DAY 32/180

Q1- Peak Index in a Mountain Array

```cpp
class Solution {
public:
    int peakIndexInMountainArray(vector<int>& arr) {
        int s = 0; // Initialize start pointer to the beginning of the array
        int e = arr.size() - 1; // Initialize end pointer to the end of the array
        // Use binary search to find the peak index
        while (s <= e) {
            int mid = (s+e)/2; // Calculate the middle index
            if (arr[mid] < arr[mid + 1]) {
                // If the value at the middle index is less than the value at the next index,
                // it means we are on the ascending slope of the mountain, so update the start pointer.
                s = mid + 1;
            } else {
                // If the value at the middle index is greater than or equal to the value at the next index,
                // it means we are on the descending slope of the mountain, so update the end pointer.
                e = mid - 1;
            }
        }
        // The loop will terminate when 's' points to the peak element
        return s; // Return the peak index
    }
};
```

Q2- Find Minimum in Rotated Sorted Array.

```cpp
class Solution {
public:
    // Function to find the minimum element in a rotated sorted array
    int findMin(vector<int>& nums) {
        int n = nums.size(); // Get the number of elements in the array
        int s = 0; // Initialize the start pointer to the beginning of the array
        int e = n - 1; // Initialize the end pointer to the end of the array
        int ans = 1e9; // Initialize a variable to store the minimum value, initially set to a large value
        // Use binary search to find the minimum element
        while (s <= e) {
            int mid = (s + e)/2; // Calculate the middle index
            if (nums[s] <= nums[mid]) {
                // If the value at the start pointer is less than or equal to the value at the middle index,
                // it means the left part of the array is sorted, so update the minimum value and move the start pointer.
                ans = min(ans, nums[s]);
                s = mid + 1;
            } else if (nums[e] >= nums[mid]) {
                // If the value at the end pointer is greater than or equal to the value at the middle index,
                // it means the right part of the array is sorted, so update the minimum value and move the end pointer.
                ans = min(ans, nums[mid]);
                e = mid - 1;
            }
        }
        return ans; // Return the minimum element in the rotated sorted array
    }
};
```

Q3- Search in a Rotated Sorted Array.

```cpp
int search(vector<int>& nums, int target) {
    int n = nums.size(); // Get the number of elements in the array
    int s = 1; // Initialize the start pointer to the second element (index 1)
    int e = n - 2; // Initialize the end pointer to the second-to-last element
    while (s <= e) {
        int mid = s + (e - s) / 2; // Calculate the middle index

        if (nums[mid] > nums[mid - 1] && nums[mid] > nums[mid + 1]) {
            // If the middle element is greater than its neighbors, it's a peak, and the next element is the minimum.
            return nums[mid + 1];
        }

        // Check which half of the array is sorted
        if (nums[s] <= nums[mid]) {
            // Left half is sorted

            if (nums[s] <= target && target <= nums[mid]) {
                // If the target is within the range of the left half, update the end pointer.
                e = mid - 1;
            } else {
                // Otherwise, update the start pointer.
                s = mid + 1;
            }
        } else {
            // Right half is sorted

            if (nums[mid] <= target && target <= nums[e]) {
                // If the target is within the range of the right half, update the start pointer.
                s = mid + 1;
            } else {
                // Otherwise, update the end pointer.
                e = mid - 1;
            }
        }
    }
    return -1; // If the target is not found, return -1.
}
```

Q4- Kth Missing Positive Number.

```cpp
class Solution {
public:
    // Function to find the k-th missing positive integer in a sorted array
    int findKthPositive(vector<int>& arr, int k) {
        int n = arr.size(); // Get the number of elements in the array
        int s = 0; // Initialize the start pointer to 0
        int e = n - 1; // Initialize the end pointer to the last index

        while (s <= e) {
            int mid = (s + e) / 2; // Calculate the middle index
            int missing = arr[mid] - (mid + 1); // Calculate the number of missing positive integers in the current range

            if (missing < k) {
                // If the number of missing positive integers in the current range is less than k,
                // update the start pointer to search in the right half.
                s = mid + 1;
            } else {
                // If the number of missing positive integers in the current range is greater than or equal to k,
                // update the end pointer to search in the left half.
                e = mid - 1;
            }
        }
        return s + k; // Return the k-th missing positive integer by adding k to the current start pointer.
    }
};
```

Q5- Find Peak Element

```cpp
int findPeakElement(vector<int>& nums) {
    int n = nums.size(); // Get the number of elements in the array
    if (n == 1) {
        return 0; // If there is only one element, it is a peak.
    }
    if (n == 2) {
        // If there are two elements, return the index of the greater element as the peak.
        if (nums[0] < nums[1]) {
            return 1;
        } else {
            return 0;
        }
    }
    // Handle edge cases where the peak may be at the first or last element
    if (nums[0] > nums[1]) return 0;
    if (nums[n - 1] > nums[n - 2]) return n - 1;
    int s = 1; // Initialize the start pointer to the second element
    int e = n - 2; // Initialize the end pointer to the second-to-last element
    int ans = 1e9; // Initialize a variable to store the peak element, initially set to a large value

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

        if (nums[mid] > nums[mid - 1] && nums[mid] > nums[mid + 1]) {
            // If the middle element is greater than its neighbors, it's a peak, and update the answer.
            ans = mid;
            e = mid - 1;
        } else if (nums[mid] < nums[mid - 1]) {
            e = mid - 1; // If the middle element is less than the previous element, move the end pointer.
        } else if (nums[mid] < nums[mid + 1]) {
            s = mid + 1; // If the middle element is less than the next element, move the start pointer.
        }
    }

    return ans; // Return the index of the peak element.
}
```

Q6- Special Array with X elements greater than X.

```cpp
int binarySearch(vector<int>& nums, int x) {
    int n = nums.size(); // Get the number of elements in the array
    int s = 0; // Initialize the start pointer to the beginning of the array
    int e = n - 1; // Initialize the end pointer to the end of the array
    int ans = 0; // Initialize a variable to store the count of elements greater than or equal to x

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

        if (nums[mid] < x) {
            ans = mid + 1; // Update the answer with the current index and move the start pointer to the right.
            s = mid + 1;
        } else {
            e = mid - 1; // If the element at the middle index is greater than or equal to x, move the end pointer to the left.
        }
    }

    return n - ans; // Return the count of elements greater than or equal to x.
}
int specialArray(vector<int>& nums) {
    sort(nums.begin(), nums.end()); // Sort the input array

    for (int i = 1; i <= nums.size(); i++) {
        int count = binarySearch(nums, i); // Count elements greater than or equal to i using binary search

        if (count == i) {
            return i; // If the count matches the current element, return the special array value.
        }
    }

    return -1; // If no such special array is found, return -1.
}
```

Q7- Valid Perfect Square

```cpp
class Solution {
public:
    // Function to check if a number is a perfect square
    bool isPerfectSquare(int num) {
        long long s = 0; // Initialize the start pointer to 0
        long long e = num; // Initialize the end pointer to the given number

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

            long long square = mid * mid; // Calculate the square of the middle value

            if (square == num) {
                return true; // If the square of the middle value is equal to the given number, it's a perfect square.
            } else if (square < num) {
                s = mid + 1; // If the square is less than the given number, move the start pointer to the right.
            } else {
                e = mid - 1; // If the square is greater than the given number, move the end pointer to the left.
            }
        }
        return false; // If no perfect square is found, return false.
    }
};
```

Q8- Seach In Rotated Sorted Array – II

```cpp
bool search(vector<int>& nums, int target) {
    int n=nums.size();
    int s=0,e=n-1;
    while(s<=e){
        int mid=s+(e-s)/2;
        if(nums[mid]==target){
            return 1;
        }
        //check which half is sorted
        if(nums[s]<=nums[mid]){
            if(nums[s]==nums[mid]){
                s++;
                continue;
            }
            else if(nums[s]<=target&&target<=nums[mid]){
                e=mid-1;
            }
            else{
                s=mid+1;
            }
        }
        else{
            if(nums[mid]==nums[e]){
                e--;
                continue;
            }
            if(nums[mid]<=target&&target<=nums[e]){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        mid=s+(e-s)/2;
    }
    return 0;
}
```