CURSO : **AMBIENTES DE DESARROLLO DE SOFTWARE**

ALUMNO : Ing. Jhoss Adelfo Bryan Magallanes Morón.

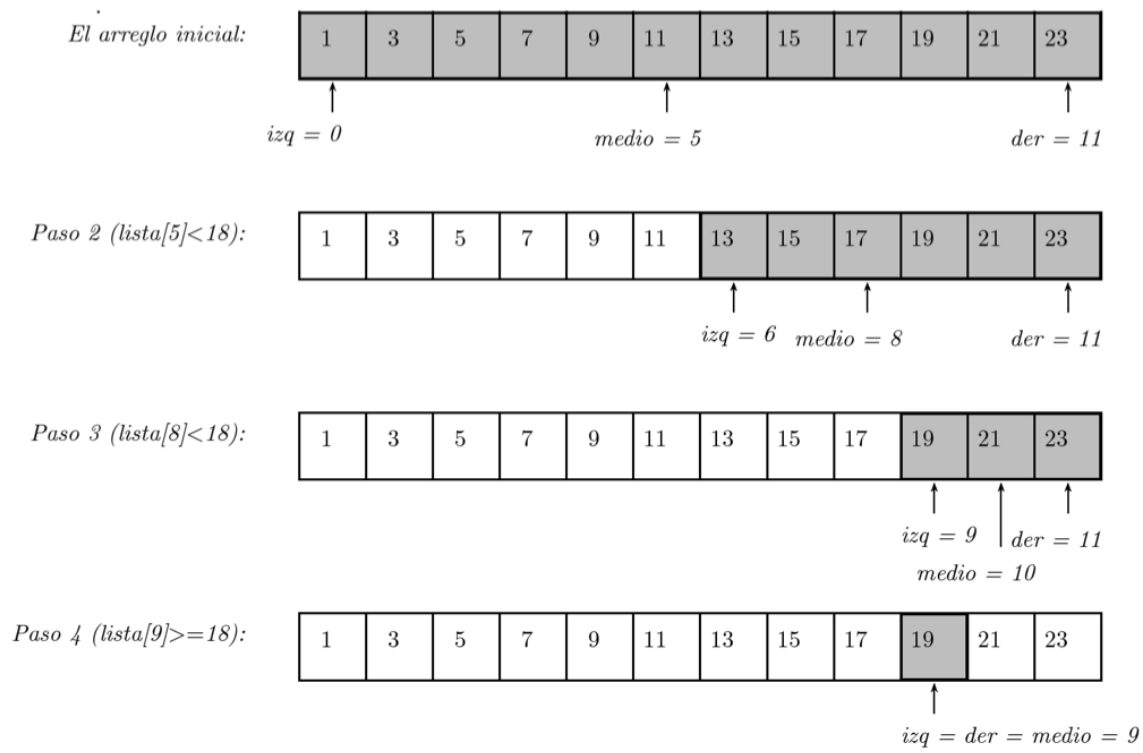
DOCENTE : Mag. Ing. Efraín Ricardo Bautista Ubillús

INVESTIGAR

1. **¿Por qué la búsqueda binaria es de COMPLEJIDAD LOGARÍTMICA?**

   La búsqueda binaria tiene una complejidad logarítmica debido a cuando el código se ejecuta va reduciendo una lista múltiple a la mitad por cada iteración que realiza; que a comparación de otros algoritmos que hacen la comparación con todos los elementos de lista en el peor caso; la búsqueda binaria lo que hace al momento de dividir la lista en dos, también reduce las comparaciones a realizar, optimizando de esa manera el tiempo y el número de iteraciones o corridas como comúnmente se le conoce.

   Es de complejidad logarítmica debido a que el espacio requerido por el algoritmo es **el mismo para cualquier cantidad de elementos en el arreglo.**



Seudocódigo:

Datos de entrada:

vec: vector en el que se desea buscar el dato

tam: tamaño del vector. Los subíndices válidos van desde 0 hasta tam-1 inclusive.

dato: elemento que se quiere buscar.

Variables

centro: subíndice central del intervalo

inf: límite inferior del intervalo

sup: límite superior del intervalo

inf = 0 ← O(1)

sup = tam-1

Mientras inf <= sup:

centro = ((sup - inf) / 2) + inf // División entera: se trunca la fracción

Si vec[centro] == dato devolver verdadero y/o pos, de lo contrario:

Si dato < vec[centro] entonces:

sup = centro - 1

En caso contrario: ← O(n/2)

inf = centro + 1

Fin (Mientras)

Devolver Falso

Comprobación:

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

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

= 1 + Log(n) = Log (n)

2. **¿Por qué el algoritmo de Fibonacci de doble recursividad tiene COMPLEJIDAD EXPONENCIAL?**

Porque en la recursividad utilizada repite el mismo proceso n veces por cada operación realizada, que conforme se ejecuta añade más complejidad al cálculo, y más iteraciones por el número de veces que corre el algoritmo.

```
static void Main(string[] args)
        {
            var stopWatch = new Stopwatch();
            stopWatch.Start();
            Exponential exponential = new Exponential();
```

```
ExponentialExampleType exponentialExampleType = ExponentialExampleType.Fibonacci;
            switch (exponentialExampleType)
            {
                case ExponentialExampleType.Fibonacci:                    O(n)
                    int n = 40; //8 40 80
                    for (int i = 1; i <= n; i++)
                    {
                        long fibonacci = exponential.Fibonacci(i);
                        Console.WriteLine("fibonacci {0} = {1}", i, fibonacci);
                    }
                    break;
            }
            Console.WriteLine("Time elapsed: {0:0.00} seconds", Math.Round(stop-
Watch.ElapsedMilliseconds / 1000.0, 2));
            Console.ReadLine();
        }

        /// Complexity: O(2^N)
        internal int Fibonacci(long n)                                    O(n)
        {
            if (n < 0)
            {
                throw new Exception("n can not be less than zero");
            }
            if (n <= 2)
            {
                return 1;
            }                                                             O(2ⁿ)
            return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
    }
    }
```

Demostración:

= O(n)xO(n) + **O(2ⁿ)**

= **O(2ⁿ)**