1. **Objetivo del Ejercicio**

Implementar la función de **búsqueda binaria** para encontrar un número específico dentro de un **arreglo de enteros estático** (int[]).

Requisitos

1. La función debe recibir el arreglo, el tamaño del arreglo y el objetivo.
2. Debe devolver el **índice** del elemento buscado o **-1** si no se encuentra.

---

2. La **búsqueda binaria con *strings*** (cadenas de texto) sigue exactamente el mismo principio, ya que el único requisito es que los elementos estén **ordenados**. En el caso de *strings*, esto significa que deben estar ordenados **alfabéticamente** (orden lexicográfico).

Ejercicio de Búsqueda Binaria con Strings

Objetivo del Ejercicio

Implementar la función de **búsqueda binaria** para encontrar una palabra específica dentro de un **arreglo de cadenas de texto estático** (const char* []).

Requisitos

1. El arreglo de cadenas debe estar **ordenado alfabéticamente**.
2. La comparación entre cadenas se realiza usando la función strcmp() de la librería <cstring>.
3. La función debe devolver el **índice** de la cadena buscada o **-1** si no se encuentra.

---

3. **Responde las siguientes preguntas:**

¿Generalmente se usa búsqueda binaria para un arreglo de strings (como se hizo en el anterior literal) ?

Si pero con una condición importante: solo funciona si la lista o arreglo de strings está previamente ordenado alfabéticamente (lexicográficamente).

La búsqueda binaria se aplica a *strings* por las mismas razones que se aplica a los números: eficiencia y velocidad. La búsqueda binaria, al descartar la mitad de los datos en cada paso, reduce drásticamente el tiempo a O(log n). Esto es crucial para diccionarios, directorios y bases de datos grandes.

¿Si quisieras encontrar una letra dentro de una palabra (n dentro de universidad), la búsqueda binaria seria el método mas eficiente o el mas simple de implementar?

La búsqueda binaria no es adecuada para este caso, por una razon**:**

Requisito de Ordenación: Para usar la búsqueda binaria, el arreglo de búsqueda debe estar ordenado. En el caso de la palabra "universidad", sus letras:

u,n,i,v,e,r,s,i,d,a,d ...no están ordenadas alfabéticamente.

Para poder usar la búsqueda binaria, primero tendríamos que convertir la palabra en un arreglo de caracteres y luego **ordenarlo**:

a, d, d, e, i, i, n, r, s, u, v

Solo después de este paso podrías aplicar la búsqueda binaria.

4. **Ejercicio Combinado: Ordenar y Buscar**

Objetivo del Ejercicio

1. Implementar la función de **Ordenación Burbuja** (ordenarBurbuja) para ordenar alfabéticamente una cadena de texto.
2. Implementar la función de **Búsqueda Binaria** (busquedaBinariaLetra) para encontrar la letra objetivo en la cadena ya ordenada.

5. 1- Se cuenta con un archivo directo llamado `ALUMNOS.DAT` con información de alumnos de una universidad. El archivo se abre para lectura y se indexa por medio de su clave primaria (`#legajo`). El índice se mantiene en memoria principal.

   Se ofrece la posibilidad al usuario de buscar alumnos por legajo. La búsqueda binaria se aplica sobre el índice y se accede en forma directa (`seek`) al archivo para recuperar el registro.

   Se asume que el archivo está ordenado de manera ascendente por su clave primaria (`#legajo`). Si éste no fuese el caso, el índice debería ordenarse antes de aplicar una búsqueda binaria sobre él.

   Aclaración: no se provee el archivo `ALUMNOS.DAT` (debido a que es binario). Deberá ser generado antes de la ejecución de este programa).

6. 4- Localice un entero en un arreglo usando una versión recursiva del algoritmo de búsqueda binaria. El número que se debe buscar (clave) se debe ingresar como argumento por línea de comandos. El arreglo de datos debe estar previamente ordenado (requisito de la búsqueda binaria).

   Comparar esta versión recursiva de la búsqueda binaria con la versión iterativa dada en el capítulo 5.

7. **Ejemplo 10.6**

Encontrar el algoritmo de búsqueda binaria para encontrar un elemento K en una lista de elementos $X_1, X_2, \ldots, X_n$ previamente clasificados en orden ascendente.

El array o vector X se supone ordenado en orden creciente si los datos son numéricos, o alfabéticamente si son caracteres. Las variables BAJO, CENTRAL, ALTO indican los límites inferior, central y superior del intervalo de búsqueda.

```
algoritmo busqueda_binaria
  //declaraciones
inicio
  //llenar (X,N)
  //ordenar (X,N)
  leer(K)
  //inicializar variables
  BAJO ← 1
  ALTO ← N
  CENTRAL ← ent ((BAJO + ALTO) / 2)
  mientras (BAJO =< ALTO) y (X[CENTRAL] <> K) hacer
    si K < X[CENTRAL] entonces
      ALTO ← CENTRAL - 1
    si_no
      BAJO ← CENTRAL + 1
    fin_si
    CENTRAL ← ent ((BAJO + ALTO) / 2)
  fin_mientras
  si K = X[CENTRAL] entonces
    escribir('Valor encontrado en', CENTRAL)
  si_no
    escribir('Valor no encontrado')
  fin_si
fin
```

8. **Ejercicio Avanzado: Tabla Hash con Lista Enlazada Manual**

Objetivo del Ejercicio

Implementar una clase TablaHash utilizando:

1. Un **arreglo estático** de punteros (Nodo*[]) para la tabla principal.
2. Una **Lista Enlazada Simple** (creada con struct Nodo) en cada posición del arreglo para manejar las colisiones por **Encadenamiento Separado**.