Esta es la Tarea 2 del curso *Estructuras de Datos*, 2023-2. La actividad se debe realizar en **parejas** o de forma **individual**. Sus soluciones deben ser entregadas a través de Discord a más tardar el día **18 de Agosto a las 23:59**. Todos los puntos de la tarea deben ser realizadosen un único archivo llamado `tarea2.pdf`. En caso de dudas y aclaraciones puede escribir por el canal `#tareas` en el servidor de *Discord* del curso o comunicarse directamente con el profesor y/o el monitor.

Estudiantes: Daniel Felipe Moncada Tello y Juan Camilo Vasquez Jaramillo
id: 8976528 y 8976396

Complejidad Teórico-Práctica

1. Escriba en Python una función que permita calcular el valor de la función de Fibonacci para un número *n* de acuerdo a su definición recursiva. Tenga en cuenta que la función de Fibonacci se define recursivamente como sigue:

$$Fibo(0) = 0$$
$$Fibo(1) = 1$$
$$Fibo(n) = Fibo(n-1) + Fibo(n-2)$$

Obtenga el valor del tiempo de ejecución para los siguientes valores (en caso de ser posible):

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.126s	35	4.482s
10	0.131s	40	37.623s
15	0.136s	45	7m05.234s
20	0.145s	50	No lo ejecuta
25	0.153s	60	No lo ejecuta
30	0.530s	100	No lo ejecuta

¿Cuál es el valor más alto para el cuál pudo obtener su tiempo de ejecución? ¿Qué puede decir de los tiempos obtenidos? ¿Cuál cree que es la complejidad del algoritmo?

```python
def fiboRecursiva(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fiboRecursiva(n-1)+fiboRecursiva(n-2)

fiboRecursiva(30)
```

Respuesta: el valor más alto que se logró alcanzar fue el 45 ya después con las demás entradas al pasar

del tiempo tampoco terminaba de ejecutarse, la función Fibonacci recursiva no es recomendable para ejecutar entradas con tamaños grandes, creo que la complejidad del algoritmo seria algo mayor a lo que hayamos visto así que con certeza no sabría decir cual me parece que es.

2. Escriba en Python una función que permita calcular el valor de la función de Fibonacci utilizando ciclos y sin utilizar recursión. Halle su complejidad y obtenga el valor del tiempo de ejecución para los siguientes valores:

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1

    numero_anterior = 0
    numero_actual = 1

    for i in range(2, n + 1):
        numero_futuro = numero_actual + numero_anterior
        numero_anterior = numero_actual
        numero_actual = numero_futuro

    return numero_actual

fibonacci(10)
```

Complejidad del algoritmo: O(n)

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.142s	45	0.145s
10	0.145s	50	0.141s
15	0.145s	100	0.127s
20	0.157s	200	0.126s
25	0.149s	500	0.132s
30	0.153s	1000	0.125s
35	0.126s	5000	0.122s
40	0.142s	10000	0.126s

3. Considere la siguiente definición para la operación `mostrarDivisores` solicitada en el ejercicio 5 de la Tarea 1:

```
import math

def mostrarDivisores(N):
  divs1, divs2, ac = [], [], 0
  i = 1
  while i * i  <= N:
    if N % i == 0:
      divs1.append(i)
      divs2.append(N // i)
      ac += divs1[-1] + divs2[-1]
    i += 1
  print("Divisores número %d:" % N)
  for i in range(len(divs1)):
    print(" --> %d," % divs1[i])
  ini = len(divs2)  - 1 if divs2[-1] != divs1[-1] else len(divs2) - 2

  for i in range(ini, -1, -1):
    if i > 0:  print(" --> %d," % divs2[i])
    else: print(" --> %d" % divs2[i])
  print()
  print("Suma de los divisores del número %d: %d" % (N, ac))

n = 10000
mostrarDivisores(n)
```

Mejor caso:	Peor caso:
3	3
1	1
$\sqrt{n}+1$	$\sqrt{n}+1$
$\sqrt{n}$	$\sqrt{n}$
0	$\sqrt{n}$
0	$\sqrt{n}$
0	$\sqrt{n}$
$\sqrt{n}$	$\sqrt{n}$

Ejecute el código anterior con los siguientes valores y mida el tiempo de ejecución. Además, reemplace en el código anterior la definición de la operación `mostrarDivisores` por la definición que usted presentó en la Tarea 1 y mida también el tiempo de ejecución con los valores en la tabla.

Tamaño Entrada	Tiempo Solución Propia	Tiempo Solución Profesor
10000	0.132s	0.142s
50000	0.127s	0.126s
100000	0.141s	0.126s
1000000	0.164s	0.127s
10000000	0.547s	0.144s
100000000	4.195s	0.143s
1000000000	40.354s	0.159s

Para la medición del tiempo de ejecución debe seguir las instrucciones que se darán en clase. Tenga en cuenta que en cada ejecución debe cambiar el valor de la variable `n` por cada valor en la tabla.

Responda las siguientes preguntas:

(a) ¿qué tan diferentes son los tiempos de ejecución y a qué cree que se deba dicha diferencia?
R/ se puede ver que la diferencia se hace notable en la antepenúltima entrada, esta diferencia debe ser porque el código de la solución del profesor aunque sea mas extensa, esta mucho mas optimizado y requiere de menos procesos para dar solución al tamaño de entrada.

(b) ¿cuál es la complejidad de la operación o bloque de código en el que se determinan los

Solución Profesor: $T(n) = 3 + 1 + \sqrt{n} + 1 + \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} = 5 + 6\sqrt{n} \rightarrow O(n)$
Solución propia: $T(n) = O(n)$

Ejercicios de Complejidad Teórica

4. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
void algoritmo1(int n){
   int i, j = 1;                          → 2
   for(i = 1; i < n * n; i = i * 3){      → [ 2log3 n + 1 ]
       int suma = i + j;                  → [ 2log3 n ]
       printf("Suma %d\n", suma);         → [ 2log3 n ]
       ++j;                               → [ 2log3 n ]
   }
}
```

Qué se obtiene al ejecutar `algoritmo1(10)`? Explique.

R/

Observaciones del algoritmo:

- Al ejecutar el algoritmo1 (10) se imprime:

  Suma 2

  Suma 5

  Suma 12

  Suma 31

  Suma 86

$T(n) = 2 + 2\log_3 n + 1 + 2\log_3 n + 2\log_3 n + 2\log_3 n = 3 + 8\log_3 n \longrightarrow O(\log n)$

5. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
int algoritmo2(int n){
   int res = 1, i, j;                     → 3

   for(i = 1; i <= 3 * n; i += 5)         → (3n/5)+1
       for(j = 0; j <= sqrt(n); j++)      → ⌊ (3n/5)(√n+2) ⌋
           res += n;                      → ⌈ (3n/5)(√n+1) ⌉

   return res;                            → 1
}
```

Qué se obtiene al ejecutar `algoritmo2(8)`? Explique.

R/

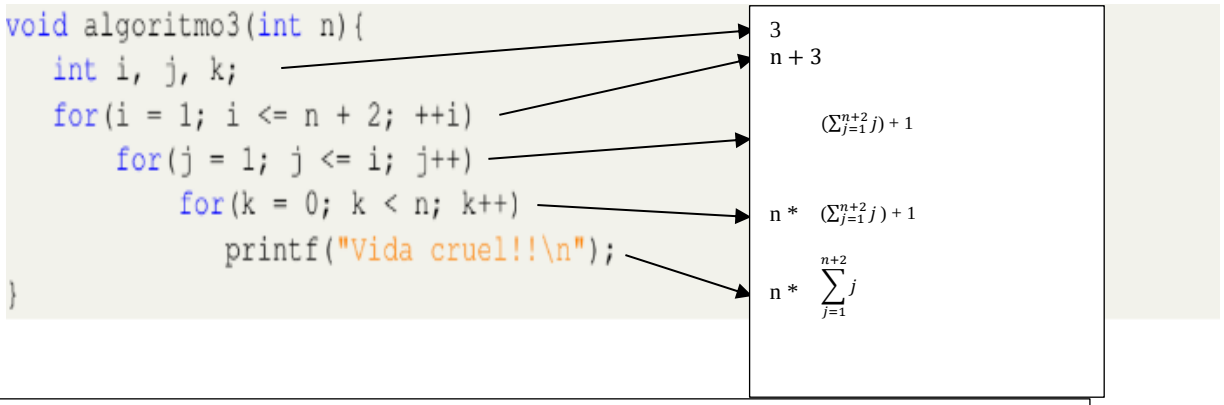
Observaciones del algoritmo:

- Al ejecutar el algoritmo2 (8) no imprime nada, pero si vemos el valor que guarda res al retornarlo, seria

  Res: 121

$T(n) = 3 + (3n/5)+1 + (3n/5)(\sqrt{n}+2) + (3n/5)(\sqrt{n}+1) + 1 = 5 + (12n+6n\sqrt{n})/5 \longrightarrow O(n)$

6. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

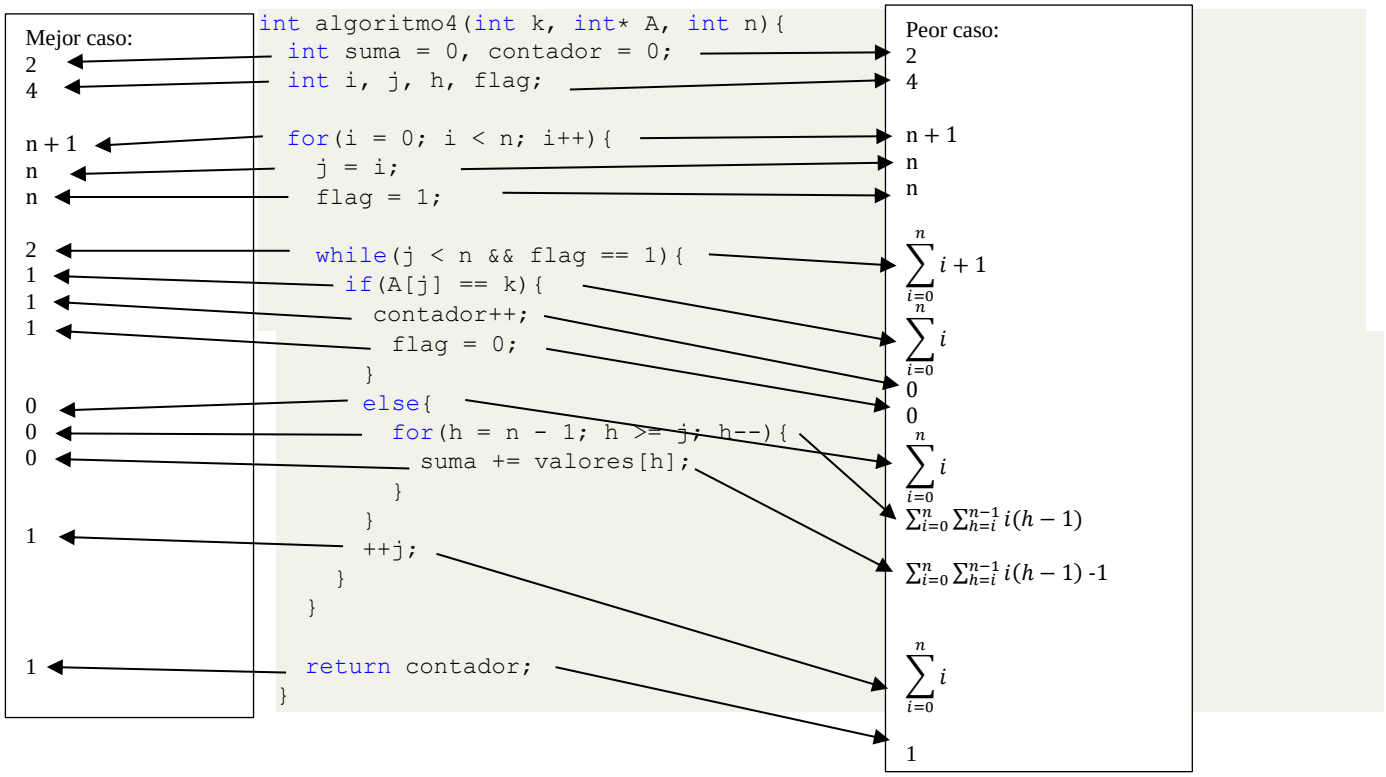


R/

$T(n) = 3 + n + 3 + (\sum_{j=1}^{n+2} j) + 1 + n\,(\sum_{j=1}^{n+2} j) + 1 + n\,(\sum_{j=1}^{n+2} j) = 11 + 19n/2 + 11n^2/2 + n^3$

$O(n^3)$

7. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:



Observaciones del algoritmo:

-

Mejor caso: T(n) = 2 + 4 + n + 1 + n + n + 2 + 1 + 1 + 1 + 1 + 1 = 3n + 14 ⟶ O(n)
Peor caso: T(n) = 2 + 4 + n + 1 + n + n + $\sum_{i=0}^{n} i + 1 + \sum_{i=0}^{n} i + \sum_{i=0}^{n} i + \sum_{i=0}^{n} \sum_{h=i}^{n-1} i(h-1) + \sum_{i=0}^{n} \sum_{h=i}^{n-1} i(h-1)$ -1 + $\sum_{i=0}^{n} i$ +1 n ⟶ $O(n^2)$

8. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
void algoritmo5(unsigned int n){
  int i = n;
  while(i > 0){
    printf("%d\n", i);
    i -= n / 4;
  }
}
```

Si n es divisible entre 4:	Si n NO es divisible entre 4:
1	1
5	6
4	5
4	5

R/

Observaciones del algoritmo:

- A partir de lo visto si (n < 4) al entrar al while este se convertirá en un ciclo infinito ya que a nuestra variable i no se va a alterar al restarle 0 que será el resultado de (n < 4)/4 .
- También puedo concluir que el algoritmo tiene dos distintas situaciones en las cuales se ejecutara mas veces una que en la otra a partir de si la entrada es divisible entre 4 o no.

Divisible: T(n) = 1 + 5 + 4 + 4 = 14 ⟶ O(1)
NO divisible: T(n) = 1 + 6 + 5 + 5 = 17 ⟶ O(1)