TASK : 5
DATE: 31/1/25

IMPLEMENT VARIOUS SEARCHING AND SORTING OPERATIONS IN PYTHON PROGRAMMING

AIM:-

To write a Python Program to Implement various searching and sorting operations.

TASK 5-1:-

AIM:-

To write a Python program that takes a sorted array of integers, computes the square of each number, and returns a new array with the squared values sorted in non-decreasing order.

ALGORITHM:-

1:- Start

2:- Read the input array nums (sorted in non-decreasing order).

3:- Initialize:

* n = length of nums
* res = list of size n filled with 0 (to store result)
* Two pointers:
  - l = 0 (left index)
  - r = n-1 (right index)
  - pos = n-1 (position to fill results from rightmost end).

4:- Repeat while l <= r:

* if |nums[l]| > |nums[r]| →
  - Square nums[l] and assign to res[p
  - Increment l by 1
* Else →

- Square nums[r] and assign to res[pos]
- Decrement r 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]

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]

PROGRAMS :-

```
def sortedSquare(nums):
    n = len(nums)
    res = [0] * n
    l, r = 0, n-1
    pos = n-1
    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
nums1 = [-4, -1, 0, 3, 10]
nums2 = [-7, -3, 2, 3, 11]
Print("Input:", nums1)
Print("Output:", sortedSquares(nums1))
Print("\n Input:", nums2)
Print("Output:", sortedSquares(nums2))
```

TASK : 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 passes, the list will be sorted in ascending order.
6) Print the sorted list
7) End.

Input:-

Enter number of elements : 5
Enter the elements : 64, 34 25 12 22

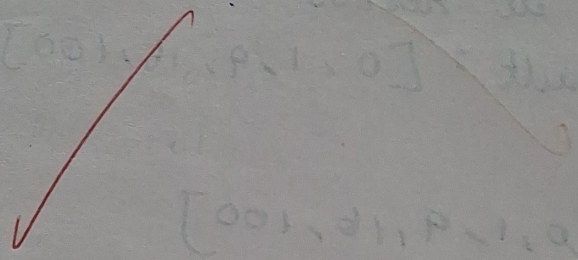
Output:-

Sorted List : [12, 22, 25, 34, 64]

PROGRAM:-

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

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)
```



TASK : 5.3

AIM:-

To write a Python Program that sorts a list of elements using the merge sort algorithm.

ALGORITHM :-

Merge Sort follows the Divide and Conquer strategy

1, Divide :
Split the list into two halves until each sub-list contains a single element.

2 Conquer :
Recursively sort the two halves.

3, Combine :
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

2, Otherwise,

- Find the middle index
- Recursively apply merge sort on the left half
- Recursively apply merge sort on the right half.
- Merge the two sorted halves into a final sorted list.

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]

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)
        left_half = arr[:mid:]
        merge_sort(left_half)
        merge_sort(Right_half)
        i = j = k = 0
        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
        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

= int(input("Enter number of elements :"))
r = list(map(int, input("Enter elements :").split()))
int("Original list:", arr)
rge_sort(arr)
t("Sorted list :", arr)
```

TASK : 5-4

AIM :-

To write a Python Program that searches for a given target element in a sorted array using Binary Search and returns the index if found, otherwise returns -1. The program must satisfy the given constraints and run in O(log n) time complexity.

ALGORITHM :-

1) Set left = 0, right = len(nums) - 1.
2) While left <= right:
   - compute middle index : mid = (left + right)
   - If nums[mid] == target, return mid.
   - If nums[mid] < target, search the right half (left = mid + 1)
   - Else, search the left half (right = mid - 1).
3) If the loop ends without finding the element, return -1.

I/O :-

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

PROGRAMS:-

```
def binary_search(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right)
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
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)")
```

TASK : 5.5

AIM:-

To create 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 neighbors :

- 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 integer n (size of the array).
2) Read n elements into array A.
3) Initialize an empty list peaks.
4) For each index i in array :
   - if i == 0 and A[i] >= A[i+1], add A[i] to peaks
   - if i == n-1 and A[i] >= A[i-1], add A[i] to peaks
   - if 0 < i < n-1 and A[i] >= A[i-1] and A[i] >= A[i+1] add A[i] to peaks.
5) Print all elements in peaks separated by space

I/O :-

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

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

RESULT:-

Thus, a Python program to implement linear searching and sorting operations is successfully verified.

PROGRAMS :-

```
def find-peaks(A, n):
    Peaks = []
    for i in range(n):
        if i == 0 and n > 1 and A[i] >= A[i+1]:
            Peaks.append(A[i])
        elif i == n-1 and n > 1 and A[i] >= A[i-1]:
            peaks.append(A[i])
        elif 0 < i < n-1 and A[i] >= A[i-1] and A[i] >= A[i+1]:
            Peaks.append(A[i])
        elif n == 1:
            Peaks.append(A[i])
    return peaks
n = int(input("Enter number of elements :"))
A = list(map(int, input("Enter elements :").split()))
Peaks = find-peaks(A, n)
Print("Peak elements :", *peaks)
```