10 - Searching & Sorting

For example:

Input	Result
5 6 5 4 3 8	3 4 5 6 8

Ex. No. : 10.1 Date:

Register No.: Name:

Merge Sort

Write a Python program to sort a list of elements using the merge sort algorithm.

```
def merge_sort(arr):

    if len(arr) <= 1:

        return arr



    # Split the array into two halves

    mid = len(arr) // 2

    left_half = arr[:mid]

    right_half = arr[mid:]



    # Recursive calls to sort each half

    left_half = merge_sort(left_half)

    right_half = merge_sort(right_half)



    # Merge the sorted halves

    sorted_arr = []

    i = j = 0

    while i < len(left_half) and j < len(right_half):
        if left_half[i] < right_half[j]:
            sorted_arr.append(left_half[i])
            i += 1
        else:
            sorted_arr.append(right_half[j])
            j += 1

    # Add remaining elements from both halves
    sorted_arr.extend(left_half[i:])
    sorted_arr.extend(right_half[j:])
    return sorted_arr

# Input
n = int(input())
arr = list(map(int, input().split()))
# Sorting
sorted_arr = merge_sort(arr)

# Output
print(*sorted_arr)
```

Input Format

The first line contains an integer,n , the size of the list a . The second line contains n, space-separated integers a[i].

Constraints

· 2<=n<=600

· 1<=a[i]<=2x10^{6}.

Output Format

You must print the following three lines of output:

1. List is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.

2. First Element: firstElement, the *first* element in the sorted list.

3. Last Element: lastElement, the *last* element in the sorted list.

Sample Input 0

3

1 2 3

Sample Output 0

List is sorted in 0 swaps.

First Element: 1

Last Element: 3

For example:

Input	Result
3 3 2 1	List is sorted in 3 swaps. First Element: 1 Last Element: 3
5 1 9 2 8 4	List is sorted in 4 swaps. First Element: 1 Last Element: 9

Ex. No. : 10.2 Date:

Register No.: Name:

Bubble Sort

Given an listof integers, sort the array in ascending order using the *Bubble Sort* algorithm above. Once sorted, print the following three lines:

1. List is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.
2. First Element: firstElement, the *first* element in the sorted list.
3. Last Element: lastElement, the *last* element in the sorted list.

For example, given a worst-case but small array to sort: a=[6,4,1]. It took 3 swaps to sort the array. Output would be

```
Array is sorted in 3 swaps.
First Element: 1
Last Element: 6
```

```
def bubble_sort(arr):

    n = len(arr)

    num_swaps = 0

    for i in range(n):

        for j in range(0, n-i-1):

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

                num_swaps += 1

    return arr, num_swaps



num_elements = int(input().strip())

array = list(map(int, input().strip().split()))
```

```
sorted_array, num_swaps = bubble_sort(array)


print(f"List is sorted in {num_swaps} swaps.")
print(f"First Element: {sorted_array[0]}")
print(f"Last Element: {sorted_array[-1]}")
```

Input Format

The first line contains a single integer n , the length of A .
The second line contains n space-separated integers,A[i].

Output Format

Print peak numbers separated by space.

Sample Input

5

8 9 10 2 6

Sample Output

10 6

For example:

Input	Result
4 12 3 6 8	12 8

Ex. No. : 10.3 Date:

Register No.: Name:

Peak Element

Given an list, find peak element in it. A peak element is an element that is greater than its neighbors.

An element a[i] is a peak element if

A[i-1] <= A[i] >=a[i+1] for middle elements. [0<i<n-1]

A[i-1] <= A[i] for last element [i=n-1]

A[i]>=A[i+1] for first element [i=0]

```
def find_peak_element(nums):

    def find_peak_util(nums, low, high):

        mid = low + (high - low) // 2



        # Check if mid is a peak element

        if (mid == 0 or nums[mid] >= nums[mid - 1]) and (mid == len(nums) - 1 or nums[mid] >=
nums[mid + 1]):

            return mid



        # If the left neighbor is greater, there must be a peak element on the left side

        if mid > 0 and nums[mid - 1] > nums[mid]:

            return find_peak_util(nums, low, mid - 1)



        # If the right neighbor is greater, there must be a peak element on the right side

        return find_peak_util(nums, mid + 1, high)
    return find_peak_util(nums, 0, len(nums) - 1)


# Get input from user
user_input = input("Enter a list of numbers separated by spaces: ")
nums = list(map(int, user_input.split()))


# Find peak element
peak_index = find_peak_element(nums)
print(f"T{peak_index} and the value is {nums[peak_index]}")
```

For example:

Input	Result
1 2 3 5 8 6	False
3 5 9 45 42 42	True

Ex. No. : 10.4 Date:

Register No.: Name:

Binary Search

Write a Python program for binary search.

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return True
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return False


sorted_list = list(map(int, input().split(',')))
target = int(input())

sorted_list.sort()
```

```
result = binary_search(sorted_list, target)

print(result)
```

Input:

1 68 79 4 90 68 1 4 5

output:

1 2

4 2

5 1

68 2

79 1

90 1

For example:

Input	Result

4 3 5 3 4 5	3 2
	4 2
	5 2

Ex. No. : 10.5 Date:

Register No.: Name:

Frequency of Elements

To find the frequency of numbers in a list and display in sorted order.

Constraints:

1<=n, arr[i]<=100

```
def count_frequencies(arr):
    frequency_dict = {}
    for num in arr:
        if num in frequency_dict:
            frequency_dict[num] += 1
        else:
            frequency_dict[num] = 1
    sorted_keys = sorted(frequency_dict.keys())
    for key in sorted_keys:
        print(key, frequency_dict[key])


# Read input from the user
input_list = list(map(int, input().split()))


# Count frequencies and display the result
count_frequencies(input_list)
```