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.

```
n = int(input())
arr = list(map(int, input().split()))

size = len(arr)

if size > 1:
    mid = size // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_size = len(left_half)
    right_size = len(right_half)

    left_sorted = []
    right_sorted = []

    if left_size > 1:
```

```
    left_mid = left_size // 2
    left_half[:left_mid] = sorted(left_half[:left_mid])
    left_half[left_mid:] = sorted(left_half[left_mid:])
    l, r, k = 0, 0, 0
    while l < left_mid and r < left_size - left_mid:
        if left_half[l] < left_half[left_mid + r]:
            left_sorted.append(left_half[l])
            l += 1
        else:
            left_sorted.append(left_half[left_mid + r])
            r += 1
        k += 1
    while l < left_mid:
        left_sorted.append(left_half[l])
        l += 1
    while r < left_size - left_mid:
        left_sorted.append(left_half[left_mid + r])
        r += 1
else:
    left_sorted = sorted(left_half)

if right_size > 1:
    right_mid = right_size // 2
```

```
    right_half[:right_mid] = sorted(right_half[:right_mid])
    right_half[right_mid:] = sorted(right_half[right_mid:])
    l, r, k = 0, 0, 0
    while l < right_mid and r < right_size - right_mid:
        if right_half[l] < right_half[right_mid + r]:
            right_sorted.append(right_half[l])
            l += 1
        else:
            right_sorted.append(right_half[right_mid + r])
            r += 1
        k += 1
    while l < right_mid:
        right_sorted.append(right_half[l])
        l += 1
    while r < right_size - right_mid:
        right_sorted.append(right_half[right_mid + r])
        r += 1
else:
    right_sorted = sorted(right_half)

i, j, k = 0, 0, 0
while i < len(left_sorted) and j < len(right_sorted):
    if left_sorted[i] < right_sorted[j]:
```

```
            arr[k] = left_sorted[i]
            i += 1
        else:
            arr[k] = right_sorted[j]
            j += 1
        k += 1

    while i < len(left_sorted):
        arr[k] = left_sorted[i]
        i += 1
        k += 1

    while j < len(right_sorted):
        arr[k] = right_sorted[j]
        j += 1
        k += 1

for i in arr:
    print(i,end=' ')
```

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

```
num_elements = int(input())

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

n = len(array)

for i in range(n):

    swapped = False

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

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

            array[j], array[j+1] = array[j+1], array[j]

            swapped = True

    if not swapped:

        break



for i in array:

    print(i,end=' ')
```

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]

```
n = int(input())

A = list(map(int, input().split()))


peaks = []


if n == 1:

    peaks.append(A[0])

else:

    if A[0] >= A[1]:

        peaks.append(A[0])

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

        if A[i] >= A[i - 1] and A[i] >= A[i + 1]:

            peaks.append(A[i])

    if A[n - 1] >= A[n - 2]:
```

```
        peaks.append(A[n - 1])


print(" ".join(map(str, peaks)))
```

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.

```
n = int(input())
lst = list(map(int, input().split()))
k = int(input())

found = False
num_set = set()

for num in lst:
    if k - num in num_set:
        found = True
        break
    num_set.add(num)

if found:
    print("Yes")
else:
    print("No")
```

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

```
a=input().split(',')
b=input()
if b in a:
    print("True")
else:
    print("False")
```