DAY-3

1) Write a program to perform the following
An empty list
A list with one element
A list with all identical elements
A list with negative numbers
Test Cases:
1. Input: []
Expected Output: []
2. Input: [1]
Expected Output: [1]
3. Input: [7, 7, 7, 7]
Expected Output: [7, 7, 7, 7]
4. Input: [-5, -1, -3, -2, -4]
Expected Output: [-5, -4, -3, -2, -1]

CODE:

```
test_cases = [

    [],

    [1],

    [7, 7, 7, 7],

    [-5, -1, -3, -2, -4]

]

test_case_number = 1

for test_case in test_cases:

    if len(test_case) > 1:

        result = sorted(test_case)

    else:

        result = test_case

    print(f"Test Case {test_case_number}:")

    print(f"Input: {test_case}")

    print(f"Expected Output: {result}")

    print("-" * 30)

    test_case_number += 1
```

OUTPUT:

```
Test Case 1:
Input: []
Expected Output: []
------------------------------
Test Case 2:
Input: [1]
Expected Output: [1]
------------------------------
Test Case 3:
Input: [7, 7, 7, 7]
Expected Output: [7, 7, 7, 7]
------------------------------
Test Case 4:
Input: [-5, -1, -3, -2, -4]
Expected Output: [-5, -4, -3, -2, -1]
```

2)Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness.
Sorting a Random Array:
Input: [5, 2, 9, 1, 5, 6]
Output: [1, 2, 5, 5, 6, 9]
Sorting a Reverse Sorted Array:
Input: [10, 8, 6, 4, 2]
Output: [2, 4, 6, 8, 10]
Sorting an Already Sorted Array:
Input: [1, 2, 3, 4, 5]
Output: [1, 2, 3, 4, 5]

CODE:

```
arr = [5, 2, 9, 1, 5, 6]

n = len(arr)

for i in range(n):

    min_idx = i

    for j in range(i + 1, n):

        if arr[j] < arr[min_idx]:

            min_idx = j

    arr[i], arr[min_idx] = arr[min_idx], arr[i]

print("Sorted array:", arr)
```

OUTPUT:

```
Sorted array: [1, 2, 5, 5, 6, 9]
```

3) Write code to modify bubble_sort function to stop early if the list becomes sorted before all passes are completed.

CODE:

```
def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        swapped = False
         for j in range(0, n-i-1):
              if arr[j] > arr[j+1]:
               arr[j], arr[j+1] = arr[j+1], arr[j]
               swapped = True
        if not swapped:
            break
arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print("Sorted array:", arr)
```

OUTPUT:

```
[11, 12, 22, 25, 34, 64, 90]
```

4) Write code for Insertion Sort that manages arrays with duplicate elements during the sorting process. Ensure the algorithm's behavior when encountering duplicate values, including whether it preserves the relative order of duplicates and how it affects the overall sorting outcome.
Examples:
1. Array with Duplicates:
o Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
o Output: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]

<u>CODE:</u>

```
def insertion_sort(arr):

    # Traverse from 1 to len(arr)

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and arr[j] > key:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]

insertion_sort(arr)

print("Sorted array:", arr)
```

OUTPUT:

```
Sorted array: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
```

5) Given an array arr of positive integers sorted in a strictly increasing order, and an integer k. return the kth positive integer that is missing from this array.
Example 1:
Input: arr = [2,3,4,7,11], k = 5
Output: 9

<u>CODE:</u>

```
def findKthPositive(arr, k):

    missing_count = 0

    current_num = 1

    index = 0

    while True:

        if index < len(arr) and arr[index] == current_num:

            index += 1

        else:

            missing_count += 1

            if missing_count == k:

                return current_num

        current_num += 1

arr = [2, 3, 4, 7, 11]

k = 5

result = findKthPositive(arr, k)

print("The", k, "th missing positive integer is:", result)
```

OUTPUT:

```
9
```

6) A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time.
Example 1:
Input: nums = [1,2,3,1]
Output: 2

CODE:

```
def findPeakElement(nums):
    left, right = 0, len(nums) - 1

    while left < right:

        mid = (left + right) // 2

        if nums[mid] < nums[mid + 1]:

            left = mid + 1

        else:

            right = mid

    return left

nums = [1, 2, 3, 1]

peak_index = findPeakElement(nums)

print("The peak element is at index:", peak_index)
```

OUTPUT:

```
The peak element is at index : 3
```

7) Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.
Example 1:
Input: haystack = "sadbutsad", needle = "sad"
Output: 0

CODE:

```
def str(haystack, needle):
    return haystack.find(needle)
haystack = "darksouls"
needle = "souls"
result = str(haystack, needle)
print("The index of the first occurrence is:", result)
```

OUTPUT:

```
The index of the first occurrence is:4
```

8) Given an array of string words, return all strings in words that is a substring of another word. You can return the answer in any order. A substring is a contiguous sequence of characters within a string
Example 1:
Input: words = ["mass","as","hero","superhero"]
Output: ["as","hero"]
Explanation: "as" is substring of "mass" and "hero" is substring of "superhero".
["hero","as"] is also a valid answer.

CODE:

```
def stringMatching(words):

    result = []

    for i in range(len(words)):

        for j in range(len(words)):

            if i != j and words[i] in words[j]:

                result.append(words[i])

                break


    return result

words = ["Booyaka", "yaka", "rio", "mysterio"]

output = stringMatching(words)

print("The substrings are:", output)
```

OUTPUT:

```
The substring are[‘yaka’, ‘rio’]
```