Trabajo 2 Estructura de datos

1.

```
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;
   }
}
```

```
void algoritmo1(int n){
        int i, j = 1;          →  1
        for(i = n * n; i > 0; i = i / 2){  →  log2(n*n)
                int suma = i + j;  → log2(n*n)
                printf("Suma %d\n", suma);  → log2(n*n)
                ++j;   →  log2(n*n)
        }
}
```

Respuesta: O(log n)

2.

```
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;


   return res;
}
```

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

Respuesta: O(n * sqrt(n))

3.

```
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");
}
```

```
void algoritmo3(int n) {
        int i, j,  k;  → 1
        for (i = n; i > 1; i- -){ →  n - 1
                for (j = 1; j <= n; j++){ →  n(n - 1)
                        for (k = 1; k <= i; k++){ → (n/2) * (n + 1)
                                printf("Vida Cruel!!\n "); → (n/2) * (n + 1)

                        }
                }
        }
}
```

Respuesta: $O(n^3)$

4.

```
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;
}
```

```
int algoritmo4(int* valores, int n){
        int suma = 0, contador = 0;  → 1
        int i, j, h, flag; → 1

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

Respuesta: O(n^2),

5.

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

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

Respuesta: O(n)

6.

```
4
5   def fibonacci(n):
6       if n == 0:
7           return 0
8       elif n == 1:
9           return 1
10      else:
11          return fibonacci(n-1) + fibonacci(n-2)
12
```

n = 5: 3.6400000681169 33e-05 segundos
n = 10: 0.0001588000013725832 segundos
n = 15: 0.0013054000010015443
n = 20: 0.014397399994777516 segundos
n = 25: 0.1534274000005098 segundos
n = 30: 1.6920851999893785 segundos
n = 35: 19.096350000007078 segundos
n = 40: 211.81974909998826 segundos

Cuál es el valor más alto para el cuál pudo obtener su tiempo de ejecución?
El valor más alto que llegue con el programa funcionando fue *n* = 40 después de esto el programa se estaba demorando demasiado o se cerraba.

¿Qué puede decir de los tiempos obtenidos?

Según los resultados pude notar que entre mas subia *n* mucho más tiempo se demoraba el programa en ejecutar, como se puede ver la diferencia entre *n* = 35 y *n* = 40 fue de casi de tres minutos.

¿Cuál cree que es la complejidad del algoritmo?

Lo que yo creo que es la complejidad del algoritmo es que sea de tipo exponencial ya que si se compara y un valor con el anterior la diferencia es de varias veces ese número multiplicado o exponenciado.

7.

```
14  def fibonacci2(n):
15      n1, n2 = 0, 1
16      count = 0
17
18      if n <= 0:
19          print("Please enter a positive integer")
20      elif n == 1:
21          return 0
22      else:
23          while count < n:
24              nth = n1 + n2
25
26              n1 = n2
27              n2 = nth
28              count += 1
```

Complejidad: O(2^n)

Tiempo de ejecución para n=5: 1.5700003132224083e-05 segundos
Tiempo de ejecución para n=10: 1.6000005416572094e-05 segundos
Tiempo de ejecución para n=15: 9.399998816661537e-06 segundos
Tiempo de ejecución para n=20: 7.099995855242014e-06 segundos
Tiempo de ejecución para n=25: 1.3500000932253897e-05 segundos
Tiempo de ejecución para n=30: 1.5900004655122757e-05 segundos
Tiempo de ejecución para n=35: 1.0700008715502918e-05 segundos
Tiempo de ejecución para n=40: 1.379998866468668e-05 segundos
Tiempo de ejecución para n=45: 1.5099998563528061e-05 segundos
Tiempo de ejecución para n=50: 1.340000017080456e-05 segundos
Tiempo de ejecución para n=100: 2.5000001187436283e-05 segundos
Tiempo de ejecución para n=200: 5.639999289996922e-05 segundos
Tiempo de ejecución para n=500: 0.00011920000542886555 segundos
Tiempo de ejecución para n=1000: 0.000255300008575432 segundos
Tiempo de ejecución para n=5000: 0.0013119999930495396 segundos
Tiempo de ejecución para n=10000: 0.003482799991616048 segundos

8.

Tiempo Solución Profesores:

Tiempo de ejecución para n = 100: 0.009248600341379642 segundos
Tiempo de ejecución para n = 1000: 0.06489520007744431 segundos
Tiempo de ejecución para n = 5000: 0.44934079982340336 segundos
Tiempo de ejecución para n = 10000: 0.7259463001973927 segundos
Tiempo de ejecución para n = 50000: 2.915684400126338 segundos
Tiempo de ejecución para n = 100000: 5.545513699762523 segundos
Tiempo de ejecución para n = 200000: 10.000347199849784 segundos

Tiempo Solución Propia:

Tiempo de ejecución para n = 100: 0.012235600035637617 segundos
Tiempo de ejecución para n = 1000: 0.09418489970266819 segundos
Tiempo de ejecución para n = 5000: 0.5693651000037789 segundos
Tiempo de ejecución para n = 10000: 1.2167956996709108 segundos
Tiempo de ejecución para n = 50000: 11.801186900120229 segundos
Tiempo de ejecución para n = 100000: 37.00694190012291 segundos
Tiempo de ejecución para n = 200000: 123.21837779972702 segundos

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

Los tiempos de ejecución comienzan bastante similares hasta n = 10000 después de eso la brecha entre el tiempo de ejecución de la solución propia y la solución de los profesores se vuelve mucho más exponencial. Yo creo que eso está pasando ya que un algoritmo es más eficiente que el otro, esto puede por por la cantidad de iteraciones totales que hace o por la capacidad de memoria usada.

(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?*

La complejidad del bloque de código que determina si un número es primo es:

En la solución propia: O(n) y en la solución del profesor: O(sqrt(n))

Esto demuestra el porqué la segunda solución es más eficiente que la primera ya que O(sqrt(n)) >O(n)