UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

Ingeniería en Computación

Algoritmos y Estructura de Datos

Práctica 9:

Análisis de Algoritmos Empírico en búsquedas recursivas

Catedrático:

Alumno: Reyes Udasco Rachelle Nerie

Matrícula: 1244220

Fecha de asignación: 23 de noviembre del 2018

Índice

Práctica No. 9

Análisis de Algoritmos Empírico en búsquedas recursivas.

1. Competencia

Clasificar los algoritmos de búsqueda según su desempeño en escenarios de prueba con distintos parámetros, para ser la selección del método más adecuado para la solución de problemas de manejo de información.

2. Marco Teórico

2.1 Búsqueda Binaria

Este método requiere que el arreglo este previamente ordenado. La búsqueda binaria funciona de la siguiente manera:

- Calcular el centro de la lista, con la fórmula (izquierdo+derecho)/2. Izquierdo y derecho son las posiciones del elemento menor y mayor del vector.
- Encontrar el elemento central del arreglo, la llave se compara con el centro si es igual aquí termina la búsqueda.
- Si no es igual determinar si la llave se encuentra en el lado izquierdo o derecho de la lista.
- Redefinir el inicio o el final según donde se haya ubicado la llave. Si la llave es mayor que el centro entonces izquierdo=centro+1. Si la llave es menor que el centro entonces derecho=derecho-1
- Repetir desde el primer paso hasta encontrar el dato o hasta que ya no sea posible dividir más.
- Si la llave no fue encontrada regresar -1.

2.2 Búsqueda Interpolada

Si los datos están distribuidos de forma uniforme, este método puede ser más eficiente que la búsqueda binaria. Básicamente el algoritmo es el mismo que el de la búsqueda binaria. La diferencia radica en que en la búsqueda interpolada no se busca el dato en el centro del arreglo, sino que se calcula su posición aproximada con la siguiente fórmula:

Búsqueda por interpolación que usa el valor de x, y los valores a los extremos $(i < j)$ del subarreglo actual, para *interpolar* la próxima posicion $g = i + \frac{(j-i)(x-A[i])}{A[j]-A[i]}$ a comparar con x.

3. Desarrollo

Implemente la búsqueda binaria e interpolada para tamaño N,

Utilizando inicialmente un arreglo tipo int de 20 posiciones elabore una tabla con los siguientes datos:

a) El tiempo para el peor de los casos
b) El tiempo para el mejor de los casos
c) El tiempo para cualquier otro caso
d) La cantidad de iteraciones realizadas en ambas búsquedas para cada caso
e) ¿Qué pasa al cambiar el tipo de datos del vector de int a float?
f) ¿Qué pasa al aumentar el tamaño de N?
g) ¿Qué pasa si el elemento NO está en el arreglo
   Determine nuevamente los incisos a-g pero con la búsqueda interpolada.
h) ¿Se puede utilizar la búsqueda binaria para cadenas? ¿Sí o no, por qué?
i) ¿Se puede utilizar la búsqueda interpolada para cadenas? ¿Sí o no, por qué?

- No se pide que capture las datos, inicialice los arreglos con los valores necesarios.
- Las funciones deben presentar en todo momento en pantalla los datos sobre los que se está realizando la búsqueda.
- Lenguaje libre.
- Elabore un reporte que inclúyelos resultados obtenidos y las conclusiones.

4. Resultados

4.1 Implementación en C

4.1.1 Búsqueda binaria recursiva implementada en una función en C

```
//Funcion de la busqueda binaria recursiva
int busquedaBinaria(int vector[], int llave, int izq, int der,int
*iteraciones) {
    int medio;

    medio = (izq+der)/2;
    *iteraciones+=1;

    //Impresion de variables utilizados y el vector a comparar en la
iteracion
    printf("\nIndices => Izquierda: %d, Medio: %d, Derecha: %d, Numero
comparado con llave: %d",izq,medio,der,vector[medio]);
    printf("\nVector => ");
    imprimirVector(vector,izq,der);

    if(izq>der) {
        printf("\nIteraciones: %d",*iteraciones);  //Elemento no
encontrado
        return -1;
    }else
        if (vector[medio]>llave)
        return busquedaBinaria(vector,llave,izq,medio-1,iteraciones);
//Elemento ubicado en la parte izquierda
    else
        if (vector[medio]<llave)
        return busquedaBinaria(vector,llave,medio+1,der,iteraciones);
//Elemento ubicado en la parte derecha
    else{
        printf("\nIteraciones: %d",*iteraciones);  //Elemento encontrado
        return medio;
    }
    return 0;
}
```

4.1.2 Búsqueda interpolada recursiva implementada en una función en C

```
//Funcion de la busqueda interpolada recursiva
int busquedaInterpolada(int vector[], int llave, int i, int j,int
*iteraciones){

   int pos = i + (((int)(j-i) / (vector[j]-vector[i]))*(llave -
vector[i]));
    *iteraciones+=1;

    //Impresion de variables utilizados y el vector a comparar en la
iteracion
    printf("\nIndices => Izquierda: %d, Pos: %d, Derecha: %d, Numero
comparado con llave: %d",i,pos,j,vector[pos]);
    printf("\nVector => ");
    imprimirVector(vector,i,j);
```

```
    if (i <= j && llave >= vector[i] && llave <= vector[j]) {
        if (vector[pos] < llave)
            return busquedaInterpolada(vector, llave, pos + 1,
j,iteraciones) ; //Elemento ubicado en la parte derecha
        else
            if (vector[pos] > llave)
            return busquedaInterpolada(vector,llave,i, pos - 1,
iteraciones) ;  //Elemento ubicado en la parte izquierda
        else {
             printf("\nIteraciones: %d",*iteraciones);
             return pos;  //Elemento encontrado
        }
    }
    printf("\nIteraciones: %d",*iteraciones);
    return -1; //Elemento no encontrado
}
```

4.2 Tabla de resultados

Enseguida se encuentran las tablas de los resultados obtenidos al realizar la búsqueda binaria e interpolada para diferentes casos.

Tabla 1. Resultados de la búsqueda binaria

Búsqueda Binaria	Peor caso		Mejor caso		Caso promedio		Caso no encontrado	
	Tiempo	Iteraciones	Tiempo	Iteraciones	Tiempo	Iteraciones	Tiempo	Iteraciones
Vector Int	22 ms	5	4 ms	1	17 ms	4	21 ms	5
Vector Float	22 ms	5	7 ms	1	17 ms	4	14 ms	5
Mayor número de elementos	24 ms	5	8 ms	1	20 ms	5	23 ms	6

Tabla 2. Resultados de la búsqueda interpolada

Búsqueda Interpolada	Peor caso		Mejor caso		Caso promedio		Caso no encontrado	
	Tiempo	Iteraciones	Tiempo	Iteraciones	Tiempo	Iteraciones	Tiempo	Iteraciones
Vector Int	11 ms	2	3 ms	1	6 ms	1	4 ms	1
Vector Float	12 ms	2	4 ms	1	6 ms	1	4 ms	1
Mayor número de elementos	22 ms	5	7 ms	1	20 ms	4	6 ms	1

4.3 Análisis de resultados

- Tiempo para el peor, mejor y cualquier caso
  El tiempo para los diferentes casos en la búsqueda binaria es mucho mayor que el de los de la búsqueda interpolar, esto demuestra el hecho de que es la mejora de dicho método.

- Cantidad de iteraciones realizadas en ambas búsquedas para cada caso
  Para la búsqueda binaria, la cantidad de iteraciones es parecida para el peor de los casos, el caso promedio y en el caso del elemento no encontrado. Sin embargo para el mejor de los casos solamente se realiza una iteración debido a que el elemento buscado se encuentra en el medio, es decir en el primer elemento que se va a comparar. Por otro lado, en la búsqueda de interpolación, la cantidad de iteraciones es notablemente menor que las de la búsqueda binaria.

- ¿Qué pasa al cambiar el tipo de datos del vector de int a float?
  Para algunos de los casos, en ambos tipos de búsqueda, el tiempo aumenta aproximadamente un milisegundo.

- ¿Qué pasa al aumentar el tamaño de N?
  En ambos casos, el tiempo aumenta considerablemente como se espera al analizar la complejidad de dichas búsquedas ya que dependen de la cantidad de elementos.

- ¿Qué pasa si el elemento NO está en el arreglo?
  Dependiendo de cómo esta implementado el algoritmo de búsqueda, este puede afectar el tiempo. En caso de que se realiza una comparación si el número se encuentra dentro del rango, habrá menos iteraciones y menor cantidad de tiempo, en caso de que no, este recorrerá todo el arreglo hasta no tener elementos con quien comparar.

- ¿Se puede utilizar la búsqueda binaria para cadenas? ¿Sí o no, por qué?
  Sí se puede utilizar para las cadenas porque la comparación que realiza no involucra métodos propiamente para números.

- ¿Se puede utilizar la búsqueda interpolada para cadenas? ¿Sí o no, por qué?
  No se pueden utilizar para cadenas debido a que emplea una fórmula matemática para encontrar la posición. Y dicha fórmula consiste en restar los elementos de la lista ubicados en el índice derecho e izquierdo, una cadena no puede ser restada.

4.4 Anexo de capturas de pantalla

4.4.1 Búsqueda Binaria

- **Peor caso O(log (n))**
  El elemento buscado se encuentra en una esquina, ya sea en el índice 0 ó n-1.
- **Mejor caso Ω (1)**
  El número se encuentra en el centro del arreglo.
- **Caso Promedio θ(log (n))**
  El número buscado se encuentra cerca de la mitad.

4.4.1.1 Vector Int

a) **Peor caso**

```
PEOR CASO

Vector: 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97
Llave: 97


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34
Vector => 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97

Indices => Izquierda: 10, Medio: 14, Derecha: 19, Numero comparado con llave: 70
Vector => 42 59 63 64 70 73 75 82 95 97

Indices => Izquierda: 15, Medio: 17, Derecha: 19, Numero comparado con llave: 82
Vector => 73 75 82 95 97

Indices => Izquierda: 18, Medio: 18, Derecha: 19, Numero comparado con llave: 95
Vector => 95 97

Indices => Izquierda: 19, Medio: 19, Derecha: 19, Numero comparado con llave: 97
Vector => 97

Iteraciones: 5
Posicion: 19
Tiempo transcurrido en ms: 22.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97
Llave: 34


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34
Vector => 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97

Iteraciones: 1
Posicion: 9
Tiempo transcurrido en ms: 4.000000
```

c) **Caso Promedio**

```
CASO PROMEDIO

Vector: 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97
Llave: 15


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34
Vector => 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97

Indices => Izquierda: 0, Medio: 4, Derecha: 8, Numero comparado con llave: 12
Vector => 1 3 6 7 12 15 25 27 30

Indices => Izquierda: 5, Medio: 6, Derecha: 8, Numero comparado con llave: 25
Vector => 15 25 27 30

Indices => Izquierda: 5, Medio: 5, Derecha: 5, Numero comparado con llave: 15
Vector => 15

Iteraciones: 4
Posicion: 5
Tiempo transcurrido en ms: 17.000000
```

d) **Elemento no encontrado**

```
CASO NO ENCONTRADO

Vector: 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97
Llave: 60


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34
Vector => 1 3 6 7 12 15 25 27 30 34 42 59 63 64 70 73 75 82 95 97

Indices => Izquierda: 10, Medio: 14, Derecha: 19, Numero comparado con llave: 70
Vector => 42 59 63 64 70 73 75 82 95 97

Indices => Izquierda: 10, Medio: 11, Derecha: 13, Numero comparado con llave: 59
Vector => 42 59 63 64

Indices => Izquierda: 12, Medio: 12, Derecha: 13, Numero comparado con llave: 63
Vector => 63 64

Indices => Izquierda: 12, Medio: 11, Derecha: 11, Numero comparado con llave: 59
Vector =>

Iteraciones: 5
Posicion: -1
Tiempo transcurrido en ms: 21.000000
```

4.4.1.2 Vector Float

a) **Peor caso**

```
PEOR CASO

Vector: 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,
Llave: 97.00


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34.00
Vector => 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0

Indices => Izquierda: 10, Medio: 14, Derecha: 19, Numero comparado con llave: 70.00
Vector => 42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,

Indices => Izquierda: 15, Medio: 17, Derecha: 19, Numero comparado con llave: 82.00
Vector => 73.0,75.0,82.0,95.0,97.0,

Indices => Izquierda: 18, Medio: 18, Derecha: 19, Numero comparado con llave: 95.00
Vector => 95.0,97.0,

Indices => Izquierda: 19, Medio: 19, Derecha: 19, Numero comparado con llave: 97.00
Vector => 97.0,

Iteraciones: 5
Posicion: 19
Tiempo transcurrido en ms: 22.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,
Llave: 34.00


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34.00
Vector => 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0

Iteraciones: 1
Posicion: 9
Tiempo transcurrido en ms: 7.000000
```

c) **Caso promedio**

```
CASO PROMEDIO

Vector: 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,
Llave: 15.00


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34.00
Vector => 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,

Indices => Izquierda: 0, Medio: 4, Derecha: 8, Numero comparado con llave: 12.00
Vector => 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,

Indices => Izquierda: 5, Medio: 6, Derecha: 8, Numero comparado con llave: 25.00
Vector => 15.0,25.0,27.0,30.0,

Indices => Izquierda: 5, Medio: 5, Derecha: 5, Numero comparado con llave: 15.00
Vector => 15.0,

Iteraciones: 4
Posicion: 5
Tiempo transcurrido en ms: 17.000000
```

d) **Caso no encontrado**

```
CASO NO ENCONTRADO

Vector: 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,
Llave: 60.00


Indices => Izquierda: 0, Medio: 9, Derecha: 19, Numero comparado con llave: 34.00
Vector => 1.0,3.0,6.0,7.0,12.0,15.0,25.0,27.0,30.0,34.0,42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,

Indices => Izquierda: 10, Medio: 14, Derecha: 19, Numero comparado con llave: 70.00
Vector => 42.0,59.0,63.0,64.0,70.0,73.0,75.0,82.0,95.0,97.0,

Indices => Izquierda: 10, Medio: 11, Derecha: 13, Numero comparado con llave: 59.00
Vector => 42.0,59.0,63.0,64.0,

Indices => Izquierda: 12, Medio: 12, Derecha: 13, Numero comparado con llave: 63.00
Vector => 63.0,64.0,

Indices => Izquierda: 12, Medio: 11, Derecha: 11, Numero comparado con llave: 59.00
Vector =>

Iteraciones: 5
Posicion: -1
Tiempo transcurrido en ms: 14.000000
```

4.4.1.3 Mayor cantidad de elementos

a) **Peor caso**

```
PEOR CASO

Vector: 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101
Llave: 101


Indices => Izquierda: 0, Medio: 12, Derecha: 24, Numero comparado con llave: 55
Vector => 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101

Indices => Izquierda: 13, Medio: 18, Derecha: 24, Numero comparado con llave: 75
Vector => 59 63 64 70 73 75 82 88 93 95 97 101

Indices => Izquierda: 19, Medio: 21, Derecha: 24, Numero comparado con llave: 93
Vector => 82 88 93 95 97 101

Indices => Izquierda: 22, Medio: 23, Derecha: 24, Numero comparado con llave: 97
Vector => 95 97 101

Indices => Izquierda: 24, Medio: 24, Derecha: 24, Numero comparado con llave: 101
Vector => 101

Iteraciones: 5
Posicion: 24
Tiempo transcurrido en ms: 24.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101
Llave: 55


Indices => Izquierda: 0, Medio: 12, Derecha: 24, Numero comparado con llave: 55
Vector => 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101

Iteraciones: 1
Posicion: 12
Tiempo transcurrido en ms: 8.000000
```

c) **Caso promedio**

```
CASO PROMEDIO

Vector: 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101
Llave: 34


Indices => Izquierda: 0, Medio: 12, Derecha: 24, Numero comparado con llave: 55
Vector => 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101

Indices => Izquierda: 0, Medio: 5, Derecha: 11, Numero comparado con llave: 15
Vector => 1 3 6 7 12 15 25 27 30 34 42 48

Indices => Izquierda: 6, Medio: 8, Derecha: 11, Numero comparado con llave: 30
Vector => 25 27 30 34 42 48

Indices => Izquierda: 9, Medio: 10, Derecha: 11, Numero comparado con llave: 42
Vector => 34 42 48

Indices => Izquierda: 9, Medio: 9, Derecha: 9, Numero comparado con llave: 34
Vector => 34

Iteraciones: 5
Posicion: 9
Tiempo transcurrido en ms: 20.000000
```

d) **Elemento no encontrado**

```
CASO NO ENCONTRADO

Vector: 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101
Llave: 100


Indices => Izquierda: 0, Medio: 12, Derecha: 24, Numero comparado con llave: 55
Vector => 1 3 6 7 12 15 25 27 30 34 42 48 55 59 63 64 70 73 75 82 88 93 95 97 101

Indices => Izquierda: 13, Medio: 18, Derecha: 24, Numero comparado con llave: 75
Vector => 59 63 64 70 73 75 82 88 93 95 97 101

Indices => Izquierda: 19, Medio: 21, Derecha: 24, Numero comparado con llave: 93
Vector => 82 88 93 95 97 101

Indices => Izquierda: 22, Medio: 23, Derecha: 24, Numero comparado con llave: 97
Vector => 95 97 101

Indices => Izquierda: 24, Medio: 24, Derecha: 24, Numero comparado con llave: 101
Vector => 101

Indices => Izquierda: 24, Medio: 23, Derecha: 23, Numero comparado con llave: 97
Vector =>

Iteraciones: 6
Posicion: -1
Tiempo transcurrido en ms: 23.000000
```

4.4.2 Búsqueda Interpolada

- **Peor caso O(log(log n))**
  Ocurrirá cuando los elementos de la matriz se distribuyan de manera uniforme en su intervalo.
- **Mejor caso Ω (n)**
  Ocurrirá cuando los elementos no estén distribuidos de manera bastante uniforme en su intervalo
- **Caso Promedio θ(log(log n))**
  Ocurrirá cuando los elementos de la matriz se distribuyan de manera uniforme en su intervalo.

4.4.2.1 Vector Int

a) **Peor caso**

```
PEOR CASO

Vector: 1 5 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100
Llave: 27

Indices => Izquierda: 0, Pos: 4, Derecha: 19, Numero comparado con llave: 22
Vector => 1 5 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100

Indices => Izquierda: 5, Pos: 5, Derecha: 19, Numero comparado con llave: 27
Vector => 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100

Iteraciones: 2
Posicion: 5
Tiempo transcurrido en ms: 11.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40
Llave: 40


Indices => Izquierda: 0, Pos: 19, Derecha: 19, Numero comparado con llave: 40
Vector => 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40

Iteraciones: 1
Posicion: 19
Tiempo transcurrido en ms: 3.000000
```

c) **Caso promedio**

```
CASO PROMEDIO

Vector: 1 3 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45
Llave: 27

Indices => Izquierda: 0, Pos: 11, Derecha: 19, Numero comparado con llave: 27
Vector => 1 3 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45

Iteraciones: 1
Posicion: 11
Tiempo transcurrido en ms: 6.000000
```

e) **Elemento no encontrado**

```
CASO NO ENCONTRADO

Vector: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40
Llave: 60

Indices => Izquierda: 0, Pos: 28, Derecha: 19, Numero comparado con llave: 60
Vector => 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40

Iteraciones: 1
Posicion: -1
Tiempo transcurrido en ms: 4.000000
```

4.4.2.2 Vector Float

a) **Peor caso**

```
PEOR CASO

Vector: 1.0,5.0,12.0,17.0,22.0,27.0,34.0,39.0,42.0,47.0,50.0,59.0,67.0,78.0,81.0,86.0,90.0,94.0,98.0,100.0,
Llave: 27

Indices => Izquierda: 0, Pos: 4, Derecha: 19, Numero comparado con llave: 22.000000
Vector => 1.0,5.0,12.0,17.0,22.0,27.0,34.0,39.0,42.0,47.0,50.0,59.0,67.0,78.0,81.0,86.0,90.0,94.0,98.0,100.0

Indices => Izquierda: 5, Pos: 5, Derecha: 19, Numero comparado con llave: 27.000000
Vector => 27.0,34.0,39.0,42.0,47.0,50.0,59.0,67.0,78.0,81.0,86.0,90.0,94.0,98.0,100.0,

Iteraciones: 2
Posicion: 5
Tiempo transcurrido en ms: 12.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1.0,3.0,5.0,7.0,9.0,11.0,13.0,15.0,17.0,19.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,37.0,40.0,
Llave: 40

Indices => Izquierda: 0, Pos: 19, Derecha: 19, Numero comparado con llave: 40.000000
Vector => 1.0,3.0,5.0,7.0,9.0,11.0,13.0,15.0,17.0,19.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,37.0,40.0,

Iteraciones: 1
Posicion: 19
Tiempo transcurrido en ms: 4.000000
```

c) **Caso promedio**

```
CASO PROMEDIO

Vector: 1.0,3.0,6.0,8.0,11.0,13.0,15.0,18.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,38.0,40.0,42.0,45.0,
Llave: 27

Indices => Izquierda: 0, Pos: 11, Derecha: 19, Numero comparado con llave: 27.000000
Vector => 1.0,3.0,6.0,8.0,11.0,13.0,15.0,18.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,38.0,40.0,42.0,45.0

Iteraciones: 1
Posicion: 11
Tiempo transcurrido en ms: 6.000000
```

f) **Elemento no encontrado**

```
CASO NO ENCONTRADO

Vector: 1.0,3.0,5.0,7.0,9.0,11.0,13.0,15.0,17.0,19.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,37.0,40.0,
Llave: 60

Indices => Izquierda: 0, Pos: 28, Derecha: 19, Numero comparado con llave: 0.000000
Vector => 1.0,3.0,5.0,7.0,9.0,11.0,13.0,15.0,17.0,19.0,21.0,23.0,25.0,27.0,29.0,31.0,33.0,35.0,37.0,40.0,

Iteraciones: 1
Posicion: -1
Tiempo transcurrido en ms: 4.000000
```

4.4.2.3 Mayor cantidad de elementos

a) **Peor caso**

```
PEOR CASO

Vector: 1 5 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111
Llave: 18

Indices => Izquierda: 0, Pos: 0, Derecha: 21, Numero comparado con llave: 1
Vector => 1 5 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111

Indices => Izquierda: 1, Pos: 1, Derecha: 21, Numero comparado con llave: 5
Vector => 5 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111

Indices => Izquierda: 2, Pos: 2, Derecha: 21, Numero comparado con llave: 12
Vector => 12 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111

Indices => Izquierda: 3, Pos: 3, Derecha: 21, Numero comparado con llave: 17
Vector => 17 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111

Indices => Izquierda: 4, Pos: 4, Derecha: 21, Numero comparado con llave: 22
Vector => 22 27 34 39 42 47 50 59 67 78 81 86 90 94 98 100 104 111

Iteraciones: 5
Posicion: -1
Tiempo transcurrido en ms: 22.000000
```

b) **Mejor caso**

```
MEJOR CASO

Vector: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40 42 44
Llave: 50

Indices => Izquierda: 0, Pos: 0, Derecha: 21, Numero comparado con llave: 1
Vector => 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40 42 44

Iteraciones: 1
Posicion: -1
Tiempo transcurrido en ms: 7.000000
```

c) **Caso promedio**

```
CASO PROMEDIO

Vector: 1 3 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45 47 50
Llave: 8

Indices => Izquierda: 0, Pos: 0, Derecha: 21, Numero comparado con llave: 1
Vector => 1 3 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45 47 50

Indices => Izquierda: 1, Pos: 1, Derecha: 21, Numero comparado con llave: 3
Vector => 3 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45 47 50

Indices => Izquierda: 2, Pos: 2, Derecha: 21, Numero comparado con llave: 6
Vector => 6 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45 47 50

Indices => Izquierda: 3, Pos: 3, Derecha: 21, Numero comparado con llave: 8
Vector => 8 11 13 15 18 21 23 25 27 29 31 33 35 38 40 42 45 47 50

Iteraciones: 4
Posicion: 3
Tiempo transcurrido en ms: 20.000000
```

g) **Elemento no encontrado**

```
CASO NO ENCONTRADO

Vector: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40 42 44
Llave: 60

Indices => Izquierda: 0, Pos: 0, Derecha: 21, Numero comparado con llave: 1
Vector => 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 40 42 44

Iteraciones: 1
Posicion: -1
Tiempo transcurrido en ms: 6.000000
```

5. Conclusión

Mediante la realización de esta práctica, se demostró que la búsqueda interpolar es mucho más eficiente a comparación de la binaria con respecto al tiempo de ejecución. Sin embargo, tiene sus desventajas las cuales se encuentra en una de las características que las diferencian, la fórmula para obtener la posición en la forma interpolar. Es una mejora considerable, no obstante, tiene mayor complejidad que la fórmula para obtener el índice central de la binaria. Además de esto, restringe a que solamente se aplique para números, convirtiendo la binaria como la opción para manejar la información con caracteres.

La forma de implementación de los algoritmos de búsqueda puede afectar el tiempo de ejecución ya que depende del orden de las comparaciones o la eficiencia del código escrito por el programador. Asimismo, el tiempo puede variar por la cantidad de elementos almacenados en el arreglo, la posición del elemento buscado, el tamaño de los datos y las condiciones del entorno en donde se emplean las búsquedas.