Soluciones a los problemas: Clase 6 - Búsqueda binaria

Problemas: Clase 6 - Búsqueda binaria

Número positivo, negativo o cero

```
# Lee un numero entero
n = int(input())

# Comprueba si es positivo, negativo o cero
if n > 0:
    print("Positivo")
elif n < 0:
    print("Negativo")
else:
    print("Cero")
```

Suma dividida entre dos

```
# Lee cuatro numeros enteros
a, b, c, d = list(map(int, input().split()))

# Calcula la media entera de a y b
med = (a + b) // 2

# Verifica si la media esta entre c y d
if c <= med and med <= d:
    print("Si")
else:
    print("No")
```

Buscar un número en lista ordenada con fuerza bruta

```
# Lee una lista de numeros enteros
li = list(map(int, input().split()))

# Lee un numero a buscar
n = int(input())

# Recorre la lista y busca el numero
for x in li:
    if x == n:
        print("Si")
        break
```

```
else:
    print("No")
```

Búsqueda binaria directa

```
# Lee una lista de numeros enteros (debe estar ordenada)
li = list(map(int, input().split()))

# Lee el numero a buscar
n = int(input())

# Inicializa los limites de la busqueda
ini = 0
fin = len(li) - 1

# Bucle de busqueda binaria
# En cada paso se compara el elemento central con el numero buscado
while ini <= fin:
    piv = (ini + fin) // 2  # Calcula el indice medio
    if n == li[piv]:        # Si lo encuentra, termina
        print("Si")
        break
    elif n > li[piv]:       # Si el numero es mayor, busca en la mitad derecha
        ini = piv + 1
    else:                   # Si es menor, busca en la mitad izquierda
        fin = piv - 1
else:
    # Si el bucle termina sin encontrarlo
    print("No")
```

Buscar el número más cercano

```
# Lee una lista de numeros enteros (ordenada)
li = list(map(int, input().split()))

# Lee el numero de referencia
n = int(input())

# Inicializa los limites de busqueda
ini = 0
fin = len(li) - 1

# Inicializa la mejor solucion con un valor muy grande
sol = 1e18

# Bucle de busqueda binaria para encontrar el numero mas cercano a n
while ini <= fin:
    piv = (ini + fin) // 2   # Calcula el indice medio
    num = li[piv]            # Toma el numero central
```

```
    # Actualiza la mejor solucion si este numero esta mas cerca de n
    # o si la distancia es igual pero el numero es menor
    if (abs(num - n) < abs(sol - n)) or (abs(num - n) == abs(sol - n) and num <
sol):
        sol = num

    # Ajusta los limites segun la comparacion
    if n < num:
        fin = piv - 1
    elif n > num:
        ini = piv + 1
    else:
        # Si encuentra el numero exacto, termina
        sol = num
        break

# Muestra el numero mas cercano encontrado
print(sol)
```

Contar ocurrencias usando búsqueda binaria

```
# Funcion para encontrar la primera posicion de n en la lista
def encontrar_primera(lista, n):
    ini, fin = 0, len(lista) - 1
    res = -1
    # Busqueda binaria modificada
    while ini <= fin:
        mid = (ini + fin) // 2
        if lista[mid] == n:      # Si encuentra n, guarda la posicion y sigue a la
izquierda
            res = mid
            fin = mid - 1
        elif lista[mid] < n:     # Si el valor es menor, busca a la derecha
            ini = mid + 1
        else:                    # Si el valor es mayor, busca a la izquierda
            fin = mid - 1
    return res

# Funcion para encontrar la ultima posicion de n en la lista
def encontrar_ultima(lista, n):
    ini, fin = 0, len(lista) - 1
    res = -1
    # Busqueda binaria modificada
    while ini <= fin:
        mid = (ini + fin) // 2
        if lista[mid] == n:      # Si encuentra n, guarda la posicion y sigue a la
derecha
            res = mid
            ini = mid + 1
        elif lista[mid] < n:     # Si el valor es menor, busca a la derecha
```

```
            ini = mid + 1
        else:                   # Si el valor es mayor, busca a la izquierda
            fin = mid - 1
    return res

# Lee la lista ordenada y el numero a buscar
lista = list(map(int, input().split()))
n = int(input())

# Busca la primera y ultima aparicion de n
primera = encontrar_primera(lista, n)
ultima = encontrar_ultima(lista, n)

# Si no aparece, imprime 0; si aparece, cuenta cuantas veces
if primera == -1:
    print(0)
else:
    print(ultima - primera + 1)
```

Altura máxima de corte de árboles

```
# Calcula cuanta madera se obtiene al cortar los arboles a una altura h
def madera_obtenida(alturas, h):
    return sum(max(0, a - h) for a in alturas)

# Encuentra la altura maxima de corte para obtener al menos M de madera
def altura_maxima_corte(alturas, M):
    izq, der = 0, max(alturas)  # Limites de busqueda
    resultado = 0

    # Busqueda binaria sobre la altura de corte
    while izq <= der:
        mid = (izq + der) // 2
        madera = madera_obtenida(alturas, mid)  # Madera obtenida con altura mid

        if madera >= M:       # Si se obtiene suficiente madera, probar cortar mas
alto
            resultado = mid
            izq = mid + 1
        else:                 # Si falta madera, cortar mas bajo
            der = mid - 1

    return resultado

# Lee las alturas de los arboles y la madera requerida
alturas = list(map(int, input().split()))
M = int(input())

# Imprime la altura maxima de corte
print(altura_maxima_corte(alturas, M))
```