🔢 Búsqueda Binaria

💡 Introducción

La **búsqueda binaria** es un algoritmo clásico utilizado para encontrar un elemento en una lista **ordenada**. Su historia se remonta a los primeros días de la informática y las matemáticas discretas, cuando se necesitaban métodos eficientes para buscar datos sin revisar cada elemento.

Idea principal:

> Dividir el espacio de búsqueda por la mitad en cada paso, reduciendo significativamente el número de comparaciones necesarias.

🌍 Aplicaciones del **Mundo Real de la Búsqueda Binaria**

- **Bases de datos:** Buscar registros en campos ordenados (por ID, fecha, etc.).
- **Ciencia de datos:** Optimización de hiperparámetros, series temporales, análisis de grandes datasets.
- **Computación gráfica:** Ray tracing y detección de intersecciones de manera eficiente.
- **Estructuras complejas:** Árboles binarios de búsqueda (BST), árboles AVL y otros algoritmos de búsqueda avanzados.

📚 Teoría

Veamos algunas características de este algoritmo:

- Funciona únicamente en **listas ordenadas**.
- Divide el rango de búsqueda en **dos partes** en cada iteración.
- Comparación central:
  - Si el elemento medio es igual al buscado → **encontrado**.
  - Si es menor → buscar en la **mitad derecha**.
  - Si es mayor → buscar en la **mitad izquierda**.
- **Complejidad:** `O(log n)`

💻 Algoritmo por Fuerza Bruta (Búsqueda Secuencial) `O(n)`

La búsqueda secuencial consiste en recorrer todos los elementos del arreglo hasta encontrar el número deseado.
Es simple pero **menos eficiente** en listas grandes.

```
def busqueda_fuerza_bruta(lista, objetivo):
    for i in range(len(lista)):
        if lista[i] == objetivo:
            return i  # Número encontrado
    return -1  # No encontrado
```

```
# Ejemplo
lista = [1, 3, 5, 7, 9, 11, 13]
print(busqueda_fuerza_bruta(lista, 7))  # 3
print(busqueda_fuerza_bruta(lista, 8))  # -1
```

Algoritmo Optimizado: Búsqueda Binaria `O(log n)`

Idea Clave: Comparar el elemento del medio con el objetivo.

1. Si es igual → encontrado.
2. Si es menor → buscar en la mitad derecha.
3. Si es mayor → buscar en la mitad izquierda.
4. Repetir hasta encontrar el elemento o agotar el rango.

```
def busqueda_binaria(lista, objetivo):
    izquierda, derecha = 0, len(lista) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista[medio] == objetivo:
            return medio  # Número encontrado
        elif lista[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1  # No encontrado

# Ejemplo
lista = [1, 3, 5, 7, 9, 11, 13]
print(busqueda_binaria(lista, 7))  # Salida: 3
print(busqueda_binaria(lista, 8))  # Salida: -1
```

Comparación de Algoritmos

Algoritmo	Complejidad	Comentario
Fuerza bruta	O(n)	Recorre todos los elementos
Búsqueda binaria	O(log n)	Divide el rango a la mitad en cada paso

> La búsqueda binaria es mucho más eficiente **cuando la lista está ordenada**.

Dificultades Comunes

- **Lista no ordenada:** la búsqueda binaria falla.
- **Errores por uno:** cálculos incorrectos del índice medio.

Problemas: Clase 6 – Búsqueda Binaria