Tarea 2

Nombres: Jorge Casas, Juan Camilo Quiceno

Grupo A

1.

Int I, j = 1; Se repite 1 vez

for(i = n * n; i > 0; i = i / 2) se repite n+1 veces

int suma = i + j; se repite n+1 veces

printf("Suma %d\n", suma); n+1 veces

 ++j; se repite n+1

Teniendo en cuenta que i = n * n que en este caso es 8, tendremos que i = 64. Cuando se entra al ciclo tenemos que suma = i+j, que sería 64 + 1 respectivamente. El procedimiento continuaría, con i reduciéndose a la mitad por cada iteración, mientras que j incrementa por 1 al final de la iteración. El valor de suma es menor por cada iteración, hasta que llega i = 1, que corresponde al final del for, y con un resultado de suma = 8, porque suma = 1 + 7

Complejidad:

$$\text{Cuando } i = n \rightarrow n^2$$

$$= \frac{n^2}{2}$$

$$= \frac{n^2}{4}$$

$$\ldots \text{ se repite}$$

$$= log_2 (n^2)$$

$$= 2 * log_2 (n)$$

$$= O(log_n)$$

2.

Int res = 1, i, j; Se repite 1 vez

For (i = 1; i <= 2 * n; i += 4) se repite n/2 veces

For (j = 1; i <= 2 * n; i += 4) se repite n veces

Res += 2; se repite n veces

Return res; se repite 1 vez

Se retorna res = 17, entrando a ambos for, hay que tener en cuenta que para cada iteración en el primer for, se repite el segundo for 2 veces por la condición planteada de j * j < n (1*1 y 2*2, 3*3 se pasa de eso por lo que no entra), por lo tanto, se le suma 2 veces 2 a res por cada iteración de i. Esto se va repitiendo hasta 4 iteraciones del for de i, porque en la condición planteada de que i < n * 2 (que sería 16) y que se le suma 4 a i por iteración, llegando a 17 que sería el final del ciclo, llegando a 17. De forma resumida, se puede decir que a res se le suma 2, 2 veces, lo que sería 4 por iteración en j, por 4 veces que se realiza el for en i te da 16, sin embargo se le suma 1 teniendo en cuenta que res se inicializa en 1.

Complejidad:

$$n$$
$$= \frac{n}{2} * (n)$$
$$= \frac{n^{\frac{3}{2}}}{2}$$
$$= \frac{n}{2} * n^{\frac{3}{2}}$$
$$= n^{\frac{5}{2}}$$
$$= O(n^{\frac{5}{2}})$$

3.

Int i, j, k; se repite 1 vez

For (i = n; i > 1; i--) se repite n – 1 veces

For (j = 1; j <= n; j++) se repite n veces

For (k = 1; k <= i; k++) se repite i veces

Printf(“Vida Cruel!!\n”); se repite i veces

Complejidad: Al haber 3 bucles anidados, se llega a que n^3

$$n$$

$$= n - i$$

$$= n * (n - 1) * \frac{(n + 1)}{2}$$

$$= n^3$$

$$= O(n^3)$$

4. Int suma = 0, Contador = 0; se repite 1 vez

Int i, j, h, flag; se repite 1 vez

For (I = 0; I < n; i++) se repite n veces

j = i + 1; se repite n veces

flag = 0; se repite n veces

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; se repite 1 vez

Complejidad

$$n$$

$$= n - j$$

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

$$= n^3$$

$$= O(n^3)$$

El algoritmo calcula dos cosas, la primera el contador que es la parte que retorna al finalizar la función, esta variable cuenta las veces en que el elemento de valores[j] ha sido mayor que el elemento de valores[i]. Por otra parte, se realiza un procedimiento de suma que en sí no se imprime ni retorna, que es

5.

int i = 0; se repite 1 vez
while(i <= n) se repite n – 5 veces
printf("%d\n", i); se repite n – 5 veces
i += n / 5; se repite n – 5 veces

Complejidad:

$$n$$

$$= \frac{n}{5}$$

$$= n$$

$$= O(n)$$

6.

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.163 segundos	35	3.121 segundos
10	0.154 segundos	40	33.628 segundos
15	0.156 segundos	45	6.0638 minutos
20	0.157 segundos	50	No se puede
25	0.18 segundos	60	No se puede
30	0.422 segundos	100	No se puede

Hicimos el 45 y se demoró 6 minutos, el mayor registrado. Sin embargo, alcanzamos a probar el 50, y se demoró mucho más que 6 minutos y ni siquiera logramos registrarlo. Según los datos recibidos en la tabla, se empieza con tiempos menores de 1 segundo (curioso que el primero fue mayor que los 3 que le siguen, el tiempo no bajaba de 0.163 segundos), hasta que se dan saltos a

3 segundos, 33 segundos y luego a 6 minutos, se estaba esperando que fuera escalando gradualmente, pero no lo hizo, mostrando lo que consume la recursión.

Posible complejidad:

$$n$$

$$nA + nB$$

$$(O\ 2^n)$$

7.

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.155 segundos	45	0.157 segundos
10	0.156 segundos	50	0.155 segundos
15	0.156 segundos	100	0.158 segundos
20	0.156 segundos	200	0.162 segundos
25	0.153 segundos	500	0.167 segundos
30	0.156 segundos	1000	0.182 segundos
35	0.162 segundos	5000	0.944 segundos
40	0.166 segundos	10000	5.213 segundos

Complejidad: n → valor de entrada

El algoritmo realizado itera n veces, por lo que tiene una complejidad lineal: $O_{(n)}$

8.

Tamaño Entrada	Tiempo Solución Estudiantes	Tiempo Solución Profesores
100	0.172 segundos	0.172 segundos
1000	0.193 segundos	0.179 segundos
5000	0.875 segundos	0.18 segundos
10000	2.946 segundos	0.185 segundos
50000	1.12 minuto	0.274 segundos
100000	4.391 minutos	0.395 segundos
200000	16.47 minutos	0.711 segundos

a. A partir de la segunda entrada los tiempos empiezan diferenciarse. Los resultados de los profesores se mantenían en menos de 1 segundo, mientras que el de nosotros fue aumentando gradualmente, el máximo duró 16 minutos. Primero que todo, consideramos que el dividir los

labores entre varias funciones en la solución de los profesores hizo que el algoritmo sea menos pesado, además de ser menos complejo a simple vista.

b. Complejidad de la solución de los estudiantes:

Este itera desde todos los números hasta el número actual para saber si es primo

$$n$$

$$n^2$$

$$O(n^2)$$

Complejidad de la solución de los profesores:

Esta solución se desarrolla similar al nuestro a nivel de complejidad algorítmica.

$$n$$

$$n^2$$

$$O(n^2)$$