MAESTRÍA : Ingeniería de Sistemas e Informática

MENCIÓN : Ingeniería de Software

CICLO : I

CURSO : Ambientes de Desarrollo de Software

PROFESOR : Efraín R. Bautista Ubillús

ALUMNO : Gustavo Alfredo Chimoy García

ASUNTO : Laboratorio – Sesión 2

1. ***EXPLICAR PORQUE EL ALGORITMO DE LA BÚSQUEDA BINARIA TIENE COMPLEJIDAD LOGARÍTMICA***

El algoritmo de la búsqueda binaria es un algoritmo de búsqueda de vectores ordenados que disminuye la complejidad de la búsqueda.

Para calcular dicha complejidad, inicialmente el número de elementos por analizar es ***n***. Tras la primera división, el número será a lo mucho ***n/2*** (ya que queda la mitad de elementos); tras la segunda división, el número será a lo mucho ***n/4***; y así sucesivamente. Para la división de un número ***i***, el número de elementos a analizar se puede expresar así:

**[n/{2^i}]**

En el peor de los casos será cuando el elemento a buscar no se encuentre en el vector; eso significaría que tras dividir todos los elementos por analizar se queda con un número menor a 1. Por lo tanto el número máximo de llamadas a realizar es el menor número ***m*** tal que:

**[n/{2^m} < 1]**

Transformando esta fórmula a un logaritmo base 2, se tiene que:

**[n < 2^m]**

Y que:

**[log n < m]**

Eso significa que el numero ***m*** depende, no del tamaño ***n*** del vector, sino del logaritmo de dicho ***n***.

Por lo tanto el algoritmo de búsqueda binaria tiene una complejidad de orden logarítmico **(O(log n))**.

1. ***EXPLICAR PORQUE EL ALGORITMO DE LA SERIE DE FIBBONACCI TIENE COMPLEJIDAD EXPONENCIAL***

La serie de Fibonacci se expresa así:

1, 1, 2, 3, 5, 8,…

Una forma simple de generar el elemento n de esta serie es:

n = 3;

e1 = 1;

e2 = 1;

for(int i = 1; i=n; i++){

e(i) = e(i-1) + e(i-2);

}

Cuanto más grande es n, más operaciones se hacen y se incrementan de forma lineal.

En cambio al usar forma recursiva:

function Fibonacci(n){

if(n == 1 || n == 2)

return 1;

else

return Fibonacci(n) + Fibonacci(n-1);

}

Eso significa que para calcular un elemento se llama a la misma función 2 veces y cada una de dichas funciones se llama a si misma otras 2 veces, etc. Es decir crece exponencialmente. Demostrando sería así:

Paso 1 : 2 ejecuciones

Paso 2 : 2 \* (2 ejecuciones)

Paso 3 : 2 \* (2 \* (2 ejecuciones))

Paso k : (2^k) ejecuciones