Trabajo. Análisis de complejidad

**Algoritmo 1:**

sum = 0;

for(i=0;i<sqrt(n)/2;i++)

sum++;

for(j=0 ;j<sqrt(n)/4;j++)

sum++;

for(k=0;k<8+j;k++)

sum++;

***Análisis de complejidad:***

1. Operación Básica: La operación básica es la suma ya que dicho calculo se repite en todos los bucles.
2. Número de Operaciones

* Primer Bucle: sqrt(n)/2
* Segundo Bucle: sqrt(n)/4
* Tercer Bucle: 8+j => 8+ sqrt(n)/4

1. Orden de Crecimiento

El algoritmo aumenta proporcionalmente a la raíz cuadrada de n, esto quiere decir que el tiempo de ejecución aumenta proporcional a la raíz cuadrada de n.

O(sqrt(n))

1. Costo de ejecución:

T(n)=Sqrt(n)/2 + sqrt(n)/4 + (8+sqrt(n)/4)

**Algoritmo 2:**

sum = 0;

for(i=0;i<sqrt(n)/2;i++)

for(j=i;j<8+i;j++)

for(k=j;k<8+j;k++)

sum++;

***Análisis de complejidad:***

1. Operación Básica: La operación básica es la suma ya que es la operación con la que finalizan los bucles.
2. Número de Operaciones

* Primer Bucle: sqrt(n)/2
* Segundo Bucle: 8+i => 8+sqrt(n)/2
* Tercer Bucle: 8+j => 8 + (8+sqrt(n)/2)

1. Orden de Crecimiento

El algoritmo aumenta proporcionalmente a la raíz cuadrada de n.

O(sqrt(n))

1. Costo de ejecución:

T(n)=Sqrt(n)/2 + (8+sqrt(n)/2) + (8 + (8+sqrt(n)/2))

**Algoritmo 3:**

sum = 0;

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

for(j=1;j<i\*i;j++)

for(k=1;k<j;k++)

if (j % i == 1)

sum++;

***Análisis de complejidad:***

1. Operación Básica: La función fundamental del algoritmo es la cantidad de veces que se cumple la condición en el rango de iteraciones de los tres bucles.

j % i == 1

1. Número de Operaciones

* Primer Bucle: 2\*n
* Segundo Bucle: i \* i = (2\*n) ^2 - 1
* Tercer Bucle: 2\*n \* (1^2 - 1) \* 1 + (2^2 - 1) \* 2 + (3^2 - 1) \* 3 + ... + ((2n-1)^2 - 1) \* (2n-1)

1. Orden de Crecimiento

O(n^4)

1. Costo de ejecución:

T(n)= (4n^3 - 3n^2 - 4n + 3) / 3 => O(n^3).

**Algoritmo 4:**

int f(int n) {

if (n<=1)

return 1;

else

return n\*f(n/2);

}

***Análisis de complejidad:***

1. Operación Básica: La operación básica en este algoritmo es una multiplicación entre dos números en cada llamada recursiva de la función f.
2. Número de Operaciones

* F = n/2 en cada llamada

(log2(n) \* (log2(n) + 1)) / 2

1. Orden de Crecimiento

O(log2(n)^2)

1. Costo de ejecución:

T(n) = T(n/2) + O(1) = O(log n/2) + O(1) = O(log n)