EJERCICIO 1: ORDENACIÓN POR INSERCIÓN DICOTÓMICA

Precondición:

- La lista de entrada no debe estar vacía.
- La lista de entrada debe contener números enteros.
- Los números en la lista de entrada deben estar dentro del rango de valores que se pueden manejar.
- La lista de entrada no debe contener elementos repetidos.

Poscondición:

- La lista de salida estará ordenada en orden ascendente.
- La lista de salida tendrá los mismos elementos que la lista de entrada.

Datos de entrada:

- Lista de números enteros.

Datos de salida:

- Lista ordenada de números enteros.

Hipótesis:

- La lista de entrada puede ser vacía.
- La lista de entrada puede contener elementos repetidos.

Efecto:

- La lista de salida estará ordenada de manera ascendente.
- No se modificará la lista de entrada.

Pseudocódigo:

1. Definición de la función `insercion_dicotomica(lista)`:
    - Esta función toma una lista como entrada y ordena sus elementos utilizando el algoritmo de inserción dicotómica.
    - Comienza creando una lista llamada `lista_nueva` del mismo tamaño que la lista de entrada `lista`, inicializando todos los elementos en `None`.
    - Luego, copia el primer elemento de la lista de entrada directamente en `lista_nueva`.
2. Bucle principal para ordenar la lista:
    - Itera sobre los elementos restantes de la lista de entrada, comenzando desde el segundo elemento.
    - Para cada elemento, busca su posición de inserción en la lista ordenada `lista_nueva` utilizando el algoritmo de búsqueda binaria por dicotomía.
    - Inserta el elemento en la posición correcta en la lista ordenada, desplazando los elementos mayores hacia la derecha para hacer espacio.
3. Retorno de la lista ordenada:
    - Después de ordenar todos los elementos, devuelve la lista ordenada `lista_nueva`.

EJERCICIO 2: ORDENACIÓN TOPOLÓGICA

Precondición:

- El número de tareas (`n`) debe ser un entero positivo.
- La lista de restricciones debe contener tuplas donde el primer elemento (`tarea_ant`) y el segundo elemento (`tarea_sig`) sean números enteros dentro del rango de 1 a `n`.
- No debe haber ciclos en las restricciones; es decir, no debe haber dependencias circulares entre las tareas.

Poscondición:

- Si es posible, la función `ordenacion_topologica` devuelve una lista que representa el orden topológico de las tareas. Si no es posible, devuelve `None`.
- La lista de restricciones (`restricciones`) debe estar correctamente formada, con las restricciones ingresadas por el usuario o generadas aleatoriamente.
- El mensaje de salida indica el orden de las tareas o informa si no se puede determinar un orden debido a restricciones contradictorias.

Datos de entrada:

- El número de tareas (`n`).
- Las restricciones correspondientes(`restricciones`).

Datos de salida:

- Orden de las tareas si es posible (`lista ordenada`).
- Mensaje indicando que no se puede determinar un orden debido a restricciones contradictorias.

Hipótesis:

- Se asume que el usuario ingresa números válidos para el número de tareas, el número de restricciones y las restricciones individuales.
- Se asume que no hay errores en la generación aleatoria de restricciones.
- Se asume que no hay restricciones circulares entre las tareas.

Efecto:

- La función `ordenacion_topologica` construye un grafo dirigido a partir de las restricciones y utiliza el algoritmo de ordenación topológica para determinar el orden de las tareas.
- El programa principal solicita al usuario el número de tareas, la opción para generar restricciones aleatorias, y las restricciones si es necesario.
- El programa muestra el orden de las tareas o informa si no se puede determinar un orden debido a restricciones contradictorias.

Pseudocódigo:

1º. Definición de la función **`ordenacion_topologica(n, restricciones):`**

- La función recibe dos parámetros: el número total de tareas (`n`) y una lista de restricciones (`restricciones`).
- Crea un grafo dirigido representado como una lista de listas llamada *`grafo`*, donde cada índice representa una tarea y las listas asociadas contienen las tareas que deben ejecutarse después de la tarea correspondiente.
- Inicializa una lista llamada `grados_entrada` con ceros, donde cada índice representa una tarea y el valor asociado representa el número de tareas que deben ejecutarse antes de la tarea correspondiente.
- Inicializa una lista vacía llamada `orden_final_tareas` para almacenar el orden final de las tareas.

2º. Describir el grafo

- Itera sobre cada restricción en la lista de restricciones:
    - Añade una arista al grafo desde la tarea anterior hacia la tarea siguiente.
        - * Las aristas están formadas por tuplas en las que el primer número es la tarea que va primero, y el segundo número es la tarea que va después; sin embargo, que vaya después no significa que tenga que ir inmediatamente después que la tarea anterior
    - Incrementa el grado de entrada de la tarea siguiente.
        - El grado es el número de tareas siguientes que tiene una tarea; es decir, el número de aristas que parten de una tarea.

3º. Bucle principal

- Inicializa una cola (lista temporal que sirve para ordenar las tareas) con las tareas que tienen un grado de entrada cero.
    - Mientras la cola tenga algún número:
        - Extrae una tarea de la cola.

        - Agrega la tarea extraída al orden final de tareas.
        - Para cada tarea siguiente de la tarea extraída:
            - Decrementa el grado de entrada de la tarea siguiente.
            - Si el grado de entrada de la tarea siguiente es cero, la agrega a la cola.
- Si todas las tareas están en el orden final de tareas, devuelve el orden; de lo contrario, devuelve `None`.

EJERCICIO 3: COMPLETAR LAS ESPECIFICACIONES

Precondición:

- La lista `t` debe estar inicializada y no vacía.
- Los índices de inicio (`inicio`) y fin (`fin`) deben estar dentro del rango de índices válidos para la lista `t`.
- El índice de inicio (`inicio`) no puede ser mayor que el índice de fin (`fin`).

Poscondición:

- Si se puede explorar el segmento correctamente, la función modifica la lista `t` y coloca el elemento más grande del segmento en la posición de inicio del segmento.
- Si no se puede explorar el segmento, la lista `t` permanece sin cambios.

Datos de entrada:

- Longitud de la lista (`longitud`).
- Lista de números (`t`).
- Índice de inicio del segmento a explorar (`inicio`).
- Índice de fin del segmento a explorar (`fin`).

Datos de salida:

- Mensaje indicando si se pudo explorar el segmento correctamente o no.
- Si se exploró el segmento correctamente, muestra la lista `t` modificada.

Hipótesis:

- Se asume que el usuario ingresará índices válidos para el segmento a explorar.
- Se asume que la lista `t` contiene números enteros.
- Se asume que la opción de generar una lista aleatoria funciona correctamente y que la entrada del usuario se hace correctamente.

Efecto:

- La función `está_explorado` busca el elemento más grande en el segmento especificado de la lista `t` y lo coloca en la posición de inicio del segmento, si es posible.

- La función modifica la lista `t` si se encuentra un elemento máximo en el segmento.
- El programa principal solicita al usuario la opción de generar una lista aleatoria o ingresar una lista manualmente, luego solicita los índices de inicio y fin del segmento a explorar, y finalmente, muestra la lista original y si se pudo explorar el segmento o no.

Pseudocódigo:

1. Verifica la precondición:
    - Comprueba si la lista `t` está vacía.
    - Comprueba si los índices de inicio (`inicio`) y fin (`fin`) están dentro del rango de índices válidos para la lista `t`.
    - Comprueba si el índice de inicio (`inicio`) es mayor que el índice de fin (`fin`).
    - Si alguna de estas condiciones no se cumple, devuelve `False`.
2. Inicializa una variable `maximo` con el primer elemento del segmento (`t[inicio]`).
3. Encuentra el máximo del segmento:
    - Itera sobre los elementos del segmento desde `inicio + 1` hasta `fin`.
    - Si encuentra un elemento mayor que `maximo`, actualiza `maximo` con ese valor.
4. Verifica las condiciones del segmento:
    - Itera sobre los elementos del segmento desde `inicio` hasta `fin`.
    - Si encuentra un elemento igual a `maximo`:
        - Guarda una copia de seguridad del elemento (`mi`).
        - Desplaza los elementos del segmento una posición hacia la izquierda.
        - Coloca el elemento máximo en la posición de inicio del segmento.
        - Devuelve `True`.
5. Si no se encontró el máximo correctamente, devuelve `False`.