EJERCICIO 1: **BÚSQUEDA POR DICOTOMÍA**

Hipótesis: Este algoritmo busca un componente específico en una lista ordenada. Se basa en el algoritmo de búsqueda binaria para encontrar el componente deseado de manera eficiente.

Precondición: La lista debe estar ordenada de forma ascendente.

Poscondición: Se devuelve la posición del componente buscado en la lista, si está presente. En caso contrario, se indica que el componente no se encuentra en la lista.

Entrada:

- Longitud de la lista.
- Componentes de la lista, uno por uno, en orden ascendente.
- Componente que se desea buscar en la lista.

Salida:

- La posición del componente buscado en la lista, si está presente.
- Un mensaje indicando que el componente no se encuentra en la lista.

Efecto: El algoritmo utiliza el método de búsqueda binaria para encontrar el componente deseado de manera eficiente en la lista. Se actualizan las variables de inicio y fin de la tabla en cada iteración, dividiendo la búsqueda a la mitad cada vez. Si el componente es encontrado, se devuelve su posición. Si la lista no está ordenada o el componente no está presente, se indica apropiadamente.

EJERCICIO 2: **PALÍNDROMO**

Hipótesis: El algoritmo busca determinar si una palabra o frase es un palíndromo. Un palíndromo es una palabra, frase, número o cualquier otra secuencia de caracteres que se lee igual hacia adelante que hacia atrás.

Precondición: El usuario proporciona una palabra o frase para verificar si es un palíndromo.

Postcondición: Se muestra si la palabra o frase proporcionada por el usuario es un palíndromo o no.

Entrada:

- Una palabra o frase ingresada por el usuario.

Salida:

- Un mensaje que indica si la palabra o frase ingresada es un palíndromo o no.

Efecto:

El algoritmo de palíndromo se estructura en una clase llamada Palíndromo, que encapsula la lógica para determinar si una palabra o frase es un palíndromo. Dentro de esta clase, el método '*texto_filtrado()*' es responsable de procesar el texto de entrada. Primero, elimina los caracteres que no son alfanuméricos y luego reemplaza las letras con tildes por letras sin tildes. Posteriormente, convierte todo el texto a minúsculas para asegurar una comparación sin distinción entre mayúsculas y minúsculas. El método '*comprobar_texto()*' utiliza el texto filtrado para determinar si es un palíndromo. Para ello, compara el texto con su reverso y devuelve True si son iguales, lo que indica que el texto es un palíndromo, y False en caso contrario.

EJERCICIO 3: **ORDENAR FICHAS**

Hipótesis: El algoritmo asume que se tiene una lista de fichas de colores, donde cada ficha puede ser roja, verde, azul o de otro color.

Precondición: Se espera que el usuario proporcione una lista de fichas para ordenar.

Poscondición: El algoritmo devuelve la lista de fichas ordenada según el criterio establecido: primero las rojas, luego las verdes, seguidas de cualquier otro color y finalmente las azules.

Entrada:

- Una lista de fichas de colores.
- La longitud de la lista de las fichas

Salida:

- La lista de fichas ordenada según el criterio mencionado.

Efecto:

El algoritmo de ordenación de fichas organiza una lista de fichas según su color, clasificándolas en rojas, verdes, otros colores y azules. Se define la clase '*OrdenadorFichas*', que inicializa la lista de fichas y cuenta la cantidad de fichas de cada color. Luego, ordena las fichas colocando primero las rojas, luego las verdes, seguidas de cualquier otro color y finalmente las azules. Para ello, utiliza dos métodos: *contar_fichas()* y *ordenar_fichas()*. El método *contar_fichas()* recorre la lista de fichas, contabilizando cuántas son rojas, verdes, azules y de otros colores. Es importante mencionar que **SÓLO SE RECORRE LA LISTA UNA VEZ.** El método *contar_fichas()* construye una lista ordenada según el criterio mencionado, utilizando la cantidad de fichas de cada color contabilizada previamente.