

Tarea2

Kevin Yavarí 8976525

Julián Galvis 8974868

Punto 1

```
void algoritmo1(int n){
```

```
    int i, j = 1;     $\longrightarrow$  2
```

```
    for(i = n * n; i > 0; i = i / 2){  $\longrightarrow$   $\log_2(n^2)+2$ 
```

```
        int suma = i + j;
```

```
        printf("Suma %d\n", suma);
```

```
        ++j;
```

```
    }
```

```
}
```

La complejidad de la función algoritmo1 es $O(\log_2(n^2))$

Al ejecutar este procedimiento con 8 se obtiene:

- Suma 65
- Suma 34
- Suma 19
- Suma 12
- Suma 9
- Suma 8
- Suma 8

Punto 2

```
int algoritmo2(int n){
```

```
    int res = 1, i, j;     $\longrightarrow$  3
```

```
    for(i = 1; i <= 2 * n; i += 4)     $\longrightarrow$   $n/2 + 1$ 
```

```
        for(j = 1; j * j <= n; j++)     $\longrightarrow$   $(n/2)*\sqrt{n} + 1$ 
```

```
            res += 2;     $\longrightarrow$   $(n/2) * \sqrt{n}$ 
```

```
    return res;     $\longrightarrow$  1
```

```
}
```

La complejidad de la función algoritmo2 es $O(n * (\sqrt{n}))$

Punto 3

```
void algoritmo3(int n){
```

```
    int i, j, k;       $\longrightarrow$       3
```

```
    for(i = n; i > 1; i--)       $\longrightarrow$       n
```

```
        for(j = 1; j <= n; j++)       $\longrightarrow$       n - 1 (n + 1)
```

```
            for(k = 1; k <= i; k++)       $\longrightarrow$        $\sum_{i=1}^{n-1} i + 1$ 
```

```
                printf("Vida cruel!!\n");       $\longrightarrow$        $\sum_{i=1}^{n-1} i$ 
```

```
}
```

La complejidad de la función algoritmo3 es $O(n^3)$ por sus 3 ciclos

Punto 4

```
int algoritmo4(int* valores, int n){
```

```
    int suma = 0, contador = 0;       $\longrightarrow$       2
```

```
    int i, j, h, flag;       $\longrightarrow$       4
```

```
    for(i = 0; i < n; i++){       $\longrightarrow$       n + 1
```

```
        j = i + 1;       $\longrightarrow$       n
```

```
        flag = 0;       $\longrightarrow$       n
```

```
        while(j < n && flag == 0){       $\longrightarrow$ 
```

```
            if(valores[i] < valores[j]){
```

```
                for(h = j; h < n; h++){
```

```
                    suma += valores[i];
```

```
                }
```

```
            }
```

```
        else{
```

```
            contador++;
```

```
            flag = 1;
```



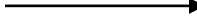

```
        }
```

```
        ++j;
```

```
    }
```

```
}
```

```
return contador;
}
```

```
void algoritmo5(int n){
    int i = 0;  1
    while(i <= n){  7
        printf("%d\n", i);  6
        i += n / 5;  6
    }
}
```

Es una función contante por lo tanto su complejidad es $O(1)$

Punto 6

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0,155s	35	4,210s
10	0,192s	40	43,025s
15	0,200s	45	8m 16,166s
20	0,210s	50	Indefinido
25	0,280s	55	Indefinido
30	0,700	60	Indefinido

El valor mas alto que pudimos ejecutar fue el 45, se puede ver que aunque se aumente poco se puede ver como el tiempo se eleva exponencialmente, diría que la complejidad es $O(2^{**}n)$ ya que se usa la recursión y se van sumando los últimos 2 numeros

Punto 7

Tamaño de entrada	tiempo	Tamaño de entrada	tiempo
5	0,100s	45	0,150s
10	0,142s	50	0,178s
15	0,160s	100	0,135s
20	0,145s	200	0,145s
25	0,170s	500	0,140s
30	0,187s	1000	0,143s
35	0,189s	5000	0,210s
40	0,139s	10000	0,550s

La complejidad es $O(n)$ ya que el ciclo for itera n veces

Punto 8

Tamaño Entrada	Tiempo Solución Propia	Tiempo Solución Profesores
100	0,375s	0,158s
1000	0,687s	0,486s
5000	0,454s	0,501s
10000	0,900s	0,313s
50000	12,123s	0,571s
100000	13,323s	1,903s
200000	13,780s	1,504s

A) La forma del profesor para encontrar los números primos es más eficiente

B)

```
def esPrimoProfesor(n):
    if n < 2: ans = False
    else:
        i, ans = 2, True
        while i * i <= n and ans:
            if n % i == 0: ans = False
            i += 1
    return ans
```

- en el mejor caso la complejidad del programa es $O(1)$ ya que solamente ejecutaría el if y nunca entraría al while
- en el peor caso $i*i$ sería mayor a n y saldría del ciclo, sabiendo que $i*i$ es i^2 , se calcularía la raíz de n para saber cuántas veces se ejecuta el while en el peor caso posible, dando una complejidad de $O(\sqrt{n})$

```
def siEsPrimoEstudiante(a):
    ans = True
    if a <= 1:
```

```
        ans = False
i = 2
while i < a and ans:
    if a % i == 0:
        ans = False
    i+=1
return ans
```

- Por el contrario en el código realizado por el estudiante, se puede ver como el ciclo a pesar de la condición siempre se ejecutara por lo menos una vez, por lo que en el mejor de los casos se daría una complejidad constante, pero en el peor de los casos se daría una complejidad de $O(n)$, que en comparación con la del profesor, generando un gran cambio en casos donde n sea un numero bastante grande.