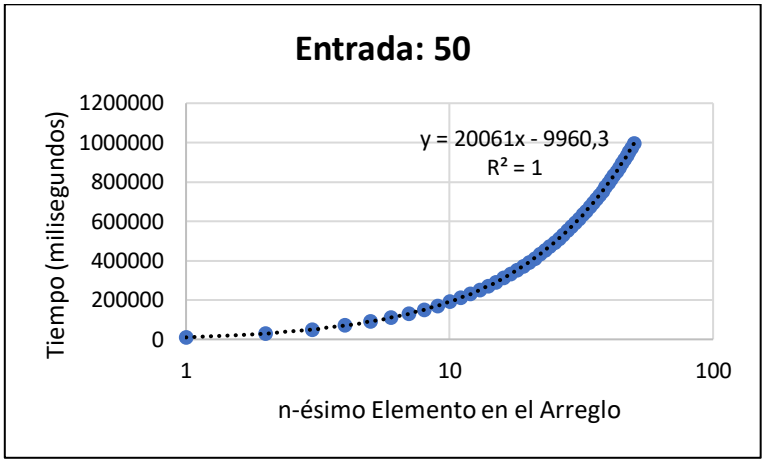


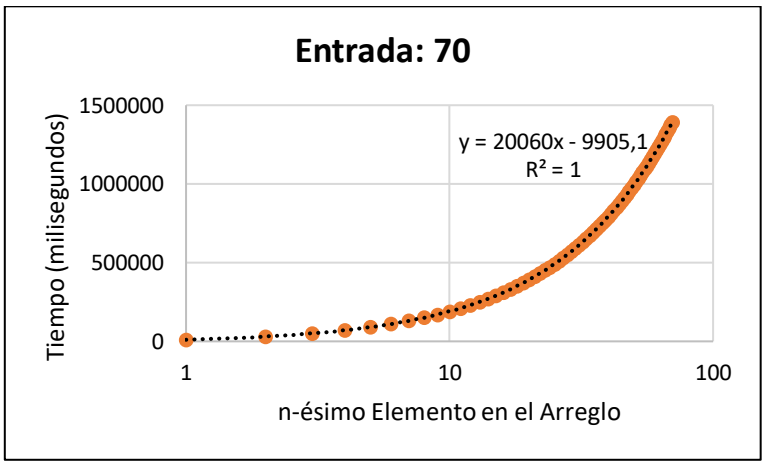


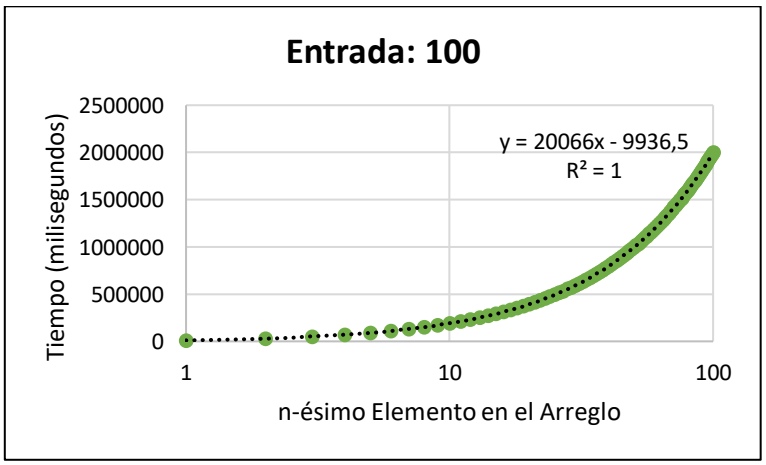


DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

MergeSort







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

3.3 ¿Qué concluyen con respecto a los tiempos obtenidos en el laboratorio y los resultados teóricos obtenidos con la notación O?

Los tiempos obtenidos en el laboratorio tienen un comportamiento muy similar a las complejidades reales en cada algoritmo, sin importar las diferentes entradas. Podemos notarlo cuando hacemos la regresión y observamos que los valores R^2 siempre son cercanos o iguales a 1, lo que indica que las diferentes ecuaciones obtenidas describen casi perfectamente cada complejidad. Para ArraySum y ArrayMax se determinó que para ambos algoritmos su complejidad es O (n), para InsertionSort es O (n^2) y, algo que no es correcto pero que igualmente no pudimos determinar el porqué del resultado, para MergeSort es O(n). En este último caso los tiempos tomados se comportaron de manera lineal cuando debieron hacerlo de manera logarítmica. Sin embargo, no fue posible arreglar este problema.

3.4 ¿Qué sucede con InsertionSort para valores grandes de N?

La complejidad del algoritmo InsertionSort es O (n^2), lo cual indica que esta aumentará de manera polinómica. Dicho esto, para valores grandes de N los tiempos serán muy grandes y el algoritmo se torna poco eficiente para arreglos de gran tamaño. Para arreglos pequeños su eficiencia es aceptable.

3.5 ¿Qué sucede con ArraySum para valores grandes de N? ¿Por qué los tiempos no crecen tan rápido como InsertionSort?

La complejidad del algoritmo ArraySum es O (n), es decir que es lineal. Por ello, los tiempos no crecerán de una manera tan pronunciada como sí lo hacen en InsertionSort, tanto para valores pequeños de N como para valores grandes. ArraySum tendrá siempre la misma sin importar el tamaño del arreglo.

3.6 ¿Qué tan eficiente es MergeSort con respecto a InsertionSort para arreglos grandes? ¿Qué tan eficiente es MergeSort con respecto a InsertionSort para arreglos pequeños?

Primero se analizará en base a los resultados obtenidos. Estos nos indicaron que la complejidad de MergeSort es lineal, motivo por el cual siempre será más eficiente que InsertionSort sin importar en tamaño del arreglo, pues desde el inicio la complejidad del segundo crecerá más aceleradamente.

Ahora, teniendo en cuenta que la complejidad teórica de MergeSort es O (n log n) bajo el principio de *divide y conquistarás* [1], este algoritmo será más eficiente que InsertionSort para arreglos grandes, puesto que se llega a un punto en el

cual la complejidad crece de manera menos pronunciada y los tiempos de ejecución se tornar mucho más cortos con respecto a los de la otra alternativa. Ahora, para arreglos cortos MergeSort no supone una alternativa eficiente dado que su complejidad es muy alta al inicio, con un crecimiento pronunciado de los tiempos de ejecución hasta determinado valor. Esto indica que InsertionSort será más eficiente para arreglos pequeños.

[1] Tomado de *https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort*

3.7 Expliquen con tus propias palabras cómo funciona el ejercicio maxSpan y por qué.

MaxSpan es un método que recibe como parámetro un arreglo de enteros y lo que hace básicamente es coger la aparición más a la izquierda y más a la derecha de algún número en el arreglo y contar cuantos números hay entre ellos contándose ellos mismos, esto es denominado el span. Acá podemos ver el código del ejercicio:

```
public int maxSpan (int [] nums) {
   int span = 0;
   int maxSpan = 0;
      for (int i = 0; i < nums.length; i++){
         span = 0;
         for (int j = i; j < nums.length; j++){
            span++;
            if (nums[i] == nums[j] && span > maxSpan)
               maxSpan = span;
         }
      }
   return maxSpan;
}
```

Se crea un primer ciclo (línea 4) para poder escoger un primer elemento del arreglo y compararlo con el resto del arreglo a través del segundo ciclo (línea 6). La idea con el primer ciclo es poner un elemento fijo y revisar todos los que hay después de él. Siempre se le va sumando de uno en uno al span (línea 7) y luego se revisa si el número que se tiene fijo es igual al número cambiante por el segundo ciclo (línea 8). Además, si el número que se tiene como span es mayor al máximo span, se pone como máximo al span que se tenía (línea 9). Es importante decir que luego del primer ciclo se debe inicializar en cero el span (línea 5), ya que, si no se hace esto, se estarían tomando en cuenta los valores generados anteriormente por otro número.

3.8 Calculen la complejidad de los numerales 2.1 y 2.2.

CountEvens (Array 2)

```
public int countEvens(int [] nums){
   int count = 0;                                  //C1
      for (int i = 0; i < nums.length; i++){       //C2 + C3*n
         if (nums[i] % 2 == 0)
            count++;                               //C4*n
      }
   return count;                                   //C5
}
```

T(n) = C1 + C2 + C3*n + C4*n + C5
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), por la Ley de la Suma.
T(n) = O (n), por la Ley del Producto.

BigDiff (Array2)

```
public int bigDiff (int [] nums){
   int max = nums[0];                              //C1
   int min = nums[0];                              //C2
      for (int i = 0; i < nums.length; i++){       //C3 + C4*n
         if (nums[i] > max)                        //C5
            max = nums[i];                         //C6*n
         if (nums[i] < min)                        //C7
            min = nums[i];                         //C8*n
     }
     return max - min;                             //C9
}
```

T(n) = C1 + C2 + C3 + C4*n + C5 + C6*n + C7 + C8*n + C9
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), por la Ley de la Suma.
T(n) = O (n), por la Ley del Producto.

Has22 (Array2)

```
public boolean has22 (int [] nums){
   for (int i = 0; i < nums.length - 1; i++){     //C1 + C2*n
      if (nums[i] == 2 && nums[i + 1] == 2)       //C3*n
         return true;                             //C4*n
   }
   return false;                                   //C5*n
}
```

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*n + C4*n
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), por la Ley de la Suma.
T(n) = O (n), por la Ley del Producto

Sum28 (Array2)

```
public boolean sum28 (int [] nums){
   int sum = 0;                                        //C1
      for (int i = 0; i < nums.length; i++){           //C2 + C3*n
          if (nums[i] == 2)                            //C4
      sum += nums[i];                                  //C5*n + C6*n
   }
      if (sum == 8)                                    //C7
         return true;                                  //C8
   return false;                                       //C9
}
```

T(n) = C1 + C2 + C3*n + C4 + C5*n + C6*n + C7 + C8 + C9
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), por la Ley de la Suma.
T(n) = O (n), por la Ley del Producto.

FizzArray (Array2)

```
public int [] fizzArray (int n){
   int [] arr = new int [n];              //C1
      for (int i = 0; i < n; i++){        //C2 + C3*n
         arr[i] = i;                      //C4*n
      }
   return arr;                            //C5
}
```

T(n) = C1 + C2 + C3*n + C4*n + C5
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), por la Ley de la Suma.
T(n) = O (n), por la Ley del Producto.

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

MaxSpan (Array3)

```
public int maxSpan (int [] nums){
   int span = 0;                                          //C1
   int maxSpan = 0;                                       //C2
   for (int i = 0; i < nums.length; i++){                 //C3 + C4*n
      span = 0;                                           //C5*n
      for (int j = i; j < nums.length; j++){              //C6*n*n
         span++;                                          //C7*n*n
         if (nums[i] == nums[j] && span > maxSpan)        //C8*n*n
            maxSpan = span;                               //C9*n*n
      }
   }
   return maxSpan;                                        //C10
}
```

T(n) = C1 + C2 + C3 + C4*n + C5*n + C6*n*n + C7*n*n + C8*n*n + C9*n*n + C10
T(n) = C' + C''*n + C*n^2
T(n) = O (C' + C''*n + C*n^2)
T(n) = O (C*n^2), por la Ley de la Suma.
T(n) = O (n^2), por la Ley del Producto.

Fix34 (Array3)

```
public int [] fix34 (int [] nums){
   for (int i = 0; i < nums.length; i++){                 //C1 + C2*n
      if (nums[i] == 3) {                                 //C3*n
         int temporal = nums[i + 1];                      //C4*n
         nums[i + 1] = 4;                                 //C5*n
         for (int j = i + 2; j < nums.length; j++){       //C6*n + C7*n*n
            if (nums[j] == 4)
            nums[j] = temporal;                           //C8*n*n
         }
      }
   }
   return nums;                                           //C9
}
```

T(n) = C1 + C2*n + C3*n + C4*n + C5*n + C6*n + C7*n*n + C8*n + C9
T(n) = C' + C''*n + C*n^2
T(n) = O (C' + C''*n + C*n^2)
T(n) = O (C*n^2), por la Ley de la Suma.
T(n) = O (n^2), por la Ley del Producto.

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

Fix45 (Array3)

```
public int [] fix45 (int [] nums){
   for (int i = 0; i < nums.length; i++){                          //C1 + C2*n
      if (nums[i] == 5 && i == 0 || nums[i] == 5 && nums[i - 1] != 4){
                                                                   //C3*n
         int p5 = i;                                               //C4*n
         for (int j = 0; j < nums.length; j++){                    //C5*n*n
            if (nums[j] == 4 && nums[j + 1] != 5){                 //C6*n*n
               int temporal = nums[j + 1];                         //C7*n*n
               nums[j + 1] = 5;                                    //C8*n*n
               nums[p5] = temporal;                                //C9*n*n
               break;                                              //C10*n*n
            }
         }
      }
   }
   return nums;                                                    //C11
}
```

T(n) = C1 + C2*n + C3*n + C4*n + C5*n*n + C6*n*n + C7*n*n + C8*n*n + C9*n*n + C10*n*n + C11
T(n) = C' + C''*n + C*n^2
T(n) = O (C' + C''*n + C*n^2)
T(n) = O (C*n^2), por la Ley de la Suma.
T(n) = O (n^2), por la Ley del Producto.

Can Balance (Array3)

```
public boolean canBalance (int [] nums){
   int acumulador = 0;                                  //C1
   for (int i = 0; i < nums.length; i++){               //C2 + C3*n
      int acumulador2 = 0;                              //C4*n
      for (int e = i; e < nums.length; e++){            //C5*n*n
         acumulador2 += nums[e];                        //C6*n*n
      }
      if (acumulador == acumulador2)                    //C7*n
         return true;                                   //C8*n
   acumulador += nums[i];                               //C9*n
   }
   return false;                                        //C10
}
```

T(n) = C1 + C2*n + C4*n + C5*n*n + C6*n*n + C7*n + C8*n + C9*n + C10
T(n) = C' + C''*n + C*n^2
T(n) = O (C' + C''*n + C*n^2)
T(n) = O (C*n^2), por la Ley de la Suma.
T(n) = O (n^2), por la Ley del Producto.

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

LinearIn (Array3)

```
public boolean LinearIn (int [] outer, int [] inner){
   int n = 0;                                          //C1
   if (inner.length != 0){                             //C2
      for (int i = 0; i < outer.length; i++){          //C3 + C4*n
         if (outer[i] == inner[n]){                    //C5*n
            n++;                                       //C6*n
            if (n == inner.length)                     //C7*n
               break;                                  //C8*n
         }
      }
   }
   return n == inner.length;                           //C9
}
```

T(n) = C1 + C2 + C3 + C4*n + C5*n + C6*n + C7*n + C9
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n^2), por la ley de la suma.
T(n) = O (n^2), por la ley del producto.

3.9 Expliquen con sus palabras las variables del cálculo de las complejidades del numeral 3.8.

En el cálculo de la complejidad de un algoritmo existen ciertas variables que juegan el papel de las entradas de los mismos. En los casos anteriores eran la n, y esta representa un parámetro para un algoritmo. Este parámetro a su vez es una función y también sirve para clasificar los algoritmos según su eficiencia. Por ejemplo, los O(n) o los O(n^2) y así. La diferencia que se tiene con los términos C es que estos son constantes, que ya están establecidas para cada algoritmo, y como lo que se quiere es clasificar y comparar un algoritmo frente a otro, es por esto que se menosprecian estas constantes, ya que para grandes valores sí afectarían la eficiencia de un algoritmo, pero no significativamente.

4) Simulacro de Parcial.

1. Supongamos que P (n, m) es una función cuya complejidad asintótica es O (n × m) y H (n, m) es otra función cuya complejidad asintótica es O (m*A (n, m)), donde A (n, m) es otra función. ¿Cuál de las siguientes funciones podría ser la complejidad asintótica de A (n, m) si tenemos en cuenta que P (n, m) > H (n, m) para todo n y para todo m?

a) O (log n)
b) O (√n)
c) O (n + m)
d) O (1)

2. Dayla sabe que la complejidad asintótica de la función P(n) es O(√n). Ayúdale a Dayla a sacar la complejidad asintótica para la función mistery (n, m).

```
01  void mistery (int n, int m){
02     for (int i = 0; i < m; ++i){
03        boolean can = P(n);
04        if(can)
05           for (int i = 1; i * i <= n; i++){
06              //Hacer algo en O (1)
07           }
08        else
09           for (int i = 1; i <= n; i++){
10              //Hacer algo en O (1)
11           }
12      }
13  }
```

La complejidad de mistery (n, m) es:

a) O (m + n)
b) O (m×n×√n)
c) O (m + n + √n)
d) O (m×n)

3. El siguiente algoritmo imprime todos los valores de una matriz. El tamaño de la matriz está definido por los parámetros largo y ancho. Teniendo en cuenta que el largo puede ser como máximo 30 mientras que el ancho puede tomar cualquier valor, ¿cuál es su complejidad asintótica en el peor de los casos?

```
01  public void matrices (int [] [] m, int largo, int ancho)
02     for (int i = 0; i < largo; i++)
03        for (int j = 0; j < ancho; j++)
04           print (m [i][j]);
05  }
```

a) O (largo)
b) O (ancho)
c) O (largo + ancho)
d) O (ancho×ancho)
e) O (1)

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

4. Sabemos que P(n) ejecuta n3 + n pasos y que D(n) ejecuta n+7 pasos. ¿Cuál es la complejidad asintótica, en el peor de los casos, de la función B(n)?

```
01   public int B (int n)
02      int getP = P(n);                    // C1 + n^3 +n
03      int ni = 0;                         // C2
04      for (int i = 0; i < n; ++i)         // C3 + C4*n
05         if(D(i) > 100)                   // C5*n(n+7)
06            ni++;                         // C6*n
07      int nj = getP + D(n) * ni;          // C7 + n^3 +n + C8(n + 7 pasos)
08   return nj;                             // C9
```

//T(n) = C1 + n^3 +n + C2 + C3 + C4*n + C5*n(n+7) + C6*n + C6*n + C7 + n^3 +n + C8(n+7 pasos) + C9
//T (n) = C''' + C'*n + C''*n2 + C*n3
//T (n) = C*n^3, por la Ley de la Suma
//T (n) = O(n^3), por la Ley del Producto

a) O (n^4)
b) O (n^3)
c) O (n^2)
d) O (2^n)

5. El siguiente algoritmo calcula las tablas de multiplicar del 1 a n. ¿Cuál es su complejidad asintótica en el peor de los casos?

```
01   public void tablas (int n)
02      for (int i = 0; i < n; i++)                  //C1 + C2*n
03         for (int j = 0; j < n; j++)               //n*(C3 + C4*n)
04            print (i + "*" + j + "=" + i*j);       //C5*n*n
```

//T(n) = C1 + C2*n + n*(C3 + C4*n) + C5*n*n
//T(n) = O (C'' + C'*n + C*n^2)
//T(n) = O(C*n^2), por la Ley de la Suma.
//T(n) = O(n^2), por la Ley del Producto.

a) O (n)
b) O (2^j)
c) O (i)
d) O (n^2)

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