Tarea 2 Estructuras de Datos

1.

```
void algoritmo1(int n){

int i, j = 1;    -> 1

for(i = n * n; i > 0; i = i / 2){  -> log2(nn)

int suma = i + j;   -> log2(nn)

printf("Suma %d\n", suma); -> log2(nn)

++j; -> log2(nn)

}

}
```

Si se ejecuta la función algoritmo1(8), se obtendrá la siguiente salida en la consola:

Suma 65 Suma 34 Suma 18 Suma 10 Suma 6 Suma 4 Suma 3 Suma 2

En términos generales, este algoritmo calcula una serie de sumas de dos números, donde el primer número de cada suma es el valor de una variable "i" que se inicializa en "n * n" y se divide por dos en cada iteración, y el segundo número de cada suma es el valor de una variable "j" que se inicializa en 1 y se incrementa en una unidad en cada iteración.

Respuesta: O(log n)

2.

```
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))

Si se ejecuta la función algoritmo2(8), se obtendrá el valor 19 como resultado. Esto se debe a que el algoritmo realiza un par de bucles anidados para calcular el valor de "res". El primer bucle for se ejecuta "2n / 4 = n/2" veces, y en cada iteración incrementa "i" en 4 unidades. El segundo bucle for se ejecuta "sqrt(n)" veces, ya que "j" se incrementa de 1 en 1 hasta que su cuadrado es mayor que "n".

Dentro del segundo bucle for, se suma 2 al valor de "res" en cada iteración. Por lo tanto, el número total de veces que se suma 2 a "res" es igual al producto del número de iteraciones de ambos bucles: "n/2 * sqrt(n)".

3.

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(n3)

4.

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 veces

j = i + 1; -> n veces

flag = 0; -> n veces

while(j < n && flag == 0){ -> n(n-1)/2 veces

if(valores[i] < valores[j]){ -> n(n-1)/2 veces

for(h = j; h < n; h++){ -> sum(j to n-1)

suma += valores[i]; -> sum(j to n-1)

Respuesta: O(n^3)

La función "algoritmo4" recibe un arreglo de enteros "valores" de tamaño "n" como entrada. Luego, inicializa una variable "suma" y otra variable "contador" en cero. A continuación, se realiza un bucle for que recorre el arreglo de valores desde el índice 0 hasta n-1.

Dentro del bucle for, se inicializa una variable "j" en "i + 1" y una bandera "flag" en cero. Luego, se ejecuta un bucle while que se ejecuta siempre que "j" sea menor que "n" y "flag" sea igual a cero.

Dentro del bucle while, se compara el valor en la posición "i" del arreglo "valores" con el valor en la posición "j". Si el valor en la posición "i" es menor que el valor en la posición "j", se ejecuta un bucle for que recorre el arreglo desde "j" hasta "n-1" y suma el valor en la posición "i" a la variable "suma" en cada iteración. Si el valor en la posición "i" no es menor que el valor en la posición "j", se incrementa la variable "contador" en uno y se establece la bandera "flag" en 1, lo que hace que el bucle while se detenga.

Finalmente, la función devuelve el valor de la variable "contador", que representa el número de veces que el valor en la posición "i" no es menor que el valor en la posición "j".

En términos generales, este algoritmo cuenta el número de pares de elementos en un arreglo "valores" donde el elemento en la posición "i" no es menor que el elemento en la posición "j", donde "j" es mayor que "i". Además, en cada par de elementos donde el elemento en la posición "i" no es menor que el elemento en la posición "j", se suma el valor del elemento en la posición "i" a una variable "suma".

5.

```
void algoritmo5(int n){

int i = 0; -> 1

while(i <= n){  -> log_5(n)+1

printf("%d\n", i);  -> log_5(n)+1

i += n / 5; -> log_5(n)+1
```

respuesta: O(log n)

6.

Tamaño de entrada	Tiempo (Segundos)	Tamaño de entrada	Tiempo (Segundos)
5	3.5299977753311396e-05	35	18.77166839997517
10	0.00014150000060908496	40	208.9529056000174
15	0.0013107999984640628	45	Demasiado
20	0.01396710000699386	50	Demasiado
25	0.152105399989523	60	Demasiado
30	1.6849952000193298	100	Demasiado

7.

Tamaño de entrada	Tiempo	Tamaño de entrada	Tiempo
5	1.5199999324977398e-05	45	1.5900004655122757e-05

10	7.599999662488699e-06	50	1.7099984688684344e-05
15	8.499977411702275e-06	100	2.929999027401209e-05
20	9.499985026195645e-06	200	4.839999019168317e-05
25	2.1199986804276705e-05	500	0.00011079999967478216
30	8.69997893460095e-06	1000	0.00023030000738799572
35	1.270000939257443e-05	5000	0.0012997000012546778
40	1.5300000086426735e-05	10000	0.003396500018425286

8.

Tamaño de entrada	Tiempo solución propia (Segundos)	Tiempo Solución profesores (Segundos)
100	0.012235605015133257	0.009248600341379642
1000	0.092641849972668142	0.06489520007744431
5000	0.4883451206147789	0.44934079982340336
10000	1.0021566994886141	0.7259463001973927
50000	10.057886900110279	2.915684400126338
100000	35.01624140042211	5.545513699762523
200000	120.73537549471752	10.000347199849784

a.

Como se puede ver en la tabla mis tiempos son mayores en todos los casos no por mucho en los primeros, pero en los últimos como el 50000,100000,200000 ya se nota bastante más llegando a una diferencia de 110 segundos en la última prueba, esta diferencia supongo que se debe a que el codigo del profesor puede estar mejor optimizado o mejor hecho.