DAY 30/180

Q1-An array is given in decreasing order with Key, Find the index of key, if key is not present, print -1.

```cpp
// Binary Search function to find the index of 'key' in the 'arr'
int binarysearch(vector<int>& arr, int key) {
    int n = arr.size();
    int s = 0;          // Initialize the start of the search range
    int e = n - 1;      // Initialize the end of the search range

    while (s <= e) {
        int mid = (s + e) / 2;  // Calculate the middle index

        if (arr[mid] == key) {
            return mid;  // If the key is found, return its index
        } else if (arr[mid] > key) {
            e = mid - 1;  // If the key is smaller, adjust the end of the search range
        } else {
            s = mid + 1;  // If the key is larger, adjust the start of the search range
        }
    }

    return -1;  // Return -1 if the key is not found in the array
}

int main() {
    int n;
    cin >> n;  // Input the number of elements in the array
    vector<int> arr(n);  // Create a vector to store the elements

    for (int i = 0; i < n; i++) {
        cin >> arr[i];  // Input the elements of the array
    }

    int key;
    cin >> key;  // Input the key to search for

    int index = binarysearch(arr, key);  // Call the binary search function

    cout << index << endl;  // Output the index of the key (or -1 if not found)
}
```

Q2- Search Insert Position (LeetCode).

```cpp
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int s = 0;                // Initialize the start of the search range
        int n = nums.size();      // Get the number of elements in the 'nums' vector
        int e = n - 1;            // Initialize the end of the search range
        int mid = s + (e - s) / 2; // Calculate the initial middle index

        while (s <= e) {
            if (nums[mid] > target) {
                e = mid - 1;         // Adjust the end of the search range if the middle element is greater than the target
            }
            else if (nums[mid] < target) {
                s = mid + 1;         // Adjust the start of the search range if the middle element is less than the target
            }
            else if (nums[mid] == target) {
                return mid;           // Return the index of the target if it is found in the 'nums' vector
            }
            mid = s + (e - s) / 2;    // Recalculate the middle index within the updated search range
        }

        return mid;  // If the target is not found, return the last calculated 'mid'
    }
};
```