Laboratorio Nro. 2
Complejidad de algoritmos

Sebastian Castaño Orozco
Universidad Eafit
Medellín, Colombia
scasta31@eafit.edu.co

Dennis Castrillón Sepúlveda
Universidad Eafit
Medellín, Colombia
dcastri9@eafit.edu.co

1) Simulacro de proyecto

Se procede a realizar el código para los algoritmos InsertionSort y MergeSort, posteriormente se miden los tiempos necesarios para ejecutar los algoritmos. Mas adelante se profundiza en el tema con las tablas y gráficas.

2) Ejercicios en línea

Se procede a realizar los ejercicios en línea en la página CodingBat para los módulos Array2 y Array 3. Mas adelante, en el numeral 3.5 se explican estos códigos en el cálculo de la complejidad en el tiempo de estos. Además, los archivos .py se adjuntan en Github.

3) Simulacro de preguntas de sustentación de Proyectos

3.1 Tabla de valores de tiempo para los algoritmos InsertionSort y MergeSort.

Insertion Sort: Se miden los tiempos para el ordenamiento de arreglos entre 1000 y 20000 valores con paso 1000. Es decir, se generan valores aleatorios y se construyen con estos valores vectores de tamaño n, donde n va entre 1000 y 20000 haciendo pasos de a 1000, obteniendo 20 vectores de distintos tamaños y tomando el tiempo requerido por el algoritmo para organizar estos valores.

Cabe resaltar que se toman tiempos para el peor de los casos, es decir, para vectores organizados de manera descendente, para ello, luego de crear los vectores con valores aleatorios, se organiza el vector de manera descendente y se procede a ingresar este vector como parámetro de la función InsertionSort. A continuación, la tabla de tiempos para el algoritmo.

Merge Sort: Se miden los tiempos para el ordenamiento de arreglos entre 1 y 1’000.000 valores con paso 50000. Es decir, se generan valores aleatorios y se construyen con estos valores vectores de tamaño n, donde n va entre 1 y 1’000.000 haciendo pasos de a 50000, obteniendo 20 vectores de distintos tamaños y tomando el tiempo requerido por el algoritmo para organizar estos valores.

Cabe resaltar que se toman tiempos para el peor de los casos, es decir, para vectores organizados de manera descendente, para ello, luego de crear los vectores con valores aleatorios, se organiza el vector de manera descendente y se procede a ingresar este vector como parámetro de la función MergeSort. A continuación, la tabla de tiempos para el algoritmo.

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

InsertionSort	
Longitud arreglo	Tiempo
0	0
1000	0.418377638
2000	1.483290672
3000	3.439148903
4000	6.094449043
5000	9.219362259
6000	15.56599402
7000	22.20002031
8000	28.50035214
9000	34.79069018
10000	44.09825444
11000	50.48008966
12000	58.71095276
13000	71.10073924
14000	86.49638414
15000	101.0855658
16000	128.669385
17000	142.9270442
18000	150.9171081
19000	187.1285534

Tabla 1. Tiempos para InsertionSort.

MergeSort		MergeSort	
Longitud arreglo	Tiempo	Longitud arreglo	Tiempo
1	0	950001	20.16118383
50001	0.749676466	1000001	21.45750546
100001	1.798583269	1050001	21.99884129
150001	2.661397934	1100001	23.40415478
200001	3.629322767	1150001	24.25082731
250001	4.744282484	1200001	25.72020078
300001	5.840384007	1250001	27.30828667
350001	6.896270752	1300001	28.11635637
400001	7.598854065	1350001	29.67776728
450001	8.888017893	1400001	31.82970929
500001	9.85691905	1450001	33.32164955
550001	10.86785007	1500001	35.80238891
600001	12.60634565	1550001	35.69419122
650001	13.91316366	1600001	35.65238261
700001	14.96847367	1650001	34.7565899
750001	15.47490978	1700001	35.46896656
800001	16.39778733	1750001	36.5646662
850001	17.14446378	1800001	36.456123
900001	18.51470327	1850001	38.5466412

Tabla 2. Tiempos para MergeSort.

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

3.2 Gráfica de tiempo de ejecución de los algoritmos

Se procede a graficar los tiempos para los algoritmos InsertionSort y MergeSort, de acuerdo con las tablas previas (Ver tablas 1 y 2). A continuación, se muestran las gráficas y su comportamiento.

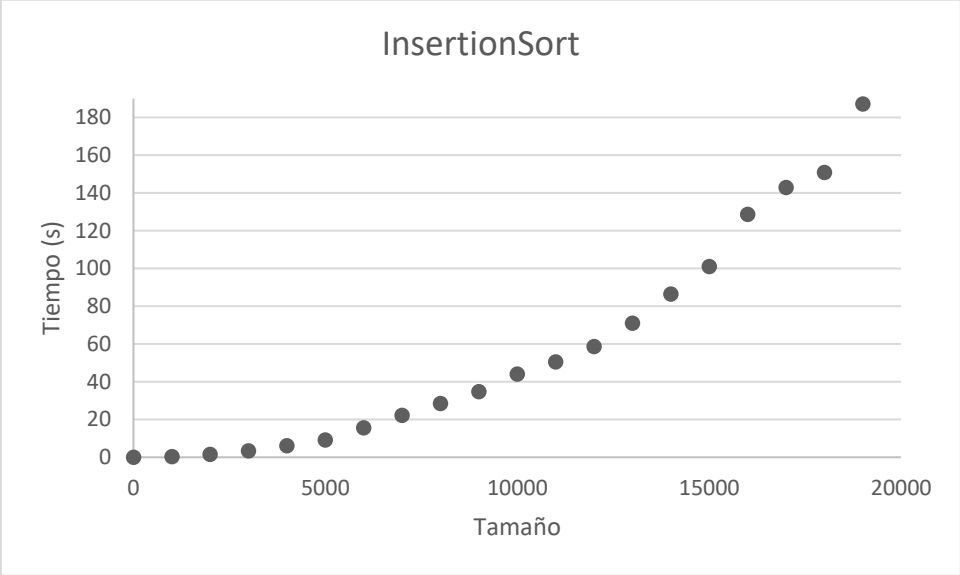


Figura 1. Comportamiento algoritmo InsertionSort

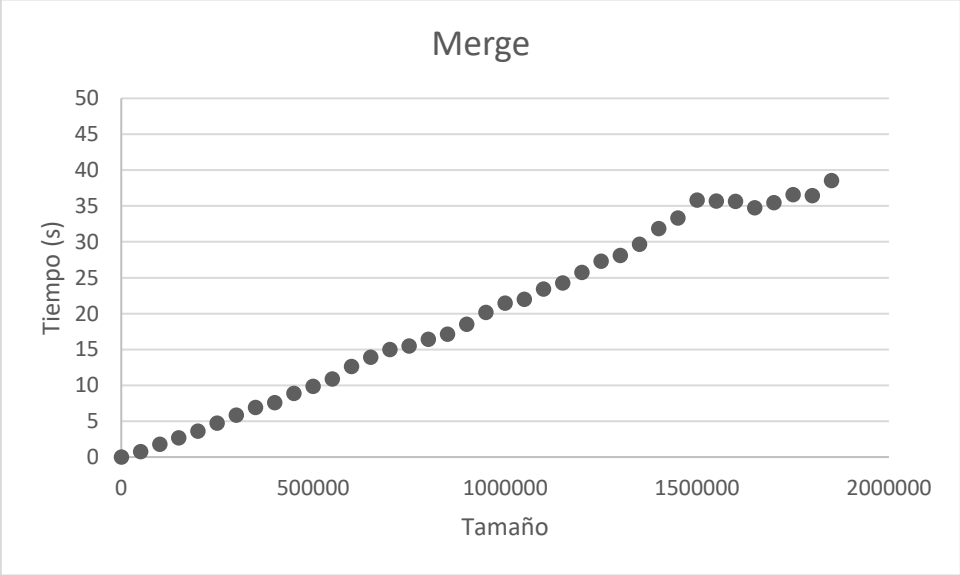


Figura 2. Comportamiento algoritmo MergeSort

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

3.3 Interpretación InsertionSort

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

R/ A continuación, se presenta el cálculo de complejidad del algoritmo InsertionSort.

```
def InsertionSort(array):
    for k in range (1,len(array)): # C_1 = n-1
        j=array[k] # C_2 = n-1
        i= k-1 # C_3 = n-1
        while i>=0 and j<array[i]:
                array[i+1]=array[i]
                array[i]=j
                i=i-1

    return(array)   # T(n)= a*(n^2)+b*n+c # O(n)= n^2
```

$$O(n) = n^2$$

Por tanto, usar el algoritmo de orden InsertionSort no sería apropiado para millones de elementos en tiempo real ya que posee un comportamiento cuadrático, por ende a medida que la longitud n (cantidad de elementos del array a ordenar) aumenta, el tiempo que requiere el algoritmo para ejecutarse incrementa a razón cuadrática, por lo que el manejo de muchos elementos sería muy complejo y demandaría mucho tiempo.

3.4 Interpretación complejidad logarítmica

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

R/ Para la complejidad asintótica de merge sort aparece un comportamiento logarítmico debido a la recursión y al bucle while presentes en el algoritmo. Por otra parte, el algoritmo tiende a estabilizarse en el tiempo después de que el arreglo que ingresa como parámetro

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

al proceso supere cierta cantidad de elementos, comportamiento característico de funciones logarítmicas.

A continuación, se presenta el calculo de la complejidad para el algoritmo MergeSort.

```
def mergeSort(array):
    if len(array)<=1:
        return (array) #C_1 = 4
    else:
        m = len(array)//2
        izq = array[:m]
        der = array[m:]
        mergeSort(izq) # T(n)=T(n/2) Recursión
        mergeSort(der) # T(n)=T(n/2) Recursión
        i=j=k=0
        while i < len(izq) and j < len(der): # T(n)= n*C
            if izq[i] < der[j]: # T(n)= n/2
                array[k]=izq[i]
                i=i+1
            else:
                array[k]=der[j]
                j=j+1
            k=k+1

        while i < len(izq):
            array[k]=izq[i]
            i=i+1
            k=k+1

        while j < len(der):
            array[k]=der[j]
            j=j+1
            k=k+1


        return (array)
```

$$T(n) = 2T\left(\frac{n}{2}\right) + nC_1$$

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

$$T(n) = \left(\frac{C_1 n(\log(n))}{\log(2)}\right) + \frac{nC_1}{2}$$

3.7 Cálculo de la complejidad en el tiempo para los ejercicios en línea

A continuación, se presenta el calculo de complejidad para cada ejercicio y su respectiva gráfica que compara el tiempo tomado para ejecutar el algoritmo de acuerdo a la longitud n del arreglo.

3.7.1 Ejercicios Array-2

Ejercicio 1: BigDiff

```
def bigDiff(n):
    maxi=n[0]
    mini=n[0] #Tn=C1=2
    if len(n)==0:
        maxi=0
        mini=0 #Tn=C2=5
    else:
        for i in range(0, len(n)):
            if maxi<n[i]:
                maxi=n[i]
            if mini>n[i]:
                mini=n[i]# Tn=C3*n, C3=8
    return maxi-mini #Tn=C4=2
```

$$T(n) = n * C_3 + C_4 + C_1$$

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

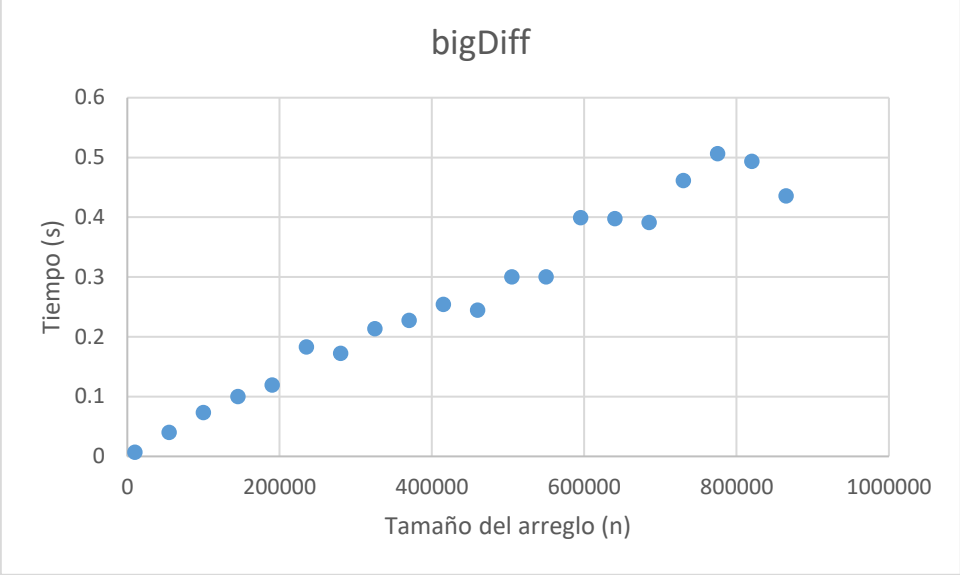


Figura 3. Comportamiento algoritmo bigDiff

Ejercicio 2: countEvents

```
def contarPares(n):
    cont=0 #C1=1
    for i in range(0,len(n)):
        if n[i]%2==0:
            cont=cont+1 #Tn=C2*n, C2=6
    return cont #Tn=C3=1
```

$$T(n) = n * C_2 + C_3 + C_1$$

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

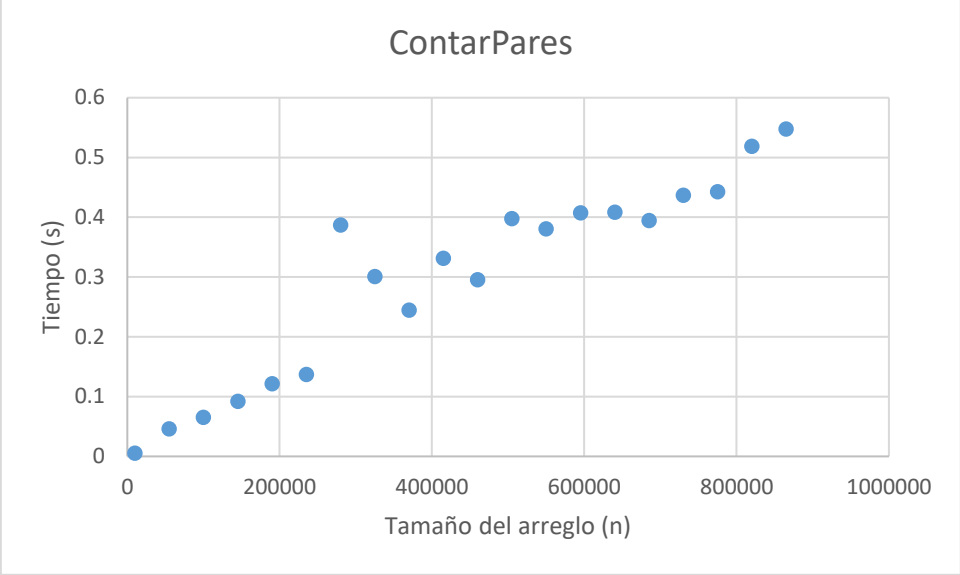
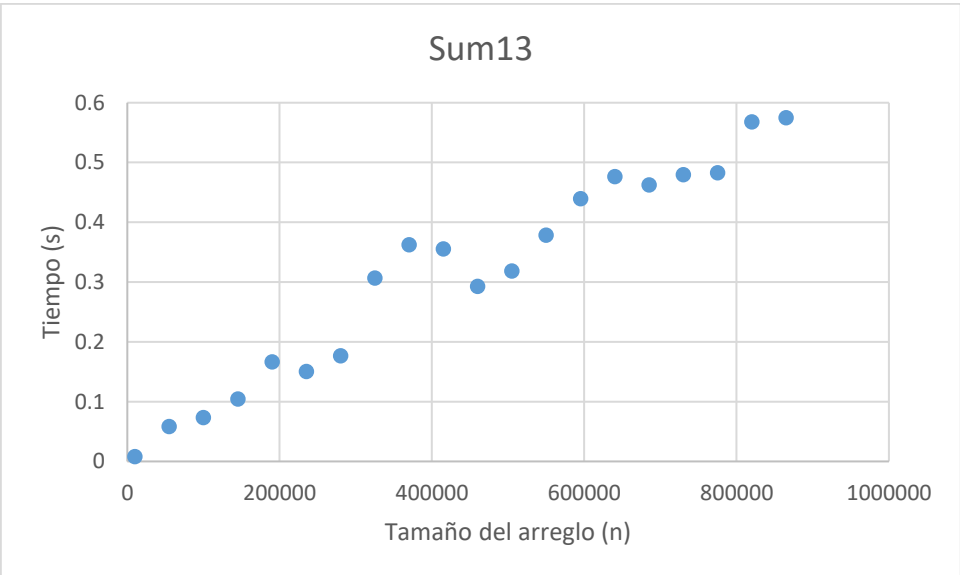


Figura 4. Comportamiento algoritmo CountEvens

Ejercicio 3: Sum13

```
def sum13(n):
    suma=0 #Tn=C1=1
    for i in range(0,len(n)):
        if n[i]!=13:
            suma=suma+n[i] #Tn=C2*n, C2=6
    return suma #Tn=C3=1
```

$$T(n) = n * C_2 + C_3 + C_1$$



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

Figura 5. Comportamiento algoritmo Sum13

Ejercicio 4: Sum28

```
def sum28(n):
    boolean=False
    suma=0 #Tn=C1=2
    for i in range(0,len(n)):
        if n[i]==2:
            suma=suma+n[i] #Tn=C2*n, C2=6
    if suma==8:
        boolean=True #Tn=C3=3
    return boolean #Tn=C4=1
```

$$T(n) = n * C_2 + C_4 + C_1 + C_3$$

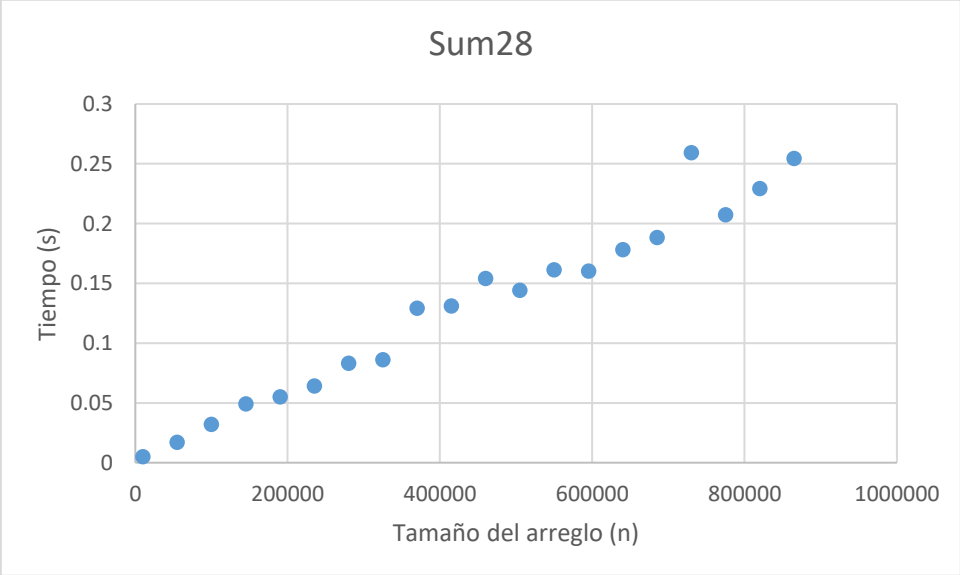


Figura 6. Comportamiento algoritmo Sum28

Ejercicio 5: Lucky13

```
def lucky13(n):
    comp=True #Tn=C1=1
    for i in range(0,len(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

```
    comp=comp and n[i]!=13 and n[i]!=1 #Tn=C2*n, C2=7
return comp #Tn=C3=1
```

$$T(n) = n * C_2 + C_1 + C_3$$

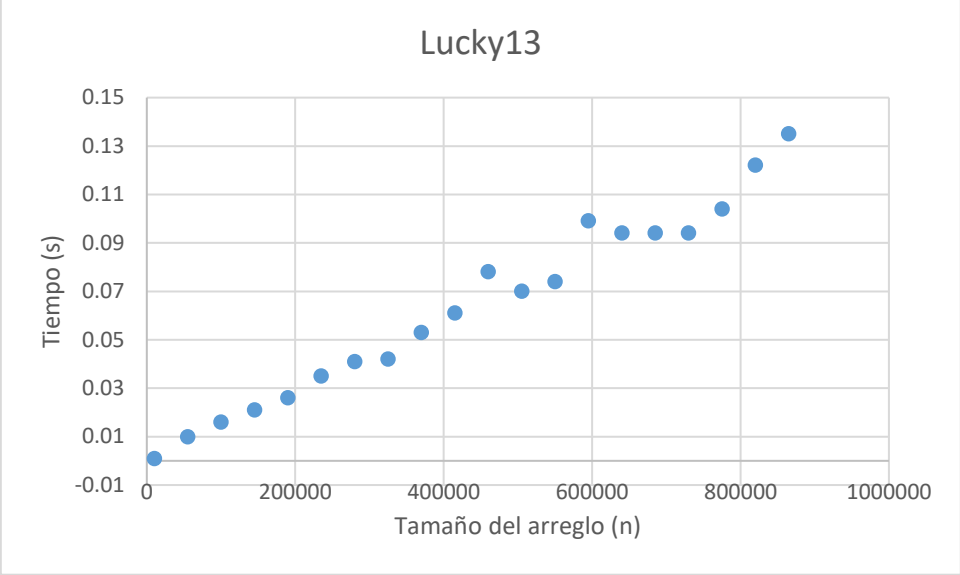


Figura 7. Comportamiento algoritmo Lucky13

3.7.2 Ejercicios Array-3

Ejercicio 1: canBalance

```
def canBalance(array):
    suma1=0
    suma2=0
    array2=array[::-1] # C_4 = 3
    for i in range (0,len(array)):
        suma1=suma1+array[i] # T(n) = n * n * C_1 * C_2
        for j in range(0,len(array2)):
            suma2=suma2+array2[j]
            if suma1==suma2:
                return True # C_1 = 7 T(n)= n * C_1
        j=j+1 # C_2 = 2
    i=i-1 # C_3 = 2
return False # C_5 = 1
```

$$T(n) = (n^2 * C_1 * C_2) + C_3 + C_4 + C_5$$

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

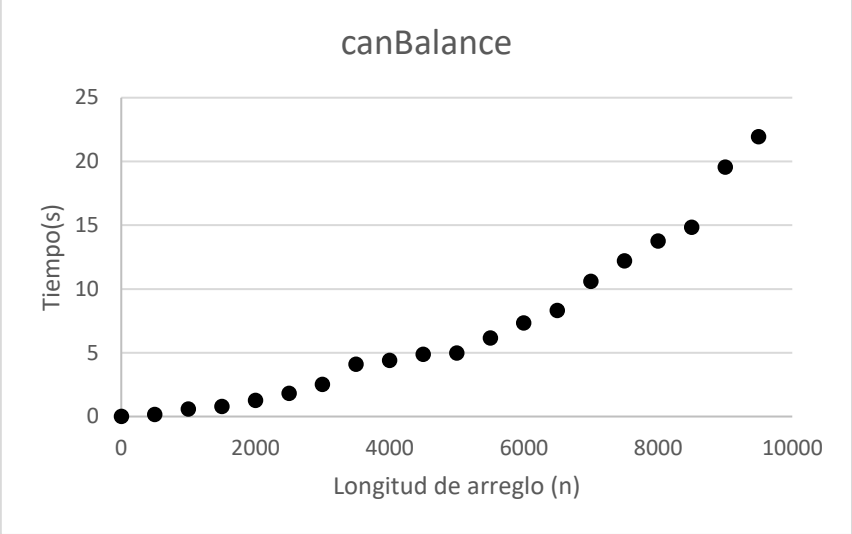


Figura 8. Comportamiento algoritmo canBalance

Ejercicio 2: countClumps

```
def countClumps(array):
    cont=0
    aux=0
    for i in range (0,len(array)-1): # T(n) = n-1 * C_2
        if array[i]==array[i+1] and aux==0: # C_2 = 8
            aux=1
            cont=cont+1
        if array[i]!=array[i+1]: # C_1 = 4
            aux=0
    print(cont) # C_3 = 3
```

$$T(n) = ((n-1) * C_2) + C_3$$

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

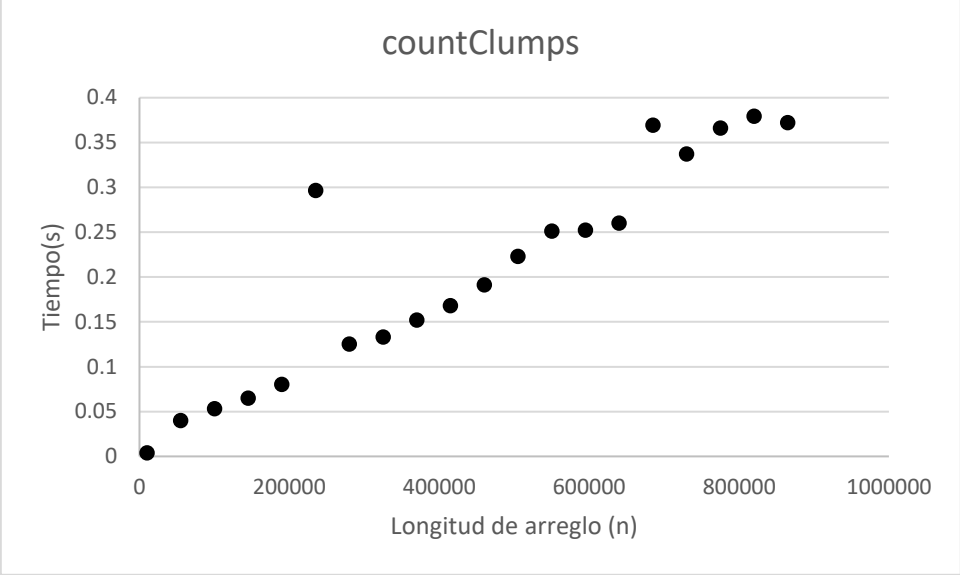


Figura 9. Comportamiento algoritmo countClumps.

Ejercicio 3: Fix34

```
def fix34(array):
    i=0
    j=0
    for i in range (0,len(array)): # T(n) = n * n * C_1
        if array[i]==3:
            if i==0:
                j=i
            else:
                j=i-1
            for j in range (j,len(array)): # T(n) = n * C_1
                if array[j]==4: # C_1 = 7
                    value=array[i+1]
                    array[i+1]=array[j]
                    array[j]=value
            j=j+1
    i=i+1
    print(array) # C_3 = 7
```

$$T(n) = (n^2 * C_1) + C_2$$

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

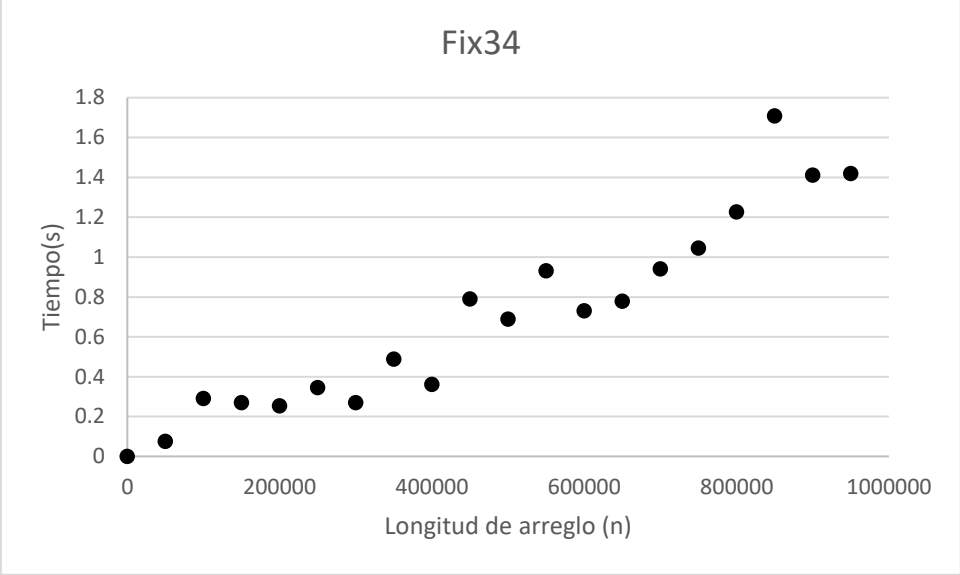


Figura 10. Comportamiento algoritmo Fix34

Ejercicio 4: Fix45

```
def fix45(array):
    i=0
    j=0
    for i in range (0,len(array)): # T(n) = n * n * C_1
        if array[i]==4:
            if i==3:
                j=i
            else:
                j=i-1
            for j in range (j,len(array)): # T(n) = n
                if array[j]==5: # C_1 = 7
                    value=array[i+1]
                    array[i+1]=array[j]
                    array[j]=value
            j=j+1
    i=i+1
    print(array) # C_2 = 7
```

$$T(n) = (n^2 * C_1) + C_2$$

Ejercicio 5: linearIn

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

```
def BorrarRepetidos(array):
    new_array=[]
    for i in range (0,len(array)): # T(n) = n * C_2
        if array[i] not in new_array: # C_2 = 3
            new_array.append(array[i])
    return (new_array)

def linearIn(array1,array2):
    array1=BorrarRepetidos(array1) # T(n) = n * C_2
    array2=BorrarRepetidos(array2) # T(m) = m * C_2
    cont=0
    for i in range (0,len(array2)): # T(n,m) = m * n * C_1
        for j in range (0,len(array1)): # T(n) = n * C_1
            if (array2[i]==array1[j]): # C_1 = 4
                cont=cont+1
    i=i+1
    if cont==len(array2): # C_3 = 6
        return True
    else:
        return False
```

$$T(n) = \big((n * C_2) + (m * C_2) + (m * n * C_1)\big) + C_3$$

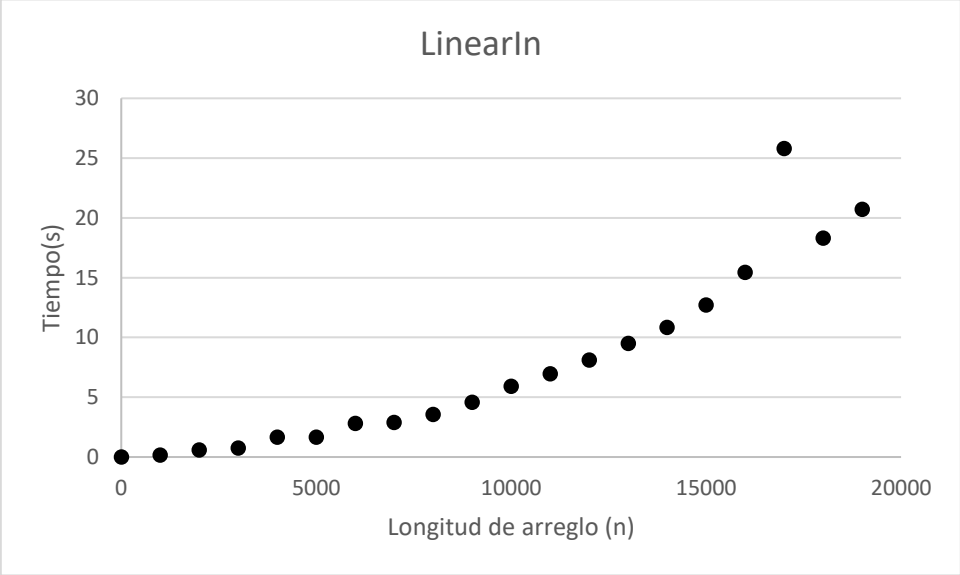


Figura 11. Comportamiento algoritmo LinearIn

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

3.8 Interpretación variables m y n del cálculo de complejidad.

Para todos los ejercicios en línea, tanto del nivel Array-2 como Array-3 la variable n representa la cantidad de elementos del arreglo que entra como parámetro a la función respectiva. En estos casos, solo se calcula la complejidad para una sola variable (n) y está complejidad en el tiempo depende directamente de que tan grande sea el arreglo.

Por otra parte, para el ejercicio LinearIn del nivel Array-3, las variables m y n representan la cantidad de elementos de los arreglos 1 y 2 respectivamente, que entran como parámetros a la función, para este caso, la complejidad en el tiempo depende también de la longitud de estos arreglos y por ende depende de m y n.

4) Simulacro de Parcial

4.2 b

4.5

4.5.1 d

4.5.2 a

4.6 El algoritmo tardará 10000 segundos en procesar 10000 datos.

$$T(n) = c * n^2$$

$$T(100) = c * n^2 = 1$$

$$c = \frac{T(n)}{n^2} = \frac{T(100)}{100^2} = 1x10^{-4}$$

$$T(10000) = c * 10000^2 = 1x10^{-4} * 10000^2 = 10000\ (segundos)$$

4.7 Todas las preposiciones son verdaderas.

4.9 a

4.14 c

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