

# Búsqueda Binaria

# Concepto

La búsqueda binaria es un algoritmo eficiente para encontrar un elemento específico en un arreglo **ordenado**. A diferencia de la búsqueda lineal (que examina cada elemento secuencialmente), la búsqueda binaria utiliza el principio de "dividir y conquistar":

1. Compara el elemento buscado con el elemento central del arreglo
2. Si son iguales, la búsqueda termina con éxito
3. Si el elemento buscado es menor, continúa la búsqueda en la mitad izquierda
4. Si el elemento buscado es mayor, continúa la búsqueda en la mitad derecha

Este proceso se repite, reduciendo a la mitad el espacio de búsqueda en cada iteración, hasta encontrar el elemento o determinar que no existe.



# Algoritmo de Búsqueda Binaria en Pseudocódigo

función búsquedaBinaria(A: arreglo ordenado, elemento: valor a buscar)

    izquierda = 0

    derecha = longitud(A) - 1

    mientras izquierda <= derecha hacer

        medio = (izquierda + derecha) / 2

        si A[medio] == elemento entonces

            retornar medio // Elemento encontrado, retornar su índice

        sino si A[medio] < elemento entonces

            izquierda = medio + 1 // Buscar en la mitad derecha

        sino

            derecha = medio - 1 // Buscar en la mitad izquierda

        fin si

    fin mientras

    retornar -1 // Elemento no encontrado

fin función



# Complejidad de la Búsqueda Binaria

- **Complejidad temporal:**

- Mejor caso:  $O(1)$  - cuando el elemento central es el buscado
- Caso promedio:  $O(\log n)$
- Peor caso:  $O(\log n)$  - cuando el elemento no está presente o es el último en ser encontrado

- **Complejidad espacial:**

- Iterativa:  $O(1)$
- Recursiva:  $O(\log n)$  debido a la pila de llamadas recursivas

La búsqueda binaria es exponencialmente más eficiente que la búsqueda lineal ( $O(n)$ ) para arreglos grandes.

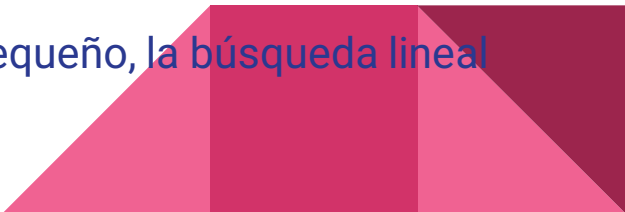


# Ventajas y Desventajas de la Búsqueda Binaria

## Ventajas:

- **Alta eficiencia:**  $O(\log n)$  es mucho mejor que  $O(n)$  para arreglos grandes
- **Predecible:** Su rendimiento es consistente y predecible
- **Versátil:** Se puede adaptar para resolver otros problemas (como encontrar puntos de inserción)

## Desventajas:

- **Requiere arreglo ordenado:** Solo funciona en colecciones ordenadas
  - **Acceso aleatorio:** Requiere que la estructura de datos permita acceso aleatorio (como arreglos)
  - **No siempre óptima para arreglos pequeños:** Para  $n$  muy pequeño, la búsqueda lineal puede ser más rápida debido a su simplicidad
- 

# Consideraciones

## Error común: desbordamiento en el cálculo del punto medio


- En lugar de  $\text{medio} = (\text{izquierda} + \text{derecha}) / 2$
- Es mejor usar:  $\text{medio} = \text{izquierda} + (\text{derecha} - \text{izquierda}) / 2$

En programación, los enteros tienen un rango limitado. Por ejemplo, en C:

- Un int típico de 32 bits puede representar valores desde -2,147,483,648 hasta 2,147,483,647
- Si intentamos almacenar un número mayor que el máximo permitido, ocurre un "desbordamiento"

## El escenario problemático

Imagina que estamos buscando en un arreglo muy grande:

- Si izquierda es 1,073,741,824 (cerca de  $2^{30}$ )
  - Y derecha es 2,147,483,646 (cerca del máximo para un int de 32 bits)
- 

# Visualización

Arreglo: [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

Buscar: 23

Iteración 1: izquierda=0, derecha=9, medio=4

$\text{arr}[\text{medio}] = 16 < 23$ , por lo que  $\text{izquierda} = \text{medio} + 1 = 5$

Iteración 2: izquierda=5, derecha=9, medio=7

$\text{arr}[\text{medio}] = 56 > 23$ , por lo que  $\text{derecha} = \text{medio} - 1 = 6$

Iteración 3: izquierda=5, derecha=6, medio=5

$\text{arr}[\text{medio}] = 23 == 23$ , ¡elemento encontrado!

Este ejemplo muestra cómo el algoritmo encontró el elemento en solo 3 comparaciones, mientras que una búsqueda lineal requeriría 6 comparaciones.

