18.Write a program to perform the following
o An empty list
o A list with one element
o A list with all identical elements
o A list with negative numbers
Test Cases:
1.Input: [7, 7, 7, 7]
o Expected Output: [7, 7, 7, 7]
Aim: To create and print different types of lists including an empty list a list with one elment a list with all identical elements and a list with negative numbers
Algorithm:
step1: Create an empty list using the [] syntax.
step2:creat a function to creat list with one element
step3: create a function to creat a list with all identical element
step4: test each function with example use cases
Program:

```
identical_elements_list = [7, 7, 7, 7]
print(identical_elements_list)
```

Input:

```
identical_elements_list = [7, 7, 7, 7]
```

Output:

```
Output
[7, 7, 7, 7]
```

Time complexity:t(n)=O(n)
Result: This program demonstrates how to handle different list scenerios in python including initialization.

19.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]

Aim: The aimof selection sort is to sort an array of elements in ascending order

Algorithm:

step1: initialize the sorted region to the empty

step2:repeat until the entire array is sorted find smallest element unsorted region to swap the small element

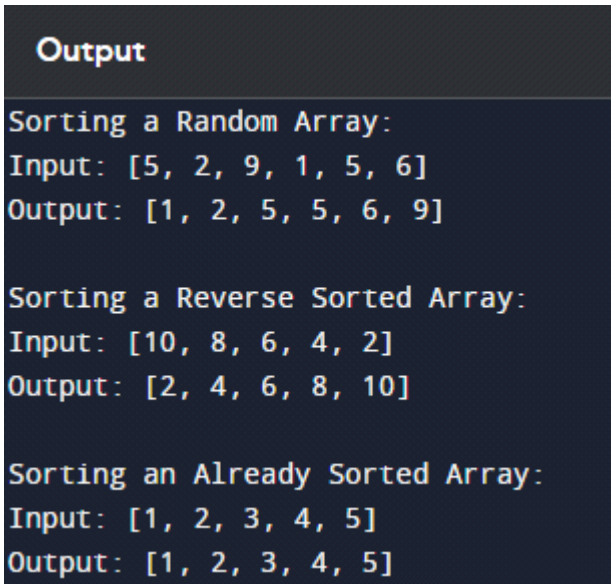
Program:

```
def selection_sort(arr):
    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]
    return arr
print("Sorting a Random Array:")
arr = [5, 2, 9, 1, 5, 6]
print("Input:", arr)
print("Output:", selection_sort(arr))
print("\nSorting a Reverse Sorted Array:")
arr = [10, 8, 6, 4, 2]
print("Input:", arr)
print("Output:", selection_sort(arr))
print("\nSorting an Already Sorted Array:")
arr = [1, 2, 3, 4, 5]
print("Input:", arr)
print("Output:", selection_sort(arr))
```

Input:

```
print("Sorting a Random Array:")
arr = [5, 2, 9, 1, 5, 6]
print("\nSorting a Reverse Sorted Array:")
arr = [10, 8, 6, 4, 2]
print("\nSorting an Already Sorted Array:")
arr = [1, 2, 3, 4, 5]
```

Output:



Time complexity:t(n)=O(n^2)

Result: This program demonstrates how to handle entire array is sorted

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

Test Cases:

Test your optimized function with the following lists:

1. Input: [64, 25, 12, 22, 11]

- **Expected Output: [11, 12, 22, 25, 64]**

2. Input: [29, 10, 14, 37, 13]

- **Expected Output: [10, 13, 14, 29, 37]**

Aim: To modify the optimization of bubble sort algorithm to sort easily in the list

Algorithm:

step1:Initialize a flag swapped to 'false'.

step2: Iterate through the list 'n-1' times.

step3:in each iteration, through the list first element to 'n-i-1' elements.

step4:If an element greater than element.

Program:

```python
def optimized_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
if __name__ == "__main__":
    test_list = [64, 25, 12, 22, 11]
    optimized_bubble_sort(test_list)
    print("Sorted list:", test_list)
```

Input:

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

Output:

```
Sorted list: [11, 12, 22, 25, 64]
```

Time complexity:T(n)=O(logn)

Result:The function will return the index of peak element input array. if array upto multiple peaks index of any of the peaks.

21. 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]

Aim :The aim is to implement on intersection sort of algorithm then can efficiency sort array with duplicate elements.

Algorithm :

step1:Iterate from index 1 to the end of the array.

step2:for each element ,compare if with previous elements and shift those elements are position to right.

step3:Insert element at current position.

Program:

```python
def insertionSort(arr):
    n = len(arr)

    if n <= 1:
        return

    for i in range(1, n):
        key = arr[i]
        j = i - 1

        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

input_array = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
insertionSort(input_array)
print("Sorted array is:", input_array)

```

Input:

```python
input_array = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
```

Output:

```
Output

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

Time complexity:T(n)=O(n^2)

Result:The result of the program sorted array=[1,12,3,3,4,5,5,6,9]

22.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

Aim :To find the kth positive integer that is missing from the array.

Algorithm :

step1:Initilize created to1 and missing count to "0"

step2:Initlize a variable to 1 ,which will be used to check if current number missing from an array.

step3:Iterate through the array and for check if current number is not equal to element.

step4:If the current number is not equal to the element.

Program:

```python
def find_kth_missing(arr, k):
    missing_set = set(range(1, arr[-1] + k + 1)) - set(arr)
    return sorted(missing_set)[k - 1]

arr = [2, 3, 4, 7, 11]
k = 5
output = find_kth_missing(arr, k)
print(output)
```

Input:

```
input_array = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
```

Output:

```
Output

9
```

Time complexity:T(n)=O(n)
Result:The function will return the kth positive integer that is missing for input array.

23. 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
Explanation: 3 is a peak element and your function should return the index number 2.
Aim: To find the peak element in the given list of array and return its index
Algorithm:
step1:Initialize 2 pointers respectfully where 'n' is length of array.
step2: While 'low' is less than original then return 'mid'.
step3:If it is greater than both and then mid is peak element
step4:Repeat step 3 2-5 untill it is greater
Program:

```python
def find_peak(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = left + (right - left) // 2
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1
    return left
nums1 = [1, 2, 3, 1]
print(find_peak(nums1))
```

Input:

```
nums1 = [1, 2, 3, 1]
```

Output:

```
2
```

Time complexity:T(n)=O(logn)

Result:The function will return the index of peak element input array. if array upto multiple peaks index of any of the peaks.

24. 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
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
Example 2:
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.

Aim: The aim is to return the index value where the 1st occurence of common element exist in both the strings

Algorithm:

step1:Preprocess the forst string to create partial match table.
step2: search for the first string in the second string

Program:

```
def strStr(haystack: str, needle: str) -> int:
    if not needle:
        return 0
    for i in range(len(haystack) - len(needle) + 1):
        if haystack[i:i + len(needle)] == needle:
            return i
    return -1
haystack1 = "sadbutsad"
needle1 = "sad"
print(strStr(haystack1, needle1))
```

Input:

```
("sadbutsad", "sad"))
```

Output:

```
0
```

Time complexity:T(n)=O(n)

Result:The index value of 1st occurence in both strings is successfully exeuted

25. 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.

Aim: To find the number of substrings in a given array

Algorithm:

step1:Iterate through each word in list.
step2: For each word, check if it is a sub string of any other word in the list.
step3:collect all such words that are sub strings of any other word.

Program:

```
def find_substrings(words):
    result = set()
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j:
                if words[i] in words[j]:
                    result.add(words[i])
    return list(result)
words = ["mass", "as", "hero", "superhero"]
print(find_substrings(words))
```

Input:

```
words = ["mass", "as", "hero", "superhero"]
```

Output:

```
['hero', 'as']
```

Time complexity:T(n)=O(n^2)

Result:Number of substrings in a string of arrays is successfully executed