ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 2
Complejidad

Luis Fernando Vargas Agudelo
Universidad Eafit
Medellín, Colombia
lvarga12@eafit.edu.co

Tomás Bedoya Henao
Universidad Eafit
Medellín, Colombia
tbedoyah@eafit.edu.co

1) Tiempos de ejecución de los algoritmos Insertion Sort y Merge Sort

Para medir los tiempos de ejecución de los diferentes algoritmos se utilizaron tres librerías las cuales ajustando los parámetros de máxima capacidad de recursión que el intérprete en este caso Python puede tener, la longitud del arreglo con el cual se ejecuta el algoritmo, creación de variables aleatorias dentro del arreglo, esto con el fin de que cada arreglo fuese totalmente diferente al anterior y así obtener datos más precisos. Adicional a esto se utilizó la librería time con el fin de conocer los milisegundos que tomaba cada algoritmo en realizar su ejecución.

A continuación, se encuentra la complejidad asintótica, notación O, para cada uno de los algoritmos y los tiempos de ejecución en milisegundos para 20 casos diferentes.

Algoritmo	**Insertion Sort**	**Merge Sort**
Complejidad	$O(n^2)$	$O(\mathrm{n} * \log n)$

Tamaño del Problema	*Tiempo (milisegundos)*	*Tiempo (milisegundos)*
500	25.92	3.97
1000	109.94	7.97
1500	229.87	11.96
2000	544.83	15.95
2500	686.93	21.94
3000	984.36	26.92
3500	1351.42	30.91
4000	1862.02	37.93
4500	2190.79	41.84
5000	3328.13	45.88
5500	3336.06	50.86
6000	3930.10	56.84
6500	4466.94	61.87
7000	5251.48	68.82
7500	6112.83	70.80
8000	7080.69	76.79

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

8500	7831.30	85.73
9000	8945.41	88.80
10000	10629.02	94.74
10500	11888.86	100.69

3) Simulacro de preguntas de sustentación de Proyectos

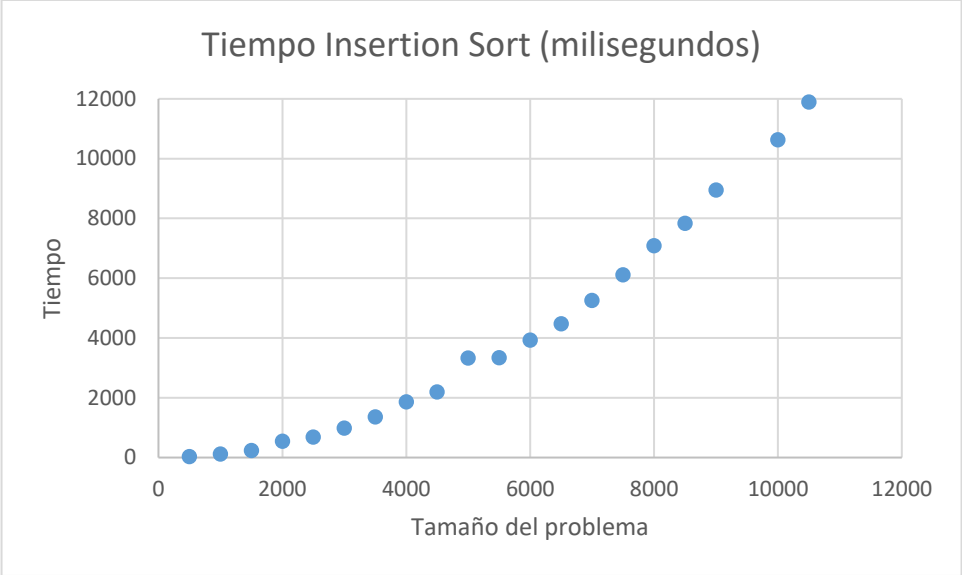
3.1

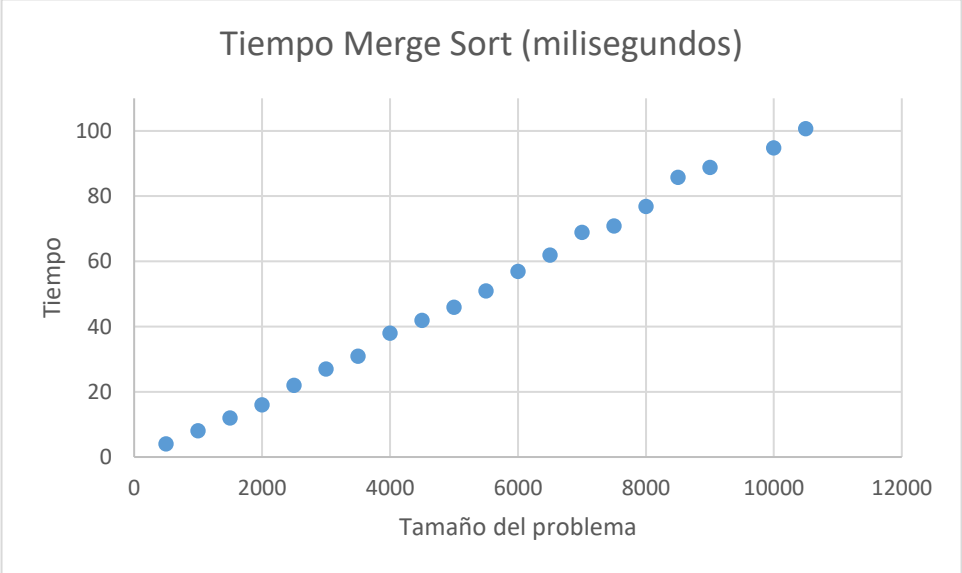
Algoritmo	**Insertion Sort**	**Merge Sort**
Complejidad	$O(n^2)$	$O(\mathrm{n} * \log n)$

Tamaño del Problema	*Tiempo (milisegundos)*	*Tiempo (milisegundos)*
500	25.92	3.97
1000	109.94	7.97
1500	229.87	11.96
2000	544.83	15.95
2500	686.93	21.94
3000	984.36	26.92
3500	1351.42	30.91
4000	1862.02	37.93
4500	2190.79	41.84
5000	3328.13	45.88
5500	3336.06	50.86
6000	3930.10	56.84
6500	4466.94	61.87
7000	5251.48	68.82
7500	6112.83	70.80
8000	7080.69	76.79
8500	7831.30	85.73
9000	8945.41	88.80
10000	10629.02	94.74
10500	11888.86	100.69

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





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

Luego de analizar ambos algoritmos con diferentes tamaños de problema es posible afirmar que la diferencia en la complejidad de los mismo es notoria, la eficiencia que presenta Merge Sort para arreglos grandes es mucho más alta que la brindada por Insertion Sort, para entender un poco más cómo se comportan con tamaños de problemas grandes se realizará una relación de tiempo entre ellos, para una solución con un arreglo de 10000 números Insertion Sort se tomó un tiempo de 10,62 segundos mientras que Merge Sort se demoró para el mismo tamaño 94,74 milisegundos, es decir que Merge Sort soluciona el mismo problema 112,19 veces más rápido que Insertion Sort, esto se ve aún más marcado en problemas con un tamaño más grande.

Debido a esto, utilizar el algoritmo Insertion Sort para problemas con un tamaño de millones de números no es viable y tardaría muchísimo en comparación con Merge Sort. Precisamente por esto Merge Sort es el algoritmo indicado para obtener resultados de arreglos grandes en tiempo real, como pueden ser los renderizados necesarios en juegos como Silent Hill, debido a que es indispensable brindarle la fluidez de juego pertinente a los usuarios y evitar grandes tiempos de carga.

3.5

3.5 (2.1.1)

```
#countEvens
def count(array, cont=0):
for i in range(len(array)):        # C1 + C2n
        if array[i]%2 == 0:        # C3 + C4n
        cont = cont + 1            # C5
return cont                        # C6
                                   # T(n) = C1+C2n+C3+C4n+C5+C6
                                   # T(n) = C4n
```

T(n) = O(n)

3.5 (2.1.2)

```
#Sum13
def Sum13(array):
sum=0                                              # C1
for i in range(len(array)):                        # C2 + C3n
        if i ==0:                                  # (C4n)n
                if array[i] != 13:                 # (C5n)n
                sum = sum + array[i]               # C6n
        else
        if array[i] != 13 and array[i-1]!=13:      # (C7n)n
                sum = sum + array[i]               # C8n
return sum                                         # C9

# T(n) = C1+C2+C3n+C4n^2+C5n^2+C6n+C7n^2+C8n+C9
# T(n) = C7n^2
T(n) = O(N^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

3.5 (2.1.3)

```
#lucky13
def lucky13(array):
        var = True                              # C1
        for i in range(len(array)):             # C2 + C3n
                if array[i]==1 or array[i]==3:  # (C4n)n + C5
                        var = False             # C6
                return var                      # C7

# T(n) = C1+C2+C3n+C4n^2+C5+C6+C7
# T(n) = C4n^2
T(n) = O(N^2)
```

3.5 (2.1.4)

```
#Matchup
def matchup(array1, array2):
        cont = 0                                                                    # C1
        for i in range(len(array1)):                                                # C2 + C3n
            if abs(array1[i] - array2[i])==1 or abs(array1[i] - array2[i])==2:       # (C4n)n + C5m
                        cont = cont + 1                                             # C6
        return cont                                                                 # C7

# T(n) = C1+C2+C3n+C4n^2+C5m+C6+C7
# T(n) = C4n^2
T(n) = O((n^2)+m)
```

3.5 (2.1.5)

```
#Sum28
def sum28(array):
        cont=0                          # C1
        for i in range(len(array)):     # C2 + C3n
                if array[i]==2:         # (C4n)n
                        cont = cont + 1 # C5
                if cont == 4:           # (C6n)n
                        return True     # C7
                else:
                        return False    # C8
# T(n) = C1+C2+C3n+C4n^2+C5+C6N^2+C7
# T(n) = C6n^2
T(n) = O(N^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

3.5 (2.2.1)

```
#Countclumps
def countclumps(array):
        cont = 0                                        # C1
        aux = None                                      # C2
        for i in range(1,len(array)):                   # C3 + C4n
                if array[i]==array[i-1]:                # (C5n)n
                        aux = array[i]                  # (C6n)n
                if i+1 == len(array) and aux!=None:     # C7 + C8 + C9n
                        cont = cont + 1                 # C10
                else:
                if aux != None and array[i]!=array[i-1]:        # C11 + C12 + C13n
                         cont = cont + 1                        # C14
                        aux = None                              # C15
        return cont                                             # C16

# T(n) = C1+C2+C3n+C4n+C5n^2+C6n^2+C7+C8+C9n+C10+C11+C12+C13n+C14+C15
# T(n) = C6n^2
T(n) = O(N^2)
```

3.5 (2.2.2)

```
#Canbalance
def canbalance(array):
        sum1 = 0                                # C1
        sum2 = 0                                # C2
        res = False                             # C3
        for i in range(len(array)):             # C4 + C5n
                sum1 = array[i] + sum1          # (C6n)n
        for j in range(len(array)):             # C7 + C8n
                sum2 = array[j]+sum2            # (C9n)n
                sum1 = sum1 - array[j]          # (C10n)n
                if sum1 == sum2:                # (C11n)n
                        return True             # C12
 return res                                     # C13
# T(n) = C1+C2+C3+C4+C5n^2+C6n^2+C7+C8n+C9n^2+C10 n^2+C11n^2+C12+C13
# T(n) = C11n^2
T(n) = O(N^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

3.5 (2.2.3)

```
#seriesup
def seriesup(n):
        array = []                          # C1
        for i in range(1,n+1):              # C2 + C3n
                for j in range(1,i+1):      # (C4 + C5n)n
                        array.append(j)     # (C6n)n
        return(array)                       # C7

# T(n) = C1+C2+C3n+C4n+C5n^2+C6n^2+C7
# T(n) = C6n^2
T(n) = O(N^2)
```

3.5 (2.2.4)

```
#Linearin
def linearin(array1,array2):
        for i in range(len(array2)):                # C1 + C2m
                key = False                         # C3
                for j in range(len(array1)):        # (C4 + C5n)m
                        if array2[i]==array1[j]:    # ((C6m)n)m
                                key = True          # C7
                if key == False:                    # (C8n)m
                        break                       # C9
return key                                          # C10

# T(n) = C1+C2m+C3+C4m+C5n*m+C6n*m^2+C7+C8n*m+C9+C10
# T(n) = C6n*m^2
T(n) = O(n*m^2)
```

3.6

Las variables n y m son ambas requerimientos de la función que se desea evaluar, esta función como se desarrolla de manera recursiva debe tener un caso base que permita parar y obtener los resultados, en los casos anteriormente analizados, la variable que estaba en el caso base n y no m, aunque cada vez que se llame nuevamente la función se debe tener en cuenta la variable m, ya que con esta se realiza una operación de comparación o asignación generando así una nueva tarea, la cual va a ser tan grande como lo sea m, de la misma forma funciona para n, pero esta variable como es quien se encuentra directamente relacionada con el caso base, debe disminuir su valor cada vez que se llame nuevamente la función, creando así la tarea de complejidad, la cual se compone por como reduce n y la comparación o asignación de m.

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

4) Simulacro de Parcial

4.1 *C*
4.2 *B*
4.3 *D*
4.4 *B*
4.5 *.*
- ***4.5.1*** *D*
- ***4.5.2*** *Sí*

4.6 *T(C * n^2) = O(n^2), para n = 10000 , T(10000) = 100’000.000 milisegundos = 100000 segundos se tardaría.*
4.7 *Opciones 1,3 y 4*
4.8 *A*
4.9 *C*
4.10 *A*
4.11 *C*
4.12 *B*
4.13 *B*
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