Esta es la Tarea 2 del curso *Estructuras de Datos*, 2023-1. La actividad se debe realizar en **parejas** o de forma **individual**. Sus soluciones deben ser entregadas a través de BrightSpace a más tardar el día **15 de Febrero a las 22:00**. En caso de dudas y aclaraciones puede escribir por el canal `#tareas` en el servidor de *Discord* del curso o comunicarse directamente con los profesores y/o el monitor.

Condiciones Generales

- Para la creación de su código y documentación del mismo use nombres en lo posible cortos y con un significado claro. La primera letra debe ser minúscula, si son más de 2 palabras se pone la primera letra de la primera palabra en minúscula y las iniciales de las demás palabras en mayúsculas. Además, para las operaciones, el nombre debe comenzar por un verbo en infinitivo. Esta notación se llama *lowerCamelCase*.

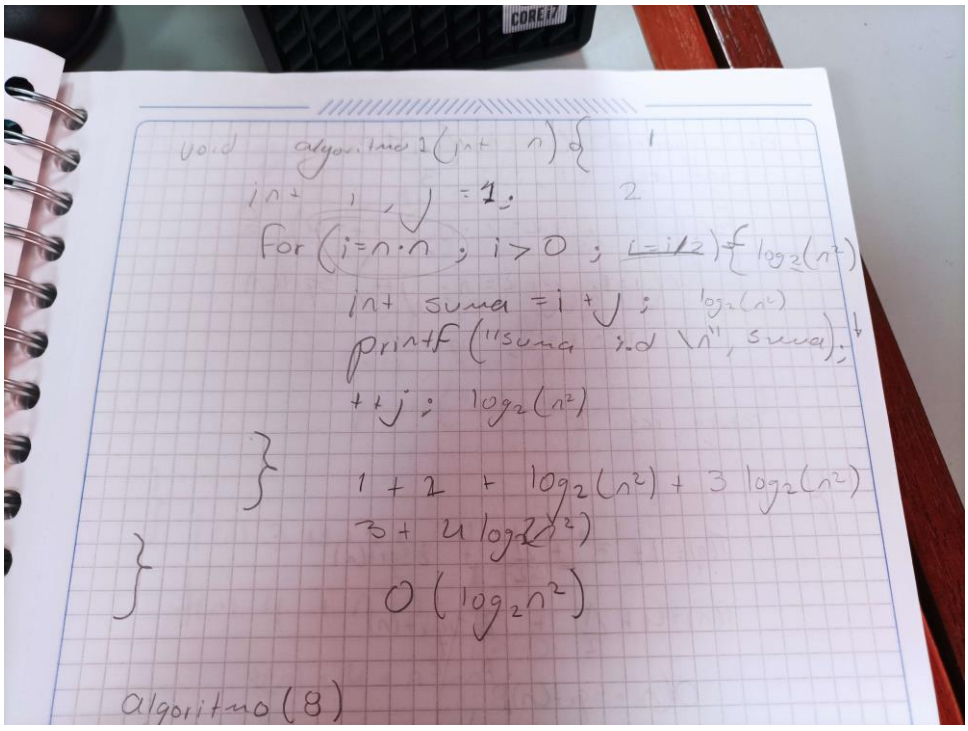
  Ejemplos de funciones: `quitarBoton`, `calcularCredito`, `sumarNumeros`

  Ejemplos de variables: `sumaGeneralSalario`, `promedio`, `nroHabitantes`

- Todos los puntos de la tarea deben ser realizados en un único archivo llamado `tarea2.pdf`.

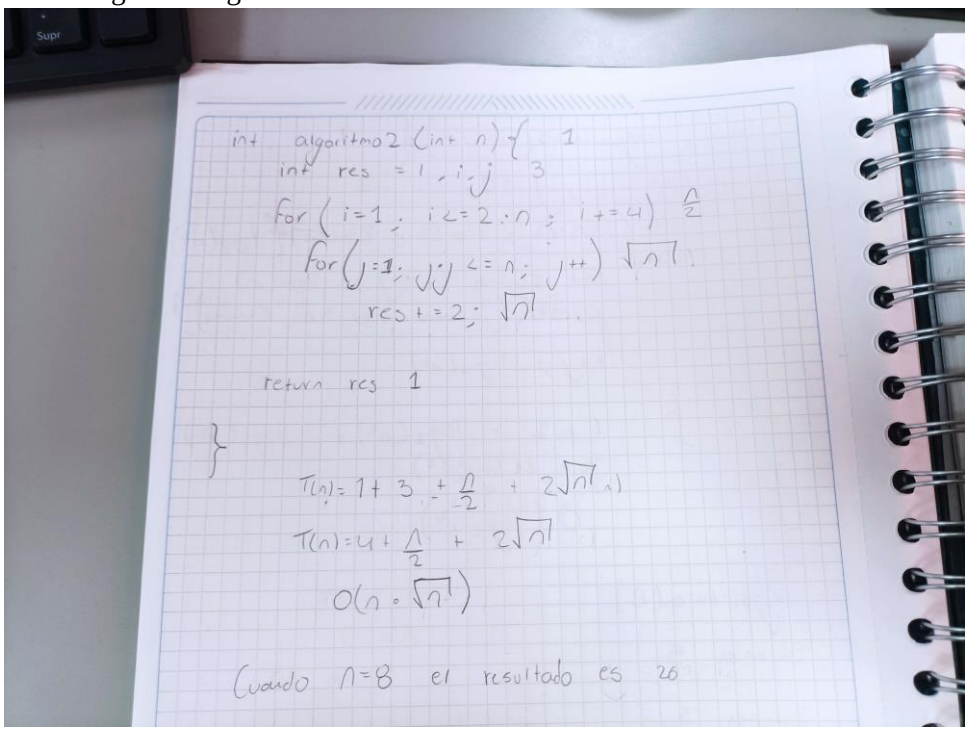
Ejercicios de Complejidad Teórica

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



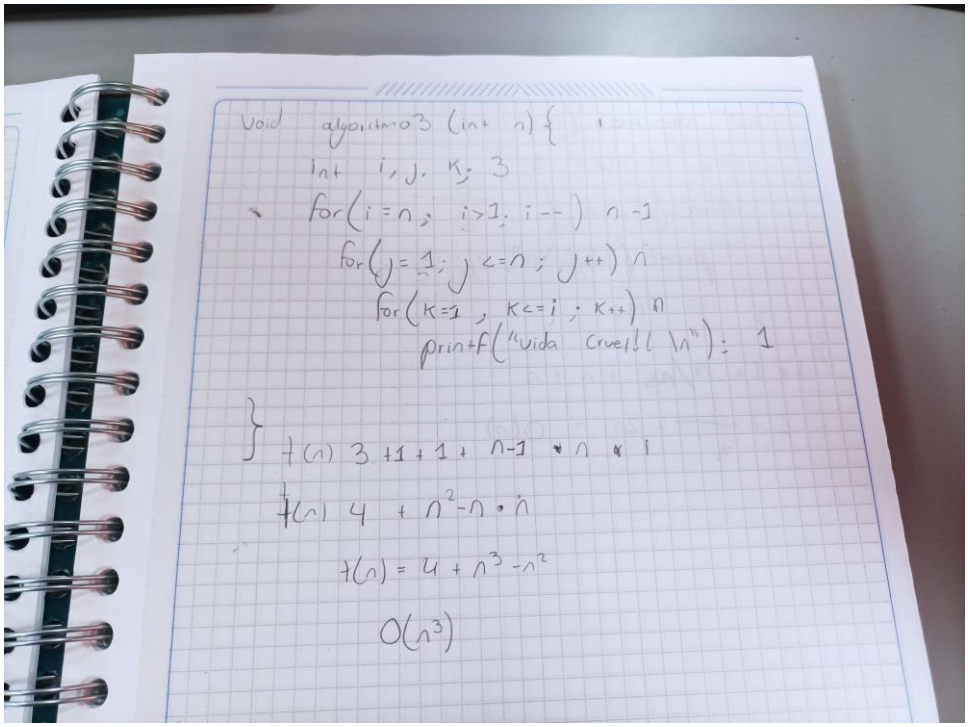
Qué se obtiene al ejecutar `algoritmo1(8)`? Explique.

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



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

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



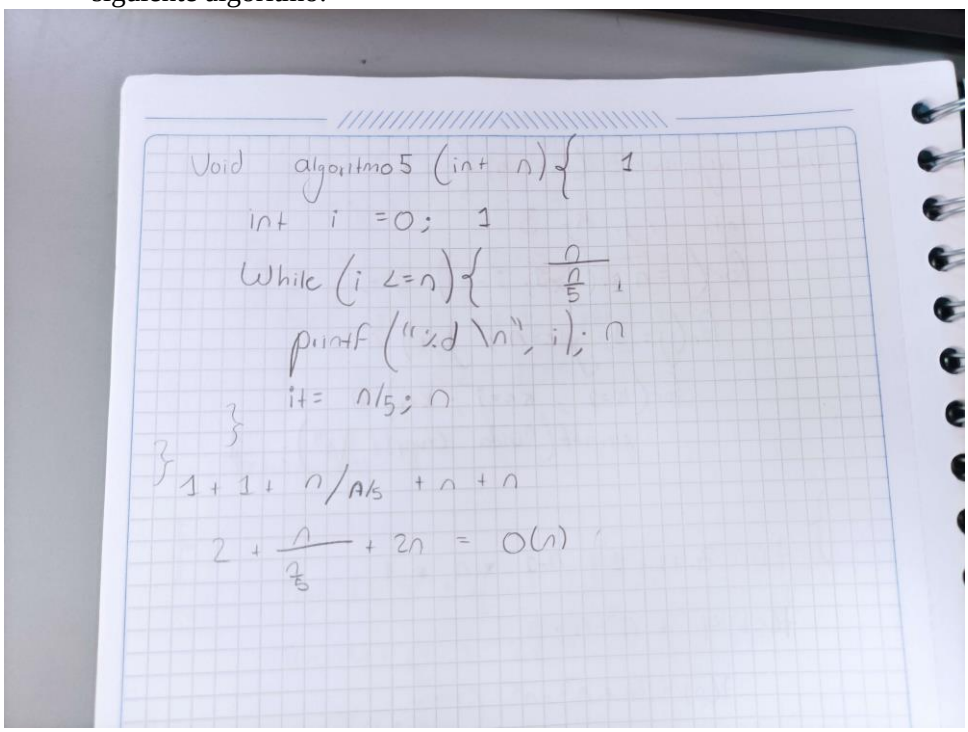
4. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

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

   for(i = 0; i < n; i++){
     j = i + 1;
     flag = 0;
     while(j < n && flag == 0){
        if(valores[i] < valores[j]){
           for(h = j; h < n; h++){
             suma += valores[i];
           }
        }
        else{
           contador++;
           flag = 1;
        }
        ++j;
     }
   }
   return contador;
}
```

¿Qué calcula esta operación? Explique.

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



Complejidad Teórico-Práctica

6. 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	0m0.041s	35	0m15.611s
10	0m0.028s	40	2m52.280s
15	0m0.026s	45	...
20	0m0.035s	50	...
25	0m0.104s	60	...
30	0m1.342s	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?

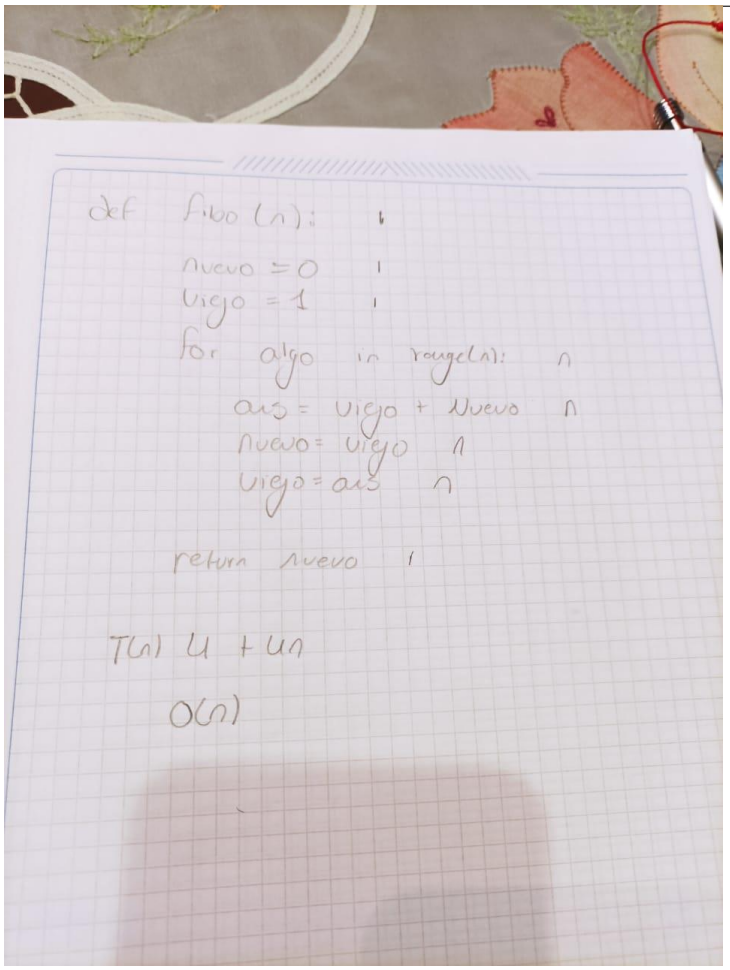
El valor más alto obtenido es de 40 en cuestión de tiempo de ejecución “aceptable”.

Se puede decir que los tiempos obtenidos aumentan a medida que se generan distintas ramificaciones del número de Fibonacci.

Yo creo que la complejidad de este algoritmo equivale a O(2^n) debido a que la respuesta se puede derivar a otras dos respuestas y así sucesivamente n veces y como solo puede tomar dos caminos equivale a 2

7. 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	0m0.038s	45	0m0.033s
10	0m0.037s	50	0m0.052s
15	0m0.037s	100	0m0.036s
20	0m0.030s	200	0m0.027s
25	0m0.056s	500	0m0.041s
30	0m0.033s	1000	0m0.035s
35	0m0.027s	5000	0m0.024s
40	0m0.046s	10000	0m0.058s



8. Ejecute la operación `mostrarPrimos` que presentó en su solución al ejercicio 4 de la Tarea 1 y también la versión de la solución a este ejercicio que se subirá a la página del curso con los siguientes valores y mida el tiempo de ejecución:

Tamaño Entrada	Tiempo Solución Propia	Tiempo Solución Profesores
100	0.032s	0.043s
1000	0.058s	0.044s
5000	1.205s	0.044s
10000	3.997s	0.120s
50000	1m27.207s	1.050s
100000	...	2.958s
200000	...	6.544s

Responda las siguientes preguntas:

(a) ¿qué tan diferentes son los tiempos de ejecución y a qué cree que se deba dicha diferencia?
Los tiempos de ejecución a medida que aumenta los tamaños de entrada se nota más la diferencia y yo creo que se debe a mi manejo de ciclos y a la diferencia de la complejidad computacional que hay entre los códigos de los profesores y el mio.

(b) ¿cuál es la complejidad de la operación o bloque de código en el que se determina si un número es primo en cada una de las soluciones?

Código punto 6 y punto 7:

#Punto 6

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

#Punto 7

```
def fibo(n):
        nuevo = 0
        viejo = 1
        for k in range(n):
                ans = viejo + nuevo
                nuevo = viejo
                viejo = ans
        return nuevo
```