Tarea 2 - Nicolas Vidal

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

Complejidad computacional

$$\begin{aligned} T(n) &= (2 * log(n) + 1) + 3 * (2 * log(n) + 1) \\ &= (2 * log(n)) + 2 + (6 * log(n)) + 3 \\ &= 8 * log(n) + 5 \end{aligned}$$

$$\therefore$$

$$T(n) = O(log(n))$$

algoritmo1(8)

```
i = 64; j = 1; suma = 65
i = 32; j = 2; suma = 34
i = 16; j = 3; suma = 19
i = 8; j = 4; suma = 12
i = 4; j = 5; suma = 9
i = 2; j = 6; suma = 8
i = 1; j = 7; suma = 8
```

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; // ---------------> 3
```

```
    for(i = 1; i <= 2 * n; i += 4) // ---> (n/2)+1
        for(j = 1; j * j <= n; j++) // --> (n/2)*(n^(0.5)+1)
            res += 2;// -----------------> (n^(0.5))*(n/2)
    return res; // ----------------------> 1
}
```

Complejidad computacional

$$
\begin{aligned}
T(n) &= 3 + \frac{n}{2} + 1 + \left(\frac{n}{2}\sqrt{n} + 1\right) + \left(\sqrt{n}\frac{n}{2}\right) \\
&= 3 + \frac{n}{2} + 1 + \left(\frac{n\sqrt{n}}{2} + 1\right) + \frac{n\sqrt{n}}{2} \\
&= 3 + \frac{n}{2} + 1 + n\sqrt{n} + 1 \\
&= 5 + \frac{n}{2} + n\sqrt{n} \\
&= n^{\frac{3}{2}} + \frac{n}{2} + 5
\end{aligned}
$$

algoritmo1(8)

```
i = 1:
    j = 1:
      res += 2;
      res = 3;
    j = 2:
      res += 2;
      res = 5;
i = 5:
    j = 1:
      res += 2;
      res = 7;
    j = 2:
      res += 2;
      res = 9;
i = 9:
    j = 1:
      res += 2;
      res = 11;
    j = 2:
      res += 2;
      res = 13;
i = 13:
    j = 1:
      res += 2;
      res = 15;
    j = 2:
      res += 2;
```

```
        res = 17;
```

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; // ----------------------> 3
    for(i = n; i > 1; i--){ // -----------> (n-1)+1 = n
        for(j = 1; j <= n; j++){ // ------> (n+1)*anterior_linea = (n+1)*(n-1)
            for(k = 1; k <= i ; k++){ //--> Sum{n->2}(i+1)+1 *anterior_linea = Sum{n-
              printf("Vida cruel!!\n");//-> Sum{2->n}(i+1) *(n*(n-1))
            }
        }
    }
}
```

Usando las propiedades:

$$\sum_{i=2}^{n}(i+1) = \sum_{i=2}^{n} i + \sum_{i=2}^{n} 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=2}^{n} i = \sum_{i=1}^{n} i - \sum_{i=1}^{1} i$$

$$T(n) = 3 + n + [(n+1) * (n-1)] + [(n * (n-1)) * (\sum_{i=2}^{n}(i+1) + 1)] + [(n * (n-1)) * (\sum_{i=2}^{n}(i+1))]$$

$$= \dots$$

$$= n^4 + 3n^3 - 8n^2 + 6n + 2$$

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; //------------> 2
  int i, j, h, flag;//----------------------> 4
```

```
  for(i = 0; i < n; i++){//-----------------> n+1
    j = i + 1;//---------------------------> n
    flag = 0;//----------------------------> n
    while(j < n && flag == 0){ //-----------> Sum{1->n-1}(t_j)
      if(valores[i] < valores[j]){ //-------> Sum{1->n-1}(t_j-1)*Sum{1->n-1}(l_j)
        for(h = j; h < n; h++){ //----------> n-j+1*anterior_linea = (n-j-1)*Sum{1->n
          suma += valores[i]; //------------> n-j*anterior_linea = (n-j)*Sum{1->n-1}(
        }
      }else{ //-----------------------------> Sum{1->n-1}(g_j)*Sum{1->n-1}(t_j-1)
          contador++; //--------------------> Sum{1->n-1}(g_j)*Sum{1->n-1}(t_j-1)
          flag = 1; //----------------------> Sum{1->n-1}(g_j)*Sum{1->n-1}(t_j-1)
      }
      ++j; //-------------------------------> Sum{1->n-1}(t_j-1)
    }
  }
  return contador;
}
```

Donde $\sum_1^{n-1} t_j$ = veces que se corre el ciclo `while` en el peor, promedio y mejor escenario

tambien, $\sum_1^{n-1} l_j$ = veces que se corre el condicional `if` en el peor, promedio y mejor escenario

por ultimo, $\sum_1^{n-1} g_j$ = veces que se corre el condicional `else` en el peor, promedio y mejor escenario

Es importante notar que g_j y l_j son excluyentes, por lo que el mejor escenario de g es es peor de j, sucesivamente...

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; //-----------> 1
  while(i <= n){ //-------> 6
    printf("%d\n", i); //-> 5
    i += n / 5; //--------> 5
  }
}
```

$$T(n) = 1 + 6 + 5 + 5$$
$$= 17$$

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	0m1.157s	35	0m5.537s
10	0m1.653s	40	0m29.764s
15	0m1.100s	45	4m58.719s
20	0m4.078s	50	Mucho tiempo
25	0m2.454s	60	Mucho tiempo
30	0m3.629s	100	Mucho tiempo

Cuál es el valor más alto para el cuál pudo obtener su tiempo de ejecución?

> 45

Qué puede decir de los tiempos obtenidos? Cuál cree que es la complejidad del algoritmo?

> O(2^n)
>
> Esto se debe a que el calculo se realiza dividiendo los llamados en 2 por nivel.

```
def fibonacci(n):
    if n < 0:
        print("Incorrecto")
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)

fibonacci(int(input('num: ')))
```

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	0m0.873s	45	0m0.904s
10	0m1.168s	50	0m1.638s
15	0m2.247s	100	0m1.534s
20	0m1.623s	200	0m1.522s
25	0m1.877s	500	0m4.295s
30	0m3.445s	1000	0m2.179
35	0m2.626s	5000	0m3.329s
40	0m0.750s	10000	0m2.025s

```
def fibonacci(n):
    a,b = 0,1
    for i in range(n):
        a,b = b,a+b
    return a

print(fibonacci(int(input('num: '))))
```

Complejidad:

O(n)

8. Ejecute la operación mostrarPrimos que presentó en su solución al ejercicio 4 de la Tarea 1.

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	0m1.907s	0m1.760s
1000	0m1.136s	0m1.236s
5000	0m2.234s	0m2.594s
10000	0m1.840s	0m1.638s
50000	0m11.865s	0m2.948s
100000	0m43.617s	0m2.748s
200000	2m33.961s	0m3.627s

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

deba dicha diferencia?

Al inicio los tiempos de ambos algoritmos son muy similares. Sin embargo al aumentar el n la diferencia de tiempos aumenta.

A partir de los 50_000 mi algoritmo incrementa el tiempo de forma casi exponencial. Mientras tanto, el tiempo del algoritmo propuesto por los profesores aumenta con una aceleracion muy baja, casi constante.

Esto se debe a que mi algoritmo tiene una complejidad computacional de categoria O(n^3) mientras que la propuesta de los profesores tiene una complejidad menor que esta.

(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?

```
def esPrimo(n):
  if n < 2: ans = False #------------> 1
  else: #----------------------------> 1
    i, ans = 2, True#----------------> 2
    while i * i <= n and ans:#-------> Peor_caso: sqrt(n)+1; mejor_caso: 1
      if n % i == 0: ans = False # --> sqrt(n)
      i += 1 # ---------------------> sqrt(n)
  return ans # ---------------------> 1
```

Peor caso

$$\begin{aligned} T(n) &= 1 + 1 + 2 + (\sqrt{n} + 1) * (2\sqrt{n}) + 1 \\ &= 2(\sqrt{n})^2 + 2\sqrt{n} + 5 \\ &= 2\sqrt{n} + 2n + 5 \end{aligned}$$

Mejor caso

$$\begin{aligned} T(n) &= 1 + 1 + 2 + (1) * (2\sqrt{n}) + 1 \\ &= 2\sqrt{n} + 5 \end{aligned}$$