TASK 5 : Implement various searching and sorting operations in python Programming

5.1

AIM :- To write a python program that takes a sorted array of integers (which may include negative numbers), Computes the square of each number, and returns a new array with the squard values sorted in non-decreasing order.

Algorithm:

1. start
2. Read the input array nums
3. initalize.
   - n = length of nums
   - res = list of size n filled with 0 (to store results)
   - Two pointers:
     l = 0 (left index)
     r = n-1 (right index)
     Pos = n-1 (Position to fill results from rightmost End).
4. Repeat while k=n.
   1. if |nums[l]| > |nums[r]| →
      - Square nums[l] and assign to res[Pos]
      - increment l by 1
   2. Else →
      - Square nums[r] and assign to res[Pos]
      - Decrement r by 1

Program :

```
def sorted_squares(nums):
    n = len(nums)
    res = [0] * n
    l, r = 0, n-1
    pos = n-1   # Fill from the end

    while l <= r:
        if abs(nums[l]) > abs(nums[r]):
            res[pos] = nums[l] * nums[l]
            l += 1
        else:
            res[pos] = nums[r] * nums[r]
            r -= 1
        pos -= 1
    return res

# --- Function calling & printing
nums1 = [-4, -1, 0, 3, 10]
nums2 = [-7, -3, 2, 3, 11]
Print("input:", nums1)
Print("output:", sortedSquares(nums1))
Print("\ninput:", nums2)
Print("output:", sortedSquares(nums2))
```

- Decrement pos by 1

5. After the loop ends, res contains all squared numbers in sorted order.
6. Return res
7. End

Example 1:

Input : nums = [-4, -1, 0, 3, 10]

output : [0, 1, 9, 16, 100]

- Compare $|-4| = 4$ and $|10| = 10$ → place 100 at res[4].
- Compare $|-4| = 4$ and $|3| = 3$ → place 16 at res[3]
- compare $|-1| = 1$ and $|3| = 3$ → place 9 at res[2]
- Compare $|-1| = 1$ and $|0| = 0$ → place 1 at res[1].
- place 0 at res[0]

Final result [0, 1, 9, 16, 100].

output : [0, 1, 9, 16, 100]

5.2

AIM : To write a python program that reads a list of numbers, sorts them in ascending order using the Bubble sort algorithm, and prints the sorted list.

Algorithm:

1. start
2. Read the number of elements n
3. Read n integers into a list arr.
4. Repeat the following for i=0 to n-1:
    - For each j=0 to n-i-2:
        - if arr[j]> arr[j+1]:
            - swap arr[j] and arr[j+1].
5. After all pases, the list will be sorted in ascending order
6. print sorted list
7. End

Input: Enter number of elements : 5

Enter the elements : 64 34 25 12 22

output:

sorted list : [12, 22, 25, 34, 64]

Program :

```
def bubble_sort (arr):
    n = len(arr)
#Traverse through all elements
for i in range(n):
    #last i elements are already sorted
for j in range (0, n-i+1):
    #swap if the element found is greater than the
    next
if arr[j] > arr[j+1]:
    arr[j], arr[j+1] = arr[j+1], arr[j]
return arr
# --- Main program ---
n = int (input ("Enter number of elements: "))
arr = list(map (int, input("Enter the elements: ").split()))
Sorted_arr = bubble_sort (arr)
Print ("sorted list:", sorted_arr)
```

5.3

AIM: To write a python program that sorts a list of elements in ascending order using the merge sort algorithm.

Algorithm: (Merge sort):

Merge sort follows the divide and Conquer stantegy

1. Divide:
   - split the list into two halves until each sub-list Contains a singh element.
2. Conquer (sort):
   - Recursively sort the two halves.
3. Combine (merge):
   - Merge the sorted halves into a single sorted list.

step by step:

1. if the list has 1 or 0 elements, it is already sorted (base case).
2. otherwise
   - Find the middle index.
   - Recursively apply merge sorted on the left half.
   - Recursively apply merge sort on the right half.
   - Merge the two sorted halves into a final sorted list.

Program :

```
# function to merge two halves
def merge_sort(arr):
    if len(arr)>1:
        # Find the middle point
        mid = len(arr)//2
    # divide into two halves
        left_half = arr[:mid]
        right_half = arr[mid:]
    # Recursive call for each half
        merge_sort (left_half)
        merge_sort (right_half)
    # merging process
    i=j=k=0
# Compare and merge
    while i< len(left_half) and j<len(right_half):
        if left_half[i]< right_half[j]:
            arr[k] = left_half[i]
            i+=1
        else:
            arr[k] = right_half[j]
            j+=1
        k+=1
# copy remaining elements
    while i<len(left_half):
        arr[k] = left_half[i]
        i+=1
        k+=1
```

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

# main program
n = int(input("Enter number of elements: "))
arr = list(map(int, input("Enter elements: ").split()))
Print("original list:", arr)
merge_sort(arr)
Print("sorted list", arr)
```

Sample I/o

Enter number of Elements : 6

Enter elements : 34 12 45 2 18 7

original list : [34, 12, 45, 2, 18, 7]

sorted list : [2, 7, 12, 18, 34, 45]

5.4

AIM: To write a python program that searches for a given target Element in a sorted array using Binary search and returns its index if found, otherwise returns -1.

The program must satisfy the given Constraints and run in O(logn) time Complexity.

Algorithm:

1. set lft = 0, right = len(nums) - 1.
2. while left <= right:
   - Compute middle index : mid = (left + right) // 2.
   - if nums[mid] = target, return mid.
   - if nums[mid] < target, search the right half (left = miad + 1).
   - Else, search the left half (right = mid - 1).
3. if the loop Ends without finding the element, return -1.

Sample I/O

Enter sorted elements : -10 -3 0 5 9 12
Enter target element : 9
Target found at index 4

Program

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

    while left <= right:
        mid = (left + right) // 2 # middle index

        if nums[mid] == target:
            return mid # Target found
        elif nums[mid] < target:
            left = mid + 1 #
        else
            right = mid - 1 # search left half

    return -1 # Target not found

# main program
nums = list(map(int, input("Enter sorted elements: ").split()))
target = int(input("Enter target Element: "))

result = binary_search(nums, target)
if result != -1:
    Print(f"Target found at index {result}")
Else:
    Print("Target not found (-1)")
```

Program:

```
def find_peaks(A, n):
    peaks = []
    for i in range(n):
        # first element condition
        if i == 0 and n>1 and A[i] >= A[i+1]:
            peaks.append(A[i])
        # last element condition
        elif i == n-1 and n>1 and A[i] >= A[i-1]:
            peaks.append(A[i])
        # middle elements condition
        elif 0<i<n-1 and A[i] >= A[i-1] and A[i] >= A[i+1]:
            peaks.append(A[i])
        # special case: if only one element
        elif n == 1:
            peaks.append(A[i])
    return peaks

# main program
n = int(input("Enter number of elements: "))
A = list(map(int, input("Enter elements: ").split()))

peaks = find_peaks(A, n)

print("Peak elements:", peaks)
```

5.5

AIM: To write a python program that finds all peak elements in a list. A peak element is defined as an element that is greater than or equal to its neighbours.

- For first element (i=0): A[0] >= A[1]
- For last element (i=n-1): A[n-1] >= A[n-2]
- For middle elements (0<i<n-1): A[i] >= A[i-1] and A[i] >= A[i+1]

Algorithm:

1. Read integers n (size of the array).
2. Read n Elements into Array A.
3. intialize an Empty list peaks.
4. For each index i in array.
5. print all Elements in peaks seperated by space.

Sample I/O

```
Enter number of elements : 7
Enter elements           : 1 3 2 5 7 6 4
Peak elements            : 3 7
```

VEL TECH	
EX No.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	5
SIGN WITH DATE	

3/9

22

RESULT: Thus, the implement of various sep searching and sorting operations in python programming has been Executed Successfully.