Carlos Daniel Vega Cardenas.

Estructuras de Datos

Profesores: Gonzalo Noreña - Carlos Ramírez Tarea 2

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:

```
void algoritmo1(int n){
    int i, j = 1;
    for(i = n * n; i > 0; i = i / 2){
        int suma = i + j;
        printf("Suma %d\n", suma);
        ++j;
    }
}
```

La complejidad de este algoritmo es de: O(n).

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:

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

```
for(i = 1; i <= 2 * n; i += 4)
    for(j = 1; j * j <= n; j++)
    res += 2;
```

La complejidad de este algoritmo es de: O(n^2).


```
    return res;
}
```

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:

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

La complejidad de este algoritmo es de: O(n^3).

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; c

    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;
}
```

La complejidad de este algoritmo es de: O(n^3).

¿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:

```
void algoritmo5(int n){
    int i = 0;
    while(i <= n){
        printf("%d\n", i);
        i += n / 5;
    }
}
```

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	0.0	35	53.78928542137146
10	0.0	40	586.1382076740265
15	0.008019447326660156	45	5262.655983924866
20	0.04025721549987793	50	
25	0.4481682777404785	60	
30	4.837419509887695	100	

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
print(fib(45))
```

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?

1. **45**
2. **Entre más grande sea el número ingresado más tiempo demora en progresar.**
3. **Creo que la complejidad del algoritmo es de: O(1).**

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	0.0	45	0.0
10	0.0	50	0.0009999275207519531
15	0.0	100	0.0

20	0.0	200	0.0009999275207519531
25	0.0	500	0.0
30	0.0	1000	0.00099945068359375
35	0.000997304916381836	5000	0.0029985904693603516
40	0.0010001659393310547	10000	0.005002498626708984

```python
def ciclosFibonacci(n):
    a = 0
    b = 1
    for i in range(n):
        a, b = b, a + b
    return a
print(ciclosFibonacci(10000))
```

Complejidad: O(n).

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	0.0049974918365478516	0.005005598068237305
1000	0.07400774955749512	0.030005455017089844
5000	0.9574038982391357	0.13412046432495117
10000	3.0428829193115234	0.31865644454956055
50000	58.455228090286255	1.5894725322723389
100000	214.0614995956421	3.3830618858337402
200000	791.3937180042267	7.359010219573975

Responda las siguientes preguntas:

(a) ¿Qué tan diferentes son los tiempos de ejecución y a qué cree que se deba

dicha diferencia?

Se puede ver una amplia diferencia en el tiempo de ejecución, debido a la complejidad del algoritmo.

(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?

Solución propia: O(n^2).

Solución profesores: O(n).