***# Tarea 1 - Práctica de búsqueda binaria***

1. Plantee el problema claramente. Identificar los formatos de entrada y salida.

Se me ha dado una lista de números, desorganizada, [5, 6, 9, 0, 2, 3, 4], esta lista antes estaba organizada, [0, 2, 3, 4, 5, 6, 9], lo que paso fue que el ultimo número paso a ser el primero, esto se repitió 3 veces.

Escribe una función que determine el número de veces que el ultimo número paso a ser primero, la función debe ser Binaria.

Input: lista rotada

Output: 3

2. Proponga algunos ejemplos de entradas y salidas. Trate de cubrir todos los casos extremos.

1. Una lista de tamaño 10 rotada 3 veces.

2. Una lista de tamaño 8 rotada 5 veces.

3. Una lista que no fue rotada en absoluto.

4. Una lista que se rotó solo una vez.

5. Una lista que se rotó `n-1` veces, donde `n` es el tamaño de la lista.

6. Una lista que se rotó `n` veces (¿recupera aquí la lista original?)

7. Una lista vacía.

8. Una lista que contiene un solo elemento.

Describir la solución:

        'nums': [6, 7, 8, 9, 0, 2, 3, 4, 5]

Una función que reciba la lista

Crear una variable que empieza en 0 llamada rotación

Buscar el numero del final y el número del comienzo

Si el numero del comienzo es mayor que el del final, rotación sumara 1

Y el numero del comienzo debe moverse

Solución lineal:

def count\_rotations(nums):

    begin, last = 0, len(nums) - 1

    rotacion = 0

    for i, b in enumerate(nums):

        #numero del inicio

        begin\_number = nums[i]

        #numero final

        last\_number = nums[last]

        if begin\_number > last\_number:

            rotacion += 1

    if rotacion > 0:

        print(f"la lista fue rotada {rotacion} veces")

    elif len(nums) == 0:

        print("la lista esta vacia")

    elif len(nums) == 1:

        print(f"la lista solo contiene un elemento, el cual es: {nums}")

    else:

        print(f"la lista esta ordenada")

count\_rotations(test["input"]["nums"])

según GPT este código es lineal e ineficiente:

def binary\_search(nums):

    begin, last = 0, len(nums) - 1

    mid\_index = (begin + last) // 2

    up = 0

    rotacion = 0

        #numero final

    final\_number = nums[last]

        #numero del medio

    mid\_number = nums[mid\_index]

        #numero anterior

    after\_number = nums[mid\_index - 1]

    while after\_number > mid\_number and not after\_number == final\_number:

        rotacion += 1

        up += 1

        after\_number = nums[mid\_index - up]

    print(f"la rotacion de la lista: {nums} \nes de: {rotacion - 1}")

binary\_search(test['input']["nums"])

esta es la respuesta correcta, la búsqueda binaria funciona como una plantilla de reasignación de valores, donde lo único que cambia son las condiciones del if, el while y los valores de inicio y final son los mismos.

def search\_rotations(nums):

    left, right = 0, len(nums) - 1

    while left <= right:

        mid = (left + right) // 2

        if nums[mid] > nums[mid + 1]:

            return mid + 1

        elif nums[mid] >= nums[left]:

            left = mid + 1

        else:

            right = mid - 1

    return 0