Tarea 2

Sabrina Sagasti

Código Estudiante #8965083

1.

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; #

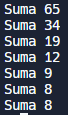
}

}

T(n) =

T(n) =

Al ejecutar el algoritmo1(8) se obtiene:



En cada línea se obtiene la suma del valor de i; siendo i inicialmente n\*n y dividiéndose entre 2 enteramente en cada iteración hasta que llegue a 0, más la cantidad de iteraciones realizadas en el instante.

2.

int algoritmo2(int n){

int res = 1, i, j; # 1

for(i = 1; i <= 2 \* n; i += 4) #

for(j = 1; j \* j <= n; j++) #

res += 2; #

return res;

}

T(n) =

T(n) =

Al ejecutar el algoritmo2(8) se obtiene:



Este valor se obtiene al sumar veces 2 a una variable que inicia en 1.

3.

void algoritmo3(int n){

int i, j, k; # 1

for(i = n; i > 1; i--) #

for(j = 1; j <= n; j++) #

for(k = 1; k <= i; k++) #

printf("Vida cruel!!\n"); #

}

T(n) =

T(n) =

T(n) =

T(n) =

T(n) =

4.

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

int suma = 0, contador = 0; # 2

int i, j, h, flag; # 4

for(i = 0; i < n; i++){ # n + 1

j = i + 1; #n

flag = 0; # n

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; # 1

}

T(n) = O()

Esto se debe a que en el peor de los casos cada ciclo se ejecutaría n veces, al ser ciclos anidados terminaría multiplicando la complejidad de cada uno, terminando en el ciclo más interno n\*n\*n

5.

void algoritmo5(int n){

int i = 0; # 1

while(i <= n){ #

printf("%d\n", i); #

i += n / 5; #

}

}

T(n) =

T(n) =

}

T(n) =

6.

def algoritmo6(n):

if n <= 1: # 1

return n #1

else: #1

return(algoritmo6 (n-1) + algoritmo6 (n-2)) #

T(n) = = O()

Esto se debe a que se llama a sí misma 2 veces para cada índice al que se le quiera hacer análisis de su número Fibonacci, a excepción de los casos bases 0 y 1.

Al completar la tabla tenemos que:

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaño de entrada | Tiempo | Tamaño de entrada | Tiempo |
| 5 | 0.000008 seg | 35 | 15.7 seg |
| 10 | 0.00006 seg | 40 | No termina |
| 15 | 0.0004 seg | 45 | No termina |
| 20 | 0.004 seg | 50 | No termina |
| 25 | 0.1 seg | 60 | No termina |
| 30 | 1.2 seg | 100 | No termina |

El valor más alto obtenido es de 15.7 segundos para una entrada de 35, esto ocurre al ser un algoritmo con un crecimiento de , lo que significa que para el tamaño de 40 el tiempo que necesitará será de segundos, resultando así que tomará 8 minutos en resolverse

7.

def recur\_fibo(n):

sol = 0 # 1

if n == 1: # 1

sol = 1 # 1

elif n == 0: # 1

sol = 0 # 1

else: # 1

a = 1 # 1

b = 0 # 1

sol = 0 # 1

for i in range(2,n+1): # (n-1) + 1

sol = a # (n-1)

a = a + b # (n-1)

b = sol # (n-1)

sol = a # 1

return sol # 1

T(n) = = O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaño de entrada | Tiempo | Tamaño de entrada | Tiempo |
| 5 | 0.00007 seg | 35 | 0.0001 seg |
| 10 | 0.00003 seg | 40 | 0.0001 seg |
| 15 | 0.00004 seg | 45 | 0.0002 seg |
| 20 | 0.0003 seg | 50 | 0.0002 seg |
| 25 | 0.003 seg | 60 | 0.0001 seg |
| 30 | 0.0005 seg | 100 | 0.0001 seg |

8.

|  |  |  |
| --- | --- | --- |
| Tamaño de entrada | Tiempo solución propia | Tiempo solución profesores |
| 100 | 0.0011 seg | 0.0003 seg |
| 1000 | 0.0123 seg | 0.0056 seg |
| 5000 | 0.4210 seg | 0.0249 seg |
| 10000 | 1.4939 seg | 0.0704 seg |
| 50000 | 28.0052 seg | 1.1867 seg |
| 100000 | 104.2676 seg | 2.7837 seg |
| 200000 | 385.5808 seg | 6.1671 seg |

La diferencia de tiempos en las soluciones se debe a una mejor implementación de la solución e el algoritmo de los profesores, estando más optimizado y teniendo un orden asintótico más pequeño que la solución creada por mi para la tarea 1.

Orden asintotico profesor:

def esPrimo(n):

if n < 2: #1

ans = False #1

else: #1

i, ans = 2, True #2

while i \* i <= n and ans: #

if n % i == 0: #

ans = False #

i += 1 #

return ans #1

T(n) =

T(n) =

Orden asintotico mio:

listaPrimos = [] #1

listaPrimos2 = [] #1

for i in range(2,limite) #

if i == 2: #

listaPrimos.append(i) #

else: #

esPrimo = True #

for j in listaPrimos: #

if i % j == 0: #

esPrimo = False #

if esPrimo == True: #

listaPrimos.append(i) #

T(n) =

T(n) =

Al final de hacer el análisis asintótico podemos notar que el algoritmo desarrollado por mi es de mientras que el de los profesores , y sabiendo que crece más rápido que podemos ver que el algoritmo de los profesores consume menos tiempo para cualquier n.