**10211CS213- Python Programming**

**Task 5:** Implement various Searching and Sorting Operations in python programming

**5.1** Given an integer array nums sorted in **non-decreasing** order, return *an array of****the squares of each number****sorted in non-decreasing order*.

**Constraints:**

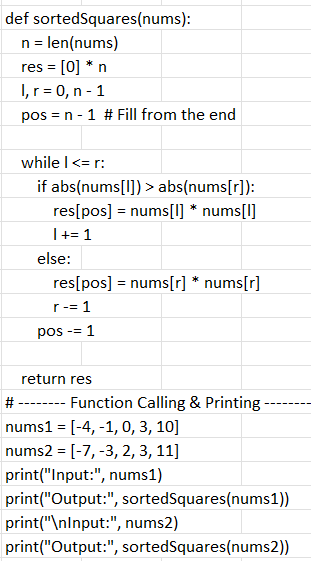
* 1 <= nums.length <= 104
* -104 <= nums[i] <= 104
* nums is sorted in **non-decreasing** order.

**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 squared values sorted in non-decreasing order.

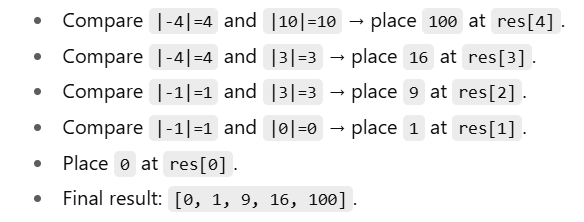
**Algorithm (Two-Pointer Method)**

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 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** l <= r:
   * If |nums[l]| > |nums[r]| →
     + Square nums[l] and assign to res[pos]
     + Increment l by 1
   * Else →
     + Square nums[r] and assign to res[pos]
     + Decrement r by 1
   * 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]

 --------------------------------------------------------------------------------------------------------------------------------------

**5.2** Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. You read an list of numbers. You need to arrange the elements in ascending order and print the result. The sorting should be done using bubble sort.

**Input Format:**The first line reads the number of elements in the array. The second line reads the array elements one by one.  
**Output Format:** The output should be a sorted list.

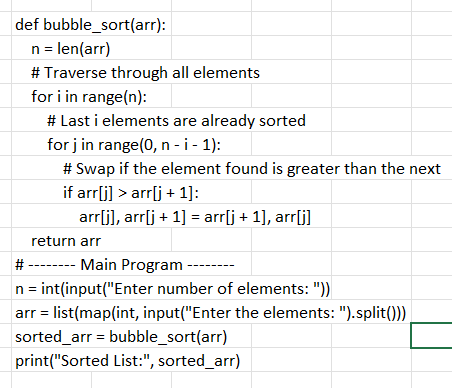
**Aim**

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 (Bubble Sort)**

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**

**Program**

****

**Input:**

Enter number of elements: 5

Enter the elements: 64 34 25 12 22

**Output:**

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

-------------------------------------------------------------------------------------------------------------------------------------

**5.3** Write a Python program to sort a list of elements using the merge sort algorithm.

**Aim:**

To write a Python program that sorts a list of elements in ascending order using the **Merge Sort algorithm**.

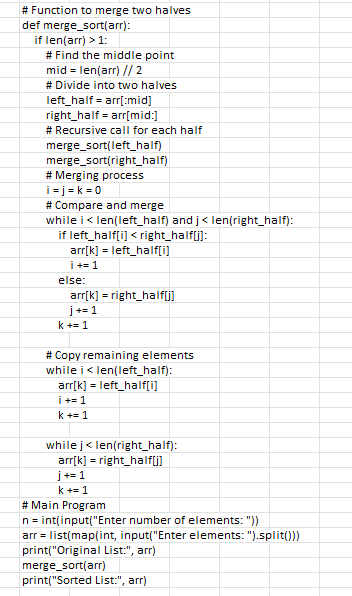
**Algorithm (Merge Sort):**

Merge Sort follows the **Divide and Conquer** strategy.

1. **Divide:**
   * Split the list into two halves until each sub-list contains a single 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 sort on the left half.
   * Recursively apply merge sort on the right half.
   * Merge the two sorted halves into a final sorted list.



**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** Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

**Constraints:**

* 1 <= nums.length <= 104
* -104 < nums[i], target < 104
* All the integers in nums are **unique**.
* nums is sorted in ascending order.

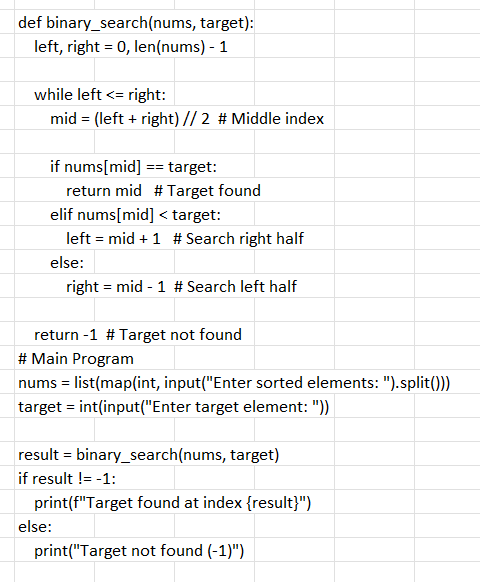
**Aim:**

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(log n)** time complexity.

**Algorithm**

1. Set left = 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 = mid + 1).
   * Else, search the left half (right = mid - 1).
3. If the loop ends without finding the element, return -1.

**Program**

****

**Sample I/O**

Enter sorted elements: -10 -3 0 5 9 12

Enter target element: 9

Target found at index 4

------------------------------------------------------------------------------------------------------------------------------

**5.5** 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]

**Aim:**

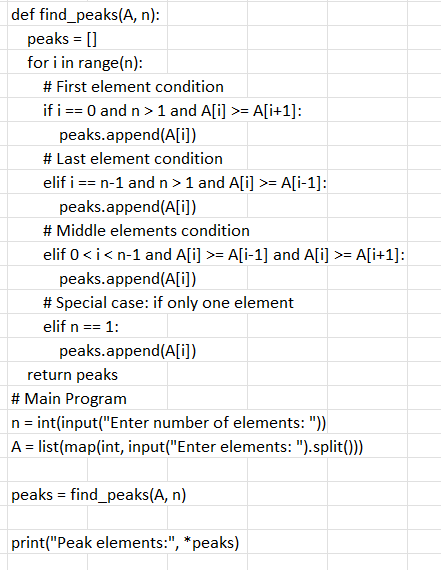
To write a Python program that finds all **peak elements** in a list.  
A **peak element** is defined as an element that is **greater than or equal to its neighbors**:

* 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.

**Program**



**Sample I/O**

Enter number of elements: 7

Enter elements: 1 3 2 5 7 6 4

Peak elements: 3 7