David Bayona, Alejandro Posada N.

Puntos de estructura de datos

1. En el algoritmo también se puede apreciar que el número de iteraciones depende del tamaño de la entrada, dicha entrada se procesa de modo que se le suma +1 y al valor original se le divide.

   Para el caso del algoritmo 1(8) tenemos que llegar a un punto donde el ciclo no puede hacer más divisiones entre 2 al tratarse de un 1, el 8 que queda al final se presenta debido a que, mientras se reiteraba el ciclo la otra variable se iba sumando de uno en uno.

   Al tomar como parámetro el 8 esta variable que se iba sumando solamente alcanza el valor de 7, este 7 se suma con el 1 resultante de las divisiones y entrega un segundo 8 al final.

   A continuación una tabla que podría ilustrar esta explicación:

Valor original al cuadrado (8^2)	**Variable tipo contador**	**Resultante de la suma**
64	1	65
32	2	34
16	3	19
8	4	12
4	5	9
2	6	8
1	7	8

Complejidad Teórica

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

2	
$\log_2 n^2$	
$\log_2 n^2 - 1$	
$\log_2 n^2 - 1$	
$\log_2 n^2 - 1$	

R\ 4 $\log_2 n^2 - 1$ $O(\log_2 n^2)$

① ~~$2^k = n^2$~~ $\log_2 n^2$

2.

```
→ int algoritmo2 (int n){
    int res = 1, i, j;
①   for (i=1; i<=2*n; i+=4)
②       for (j=1; j*j<=n; j++)
            res += 2;
    Return res;
}
```

$$// \; 3 + \frac{n}{2} + 1 + 2\left(\sqrt{n} \cdot \frac{n}{2} - \frac{n}{2} + \sqrt{n} - 1\right) + 1$$

$$4 + \frac{n}{2} + n^{3/2} - n + 2\sqrt{n} - 2$$

$$n^{3/2} - n + \frac{n}{2} + 2\sqrt{n} + 2$$

$$n^{3/2} - n/2 + 2\sqrt{n} + 2 \rightarrow O(n^{3/2})$$

① $\frac{2n}{4}$ → por el avance $\quad i <= 2 \cdot n \quad +1$

② $j^2 <= n \;=\; j <= \sqrt{n} \quad -1$

$\frac{2n+2}{4} = \frac{n}{2} + 1$

$(\sqrt{n} - 1)\left(\frac{n}{2} + 1\right)$

$(\sqrt{n} - 1)\left(\frac{n}{2} + 1\right)$

P. ej. $4 \rightarrow 7 \rightarrow 76$ (9 → 1 → 76)

Para el caso de **algoritmo 2(8)** ocurre que el primer ciclo for usa una variable "**i**" que empieza en 1, la cual se va sumando de 4 en 4 y esta variable tiene la restricción de que la iteración **i <= 2 * n**.
Dada esta restricción vamos viendo que al empezar en 1 y se va sumando de 4 en 4 nos encontramos con esta secuencia:

1 , 5 , 9 , 13 , 17

Dijimos hace un momento que la variable **"i"** debía de ser mayor o igual a **2 * n**, al tener como entrada un 8 nos daría que **i <= 16**, pero como se puede apreciar termina en un 17 ya que sobrepasa el tope en un intento de cumplir el requisito que se plantea en el primer ciclo for.

En el caso del segundo ciclo podemos observar que tiene como límite la condición de que la variable **"j"** al cuadrado debe ser menor o igual a **"n"**. (**j * j <= n**).

Para este ciclo podemos apreciar que ocurre lo mismo que en el anterior, **"j"** se va sumando en **+1** partiendo de su valor original siendo este un **1**. Al momento de llegar a **3** ya excede el límite **j * j <= n**.

Finalmente se logra apreciar que la variable "**res"** se va sumando a la medida que se va sumando "**j"**

3.

```
void algoritmo3(int n){
    int i, j, k;
①   for(i=n; i>1; i--)
②       for(j=1; j<=n; j++)
③           for(k=1; k<=i; k++)
                Print("Vida Cruel :(\n");
}
```

3

n

$n^2 - n$ es Parametro

$n^2 - 1$

$n^2 - 1$

① n // n>1 -1 (1), 1, -1

② n, 1-1 // n+1-1 (1, n+1) n

③ (k, k<=i; + ① ↑ ③ ↓

R\\ 3 + n + n - 1 + n - 1

3n + 1 O(lineal) Θ(n)

4.

```
int algoritmo7(int* valores, int n) {
    int suma=0, contador=0;                    2
    int i, j, h, flag;                         4
    for (i=0; i<n; i++) {                      n
        j= i+7;                                n-7
        flag=0;                                n-7
        while(j<n && flag==0) {                n^2
            if(valores[i]<valores[j]) {        n^2
                for (h=j; h<n; h++) {          n^3
                    suma+=valores[i];          n^3
                }
            }
        else {
            contador++;                        7
            flag=7;                            7
        }
    }
    return contador                            7
}
```


5.

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

1
5 + 1
5
5

R\ 1 + 5 + 1 + 5 + 5 O(constante)

R\ 17

6.

```python
from timeit import default_timer

def fiboo(n):
    if n<2:
        return n
    return fiboo(n-1)+fiboo(n-2)

for i in range(5):
    print(i ,",",fiboo(i))

inicio = default_timer()

fiboo(40)

fin = default_timer()

print("Tiempo: " , fin - inicio)

```

Tamaño Entrada	Tiempo	Tamaño entrada	Tiempo
5	2,78	35	2,59
10	2,83	40	23,87
15	0,00028	45	25,02
20	0,0024	50	525,02
25	0,023	60	
30	0,35	100	

7.

```python
from timeit import default_timer


def fibo(n):

    sucesion = [0,1,1]

    for i in range(n):
        sucesion.append(sum(sucesion[-2:]))
    return '\n'.join(str(e) for e in sucesion)

inicio = default_timer()

print(fibo(10000))



fin = default_timer()

print("Tiempo: " , fin - inicio)
```

(variable) inicio: Any

Tamaño Entrada	Tiempo	Tamaño entrada	Tiempo
5	0,0004	45	0,0006
10	0,0003	50	0,0007
15	0,0009	100	0,0009
20	0,0002	200	0,0023
25	0,0003	500	0,0017
30	0,0004	1000	0,04
35	0,0005	5000	0,06
40	0,0003	10000	0,28

8.

```python
from timeit import default_timer

def mostrarPrimos(cantidad):
    primos=[]
    for i in range(2, cantidad):
        primos.append(i)
        for j in range(2,i):
            if (i%j)==0:
                if i in primos:
                    primos.remove(i)

    print(primos)

mostrarPrimos(100000)

inicio = default_timer()

fin = default_timer()

print("Tiempo: " , fin - inicio)
```

Tamaño	Solución propia	Solución profe
100	6,99	2,00
1000	1,20	3,00
5000	1,49	4,99
10000	8,99	2,00
50000	8,99	2,00

100000	8,99	7,99
200000		3,99

A. Los tiempos de ejecución tienen una significativa diferencia, esto debido a que la solución planteada por el profesor se empeña más en saber manejar una mayor eficiencia, mientras que el otro código se centra más en cumplir la tarea.

B. La complejidad de nuestro código es de O(n^2) Dado que se hacen dos ciclos, uno que crea números, y otro que los elimina. Por otro lado, el ciclo de los profes tiene una complejidad de O(n), por ende, es mucho más rápido.