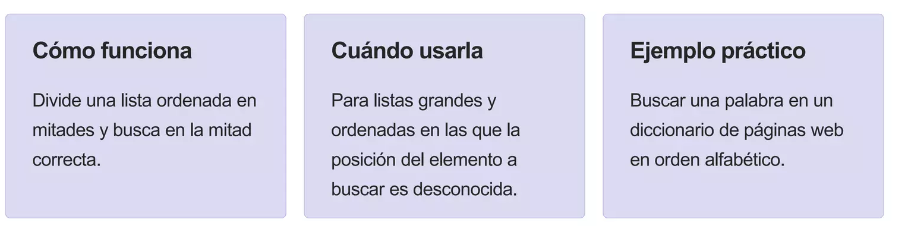
**Algoritmos de búsqueda binaria**

**1. Introducción**

La búsqueda binaria es una técnica fundamental en el campo de la informática y la ciencia de la computación. Este manual se enfoca en proporcionar una comprensión detallada de los algoritmos de búsqueda binaria, sus aplicaciones y su implementación práctica.

**2. Definición del algoritmo**

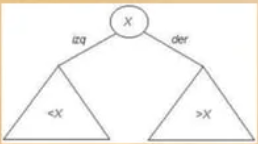
El algoritmo de búsqueda binaria es una técnica eficiente para encontrar un elemento específico en una lista ordenada de elementos. Funciona dividiendo repetidamente el conjunto de datos en mitades y descartando la mitad en la que se sabe que el elemento buscado no puede estar presente, hasta que se encuentre el elemento deseado o se determine que no está en la lista. Este enfoque aprovecha el hecho de que la lista está ordenada para reducir el número de comparaciones necesarias, logrando así una complejidad temporal de O(log n), donde n es el número de elementos en la lista. El algoritmo de búsqueda binaria es ampliamente utilizado en diversos campos, como la informática, la matemática y la ingeniería, debido a su eficiencia y su capacidad para manejar grandes conjuntos de datos de manera rápida

y efectiva.

**Imagen 1** Descripción del algoritmo

**3. Particularidades de los algoritmos de búsqueda binaria**

* La lista debe estar ordenada de acuerdo al valor de la clave, para realizar la búsqueda.
* Se reduce el vector a la mitad en comparación a <o> al elemento buscado.
* Al estar ordenada se obtiene el número total de registros.
* Es aplicable a árboles binarios y listas.
* El esfuerzo mínimo es 1, el medio es 1 log2n, el máximo log2n



**Imagen 2** Comparación de elementos

**4. Uso de los algoritmos de búsqueda binaria**

Los algoritmos de búsqueda binaria se utilizan en una amplia variedad de aplicaciones en informática y otros campos. Algunos de los usos más comunes incluyen:

1. Búsqueda en bases de datos: En sistemas de gestión de bases de datos, la búsqueda binaria se utiliza para recuperar rápidamente información almacenada en índices ordenados, lo que acelera la recuperación de datos y mejora el rendimiento de las consultas.

2. Búsqueda en árboles binarios de búsqueda: Los árboles binarios de búsqueda son estructuras de datos comunes que se utilizan para almacenar y organizar datos de manera eficiente. Los algoritmos de búsqueda binaria se utilizan para recorrer estos árboles de manera eficiente y recuperar información.

3. Búsqueda en listas ordenadas: Cuando se trabaja con listas ordenadas de elementos, la búsqueda binaria permite encontrar rápidamente un elemento específico sin tener que recorrer la lista completa, lo que resulta en un tiempo de búsqueda más rápido en comparación con otros métodos de búsqueda.

4. Algoritmos de ordenación: Algunos algoritmos de ordenación, como el algoritmo de mezcla (Merge Sort) y el algoritmo de ordenación rápida (Quick Sort), utilizan la búsqueda binaria como parte de su proceso de ordenación.

5. Búsqueda en espacios de claves: En aplicaciones como la criptografía y la seguridad informática, la búsqueda binaria se utiliza para buscar y comparar claves de manera eficiente en grandes espacios de claves.

6. Búsqueda en juegos y simulaciones: Los algoritmos de búsqueda binaria se utilizan para realizar búsquedas eficientes en estructuras de datos que representan entidades y objetos en juegos y simulaciones, lo que permite una interacción más rápida y fluida con el entorno virtual.



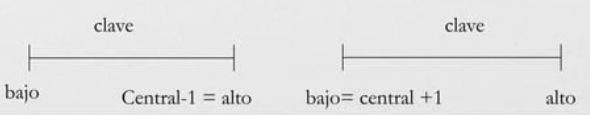
**Imagen 3** Ejemplo de una búsqueda binaria

**5. Algoritmo y codificación de la búsqueda binaria**

Suponiendo que la lista esté almacenada como un arreglo, donde los índices de la lista son bajo=0 y alto=n-1, donde n es el número de elementos del arreglo, los pasos a seguir son:

1. Calcular el índice del punto central del arreglo: **Central=(bajo+alto)/2 (División entera)**
2. Comparar el valor de este elemento central con la clave:

* Si a[central]<clave, la nueva sublista de búsqueda tiene por valores extremos de su rango bajo=central+1…alto
* Si clave <a[central], la nueva sublista de búsqueda tiene por valores extremos de su rango bajo…central-1.

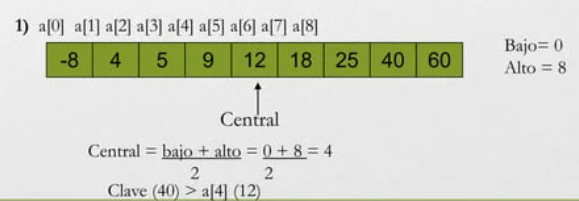


**Imagen 4** Comparación del elemento central con la clave

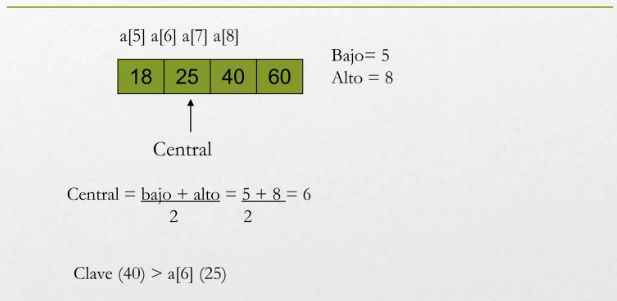
El algoritmo termina o bien porque se ha encontrado la clave, o bien porque el valor de bajo excede a alto y el algoritmo devuelve el indicador de fallo -1 (Búsqueda no encontrada)

**Ejemplo: Sea el arreglo A{-8,4,5,9,12,18,25,40,60}, buscar la clave 40:**

1) Cálculo del punto central

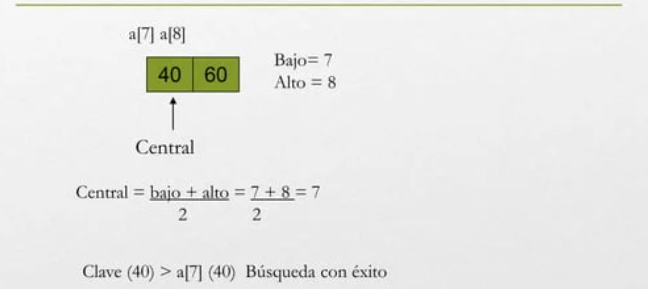
****

**Imagen 5** Cálculo del punto central y primera comparación del algoritmo

**2)** Buscar en la sublista derecha  
  


**Imagen 6** Segunda comparación del algoritmo

3) Buscar en la sublista derecha



**Imagen 7** Segunda comparación del algoritmo

Como se puede observar, este algoritmo ha requerido 3 comparaciones, frente a las 8 que hubiera necesitado la búsqueda secuencial.

**int** busquedaBin(**int** a[], **int** n, **int** clave)

{

**int** central, bajo, alto;

**int** valorCentral;

bajo = **0**;

alto = n - **1**;

**while** (bajo <= alto)

{

central = (bajo + alto)/**2**; // índice de elemento central

valorCentral = a[central]; // valor del índice central

**if** (clave == valorCentral)

**return** central; // encontrado, devuelve posición

**else** **if** (clave < valorCentral)

alto = central - **1**; // ir a sublista inferior

**else**

bajo = central + **1**; // ir a sublista superior

}

**return** -**1**; //elemento no encontrado

}

**6. Conclusiones**

En conclusión, los algoritmos de búsqueda binaria ofrecen una notable mejora en términos de eficiencia y rendimiento en comparación con los algoritmos de búsqueda secuencial. Mientras que la búsqueda secuencial implica recorrer cada elemento de una lista uno por uno, lo que resulta en un tiempo de búsqueda proporcional al tamaño de la lista (O(n)), la búsqueda binaria aprovecha la estructura ordenada de los datos para reducir significativamente el número de comparaciones necesarias.

**7. Referencias**

[1] Estructura de datos en C++, Luis Joyanes e Ignacio Zahonero.

[2] Data Structures and Algorithms in Java, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser.