Universidad del Valle de Guatemala
Facultad de Ingeniería
Teoría de la Computación
Sección 30

Ana Laura Tschen 221645
22 de Septiembre 2025

Laboratorio 8

Ejercicio 1

```
void function (int n) {
       int i, j, k, counter = 0;
       for (i = n/2; i <= n; i++) {
              for (j = 1; j+n/2 <= n; j++) {
                     for (k = 1; k <= n; k = k*2) {
                            counter++;
                     }
              }
       }
}
```

a) Encuentre la complejidad de tiempo en notación Big-Oh. Deje todo su procedimiento.

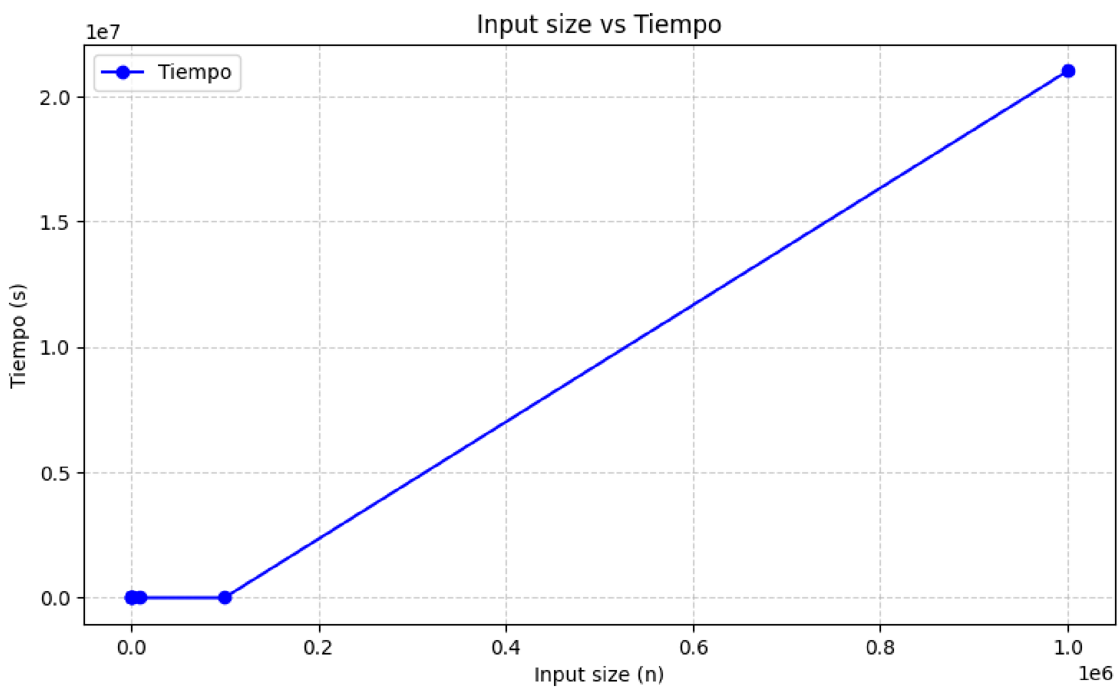
1. Analizar el primer for. El primer for contiene la condición (i = n/2; i<= n; i++) lo que significa que i se ejecuta n/2 veces dando como resulado O(n)
2. Analizar el segundo for. El segundo for posee la condición (j = 1; j + n/2 <= n; j++) que hace que j vaya desde 1 a n/2. Esto resulta en n/2 iteraciones, dando como resultados O(n).
3. Analizar el tercer for. El tercer for tiene la condición (k = 1; k<= n; k = k*2) Lo que resulta ser un logaritmo por la cantidad de veces que se puede multiplicar por 2 antes de llegar a n, dando como resultado O(log n).
4. Procedemos a multiplicar los costos de cada uno de nuestros ciclos for. Lo que sería:
$$O(n) \times O(n) \times O(\log(n)) = O(n^2 \log(n))$$

R: La complejidad sería de $O(n^2 \log(n))$

b) Ahora, escriba un programa en el lenguaje de programación de su elección para implementar el programa anterior y utilice un método de profiling para medir el tiempo de ejecución de su programa con distintos tamaños de input n: 1, 10, 100, 1000, 10000, 100000, 1000000. Coloque los resultados en una tabla y grafique los resultados obtenidos en una gráfica de tamaño de input vs. Tiempo.

Input size (n)	Time (s)
1	0.000002
10	0.000003
100	0.000030
1000	0.003188

10000	0.409877
100000	49.676073
1000000	N/A



Ejercicio 2

```
void function (int n) {
        if (n <= 1) return;
        int i, j;
        for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                        printf ("Sequence\n");
                        break;
                }
        }
}
```

a) Encuentre la complejidad de tiempo en notación Big-Oh. Deje todo su procedimiento.

1. Analizar el primer for. El primer for tiene la condición (n <= 1). Lo que nos da como resultado O(n).
2. Analizar el segundo for. El segundo for tiene la condición de (j=1; j <= n; j++) . Pero es importante mencionar que cuando n = 1 (ósea, en la primera iteración) hay un break. Que es el caso para todos nuestros inputs, nos da como resultado O(1).

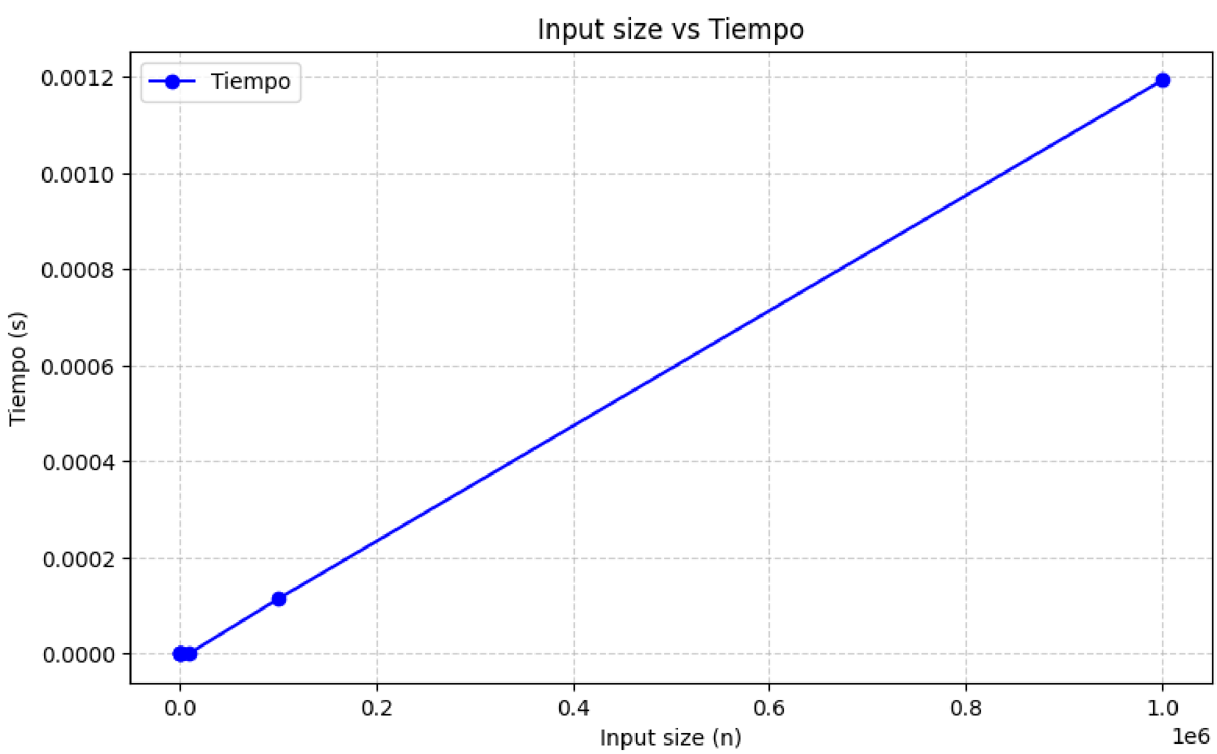
3. Procedemos a multiplicar los costos de todos nuestros ciclos de la siguiente manera:

$$O(n) \times O(1) = O(n)$$

R: Por lo que la complejidad para este ejercicio es de $O(n)$.

b) Ahora, escriba un programa en el lenguaje de programación de su elección para implementar el programa anterior y utilice un método de profiling para medir el tiempo de ejecución de su programa con distintos tamaños de input n: 1, 10, 100, 1000, 10000, 100000, 1000000. Coloque los resultados en una tabla y grafique los resultados obtenidos en una gráfica de tamaño de input vs. Tiempo.

Input size (n)	**Time (s)**
1	0.0000001
10	0.0000001
100	0.0000001
1000	0.0000002
10000	0.0000012
100000	0.000114
1000000	0.001193



Ejercicio 3

```
void function (int n) {
      int i, j;
      for (i=1; i<=n/3; i++) {
            for (j=1; j<=n; j+=4) {
                  printf("Sequence\n");
            }
      }
}
```

a) Encuentre la complejidad de tiempo en notación Big-Oh. Deje todo su procedimiento.

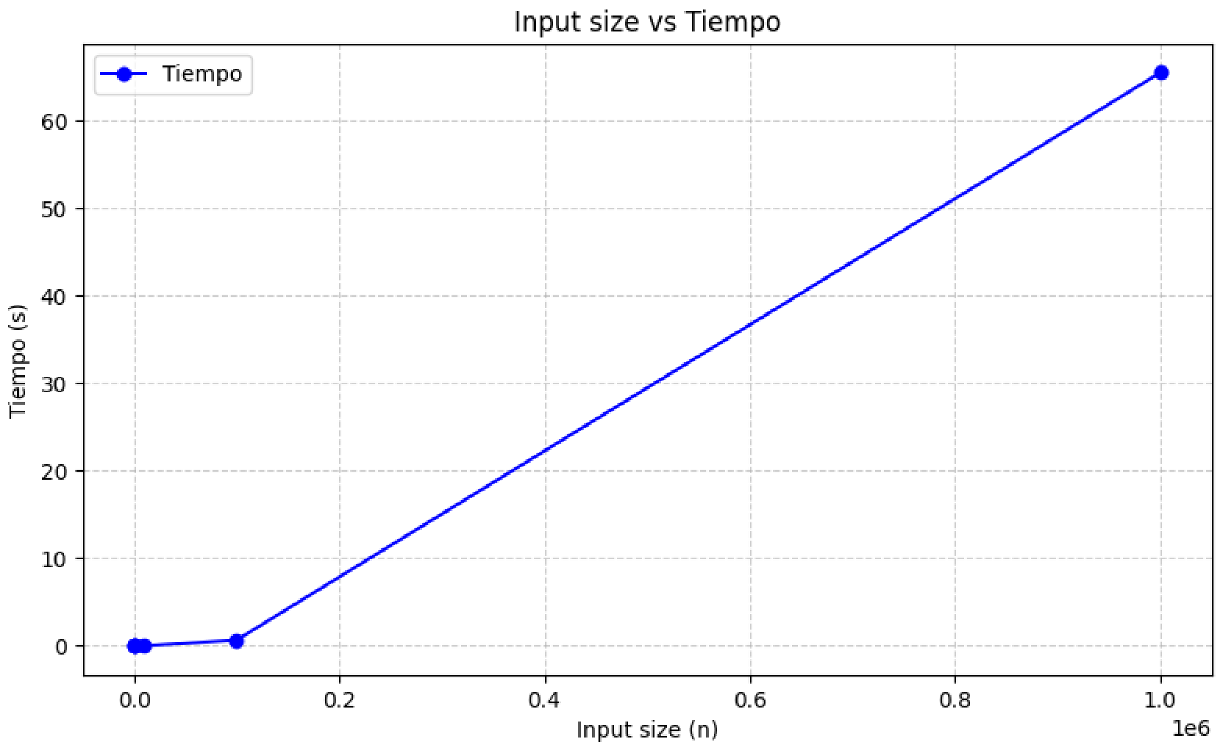
1. Analizar el primer for. El primer for tiene la condici[on de que i debe ser menor o igual a n/3. Lo que nos da como resultado un O(n).
2. Analizar el segundo for. El segundo for tiene la condición de j debe ser menor o igual a n pero se agrega 4 más luego de cada iteración, eso significa n/4 iteraciós resultando en un O(n).
3. Multiplicar los costos de los ciclos:

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

R: La complejidad de este ejercicio es de $O(n^2)$

b) Ahora, escriba un programa en el lenguaje de programación de su elección para implementar el programa anterior y utilice un método de profiling para medir el tiempo de ejecución de su programa con distintos tamaños de input n: 1, 10, 100, 1000, 10000, 100000, 1000000. Coloque los resultados en una tabla y grafique los resultados obtenidos en una gráfica de tamaño de input vs. Tiempo.

Input size (n)	Time (s)
1	0.000001
10	0.000001
100	0.000001
1000	0.000086
10000	0.006636
100000	0.664201
1000000	65.469084



Ejercicio 4. Encuentre el mejor caso, caso promedio y peor caso del algoritmo de Búsqueda Lineal (Linear Search). Deje todo su procedimiento.

MEJOR CASO

El mejor caso es donde el elemento ya se encuentra en la primera posición. Esto hace que no se deban realizar operaciones adicionales. Complejidad $O(1)$

CASO PROMEDIO

se da cuando las comparaciones de éxito son el promedio de $1 \dots n$, igual a $(n+1)/2$. y las comparaciones en fracaso $= n$. Complejidad $O(n)$

PEOR CASO

se da cuando x no se encuentra en el arreglo o está de último. Resulta en comparaciones $= n$, con una complejidad de $O(n)$

Ejercicio 5. Decida si los siguientes enunciados son verdaderos o falsos. Debe justificar sus respuestas para recibir los créditos completos.

a) **Si** $f(n) = \Theta(g(n))$ **y** $g(n) = \Theta(h(n))$, **entonces** $h(n) = \Theta(f(n))$.

b) **Si** $f(n) = O(g(n))$ **y** $g(n) = O(h(n))$, **entonces** $h(n) = \Omega(f(n))$.

c) $f(n) = \Theta(n^2)$, **donde** $f(n)$ **está definido por ser el tiempo de ejecución del programa de Python** `A(n)`:

```
def A(n):
    atupla = tuple(range(0, n))  # una tupla es una versión immutable de una
                                 # lista, que puede ser hasheada
S = set()
for i in range(0, n):
    for j in range (i + 1, n):
        S.add(atupla[i:j])       # añada la tupla (i,...,j-1) al set S
```

a) si $f(n) = \Theta(g(n))$ y $g(n) = \Theta(h(n))$, entonces $h(n) = \Theta(f(n))$

Reemplazamos $g(n)$ por $\Theta(h(n))$ (transitividad)

$\hookrightarrow$ $f(n) = \Theta(h(n))$

Por simetría

$h(n) = \Theta(f(n))$

R: verdadero //

b) Si $f(n) = O(g(n))$ y $g(n) = O(h(n))$, entonces $h(n) = \Omega(f(n))$

Reemplazamos $g(n)$ por $O(h(n))$ (transitividad)

$\hookrightarrow$ $f(n) = O(h(n))$

por definición dual $f(n) = O(h(n)) \longleftrightarrow h(n) = \Omega(f(n))$

R: verdadero //

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

- Atupla $\rightarrow O(n)$, ya que depende de n.
- for i in range $(0,n) \rightarrow O(n)$, ya que depende de n.
- for j in range $(i+1, n) \rightarrow$ subtupla de i que depende de j.

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \Theta(j-i)$$

$K = j-i$

$$= \sum_{i=0}^{n-1} \sum_{K=1}^{n-i} \Theta(k) = \sum_{i=0}^{n-1} \Theta((n-i)^2) = \Theta(n^3)$$

R: falso //

Link al repo de github: https://github.com/tsc221645/tdlc_lab8