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 **20 de Agosto a las 23:59**. Todos los puntos de la tarea deben ser realizados en 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.

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		35	
10		40	
15		45	
20		50	
25		60	
30		100	

   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?

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:

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5		45	
10		50	
15		100	
20		200	
25		500	
30		1000	
35		5000	
40		10000	

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

```
def mostrarDivisores1(N):
  divs1, divs2, ac = [], [], 0
  i = 1
  while i * i  <= N:
    if N % i == 0:
      divs1.append(i)
      divs2.append(N // i)
      if divs1[-1] != divs2[-1]:  ac += divs1[-1] + divs2[-1]
      else: ac += divs1[-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)
```

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		
50000		
100000		
1000000		
10000000		
100000000		
1000000000		

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?

(b) ¿cuál es la complejidad de la operación o bloque de código en el que se determinan los divisores en cada una de las soluciones?

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;
   for(i = 1; i < n * n; i = i * 3){
       int suma = i + j;
       printf("Suma %d\n", suma);
       ++j;
   }
}
```

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

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;

   for(i = 1; i <= 3 * n; i += 5)
       for(j = 0; j <= sqrt(n); j++)
           res += n;

   return res;
}
```

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

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

```
void algoritmo3(int n){
   int i, j, k;
   for(i = 1; i <= n + 2; ++i)
       for(j = 1; j <= i; j++)
           for(k = 0; k < n; k++)
              printf("Vida cruel!!\n");
}
```

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

```
int algoritmo4(int k, int* A, int n){
  int suma = 0, contador = 0;
  int i, j, h, flag;

  for(i = 0; i < n; i++){
    j = i;
    flag = 1;

    while(j < n && flag == 1){
      if(A[j] == k){
        contador++;
        flag = 0;
```

```
      }
      else{
        for(h = n - 1; h >= j; h--){
          suma += A[h];
        }
      }
      ++j;
    }
  }

  return contador;
}
```

¿Qué calcula esta operación? Explique.

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;
  }
}
```