```
public class Striver_Binary_Search {
    // BS1--Introduction

    // BS2--Implement Lower Bound and Upper Bound | Search Insert Position | Floor
    // and Ceil
    class Solution {
        // Lower Bound arr[ind]>=x;
        class Solution {

            // Function to find floor of x
            // arr: input array
            // n is the size of array
            static int findLowerBound(long arr[], int n, long ele) {
                int start = 0;
                int end = arr.length - 1;
                int ans = -1;
                while (start <= end) {
                    int mid = (start + end) / 2;
                    if (arr[mid] == ele)
                        return mid;
                    else if (arr[mid] >= ele) {
                        ans = mid;
                        end = mid - 1;
                    } else
                        start = mid + 1;
                }
                return ans;
            }

        }

        // Upper Bound arr[ind]>x;
        class Solution {

            // Function to find floor of x
            // arr: input array
            // n is the size of array
            static int findUpper(long arr[], int n, long ele) {
                int start = 0;
                int end = arr.length - 1;
                int ans = -1;
                while (start <= end) {
                    int mid = (start + end) / 2;
                    if (arr[mid] == ele)
                        return mid;
                    else if (arr[mid] > ele) {
                        ans = mid;
                        end = mid - 1;
                    } else
                    start = mid + 1;
            }
            return ans;
        }

    }
}

// BS3--First and Last Occurrences in Array
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int first = firstOccur(nums, target);
        int lastp = lastpOccur(nums, target);
        if (first == -1) {
            return new int[] { -1, -1 };
        } else
            return new int[] { first, lastp };
    }

    public int firstOccur(int nums[], int target) {
        int start = 0, end = nums.length - 1;
        int ans = -1;
        while (start <= end) {
            int mid = (start + end) / 2;
            if (nums[mid] == target) {
                ans = mid;
                end = mid - 1;
            } else if (nums[mid] < target) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return ans;
    }

    public int lastpOccur(int nums[], int target) {
        int start = 0, end = nums.length - 1;
        int ans = nums.length - 1;
        while (start <= end) {
            int mid = (start + end) / 2;
            if (nums[mid] == target) {
                ans = mid;
                start = mid + 1;
            } else if (nums[mid] < target) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return ans;
    }
}

// BS4--Search in Rotated Sorted Array I
class Solution {
    public int search(int[] arr, int target) {
        int low = 0;
        int high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == target) {
                return mid;
            } else if (arr[low] <= arr[mid]) {
                if (arr[low] <= target && target < arr[mid]) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            } else {
                // [mid, high] is sorted
                if (arr[mid] < target && target <= arr[high]) {
                    low = mid + 1;
                } else {
                    high = mid - 1;
                }
            }
        }
        return -1;
    }
}

// BS5--Search in Rotated Sorted Array II
class Solution {
    public boolean search(int[] arr, int target) {
        int low = 0;
        int high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == target) {
                return true;
            }
            if (arr[low] == arr[mid] && arr[mid] == arr[high]) {
                low++;
                high--;
                continue;
            } else if (arr[low] <= arr[mid]) {
                if (arr[low] <= target && target < arr[mid]) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            } else {
                if (arr[mid] < target && target <= arr[high]) {
                    low = mid + 1;
                } else {
                    high = mid - 1;
                }
            }
        }

        return false;
    }
}

// BS6--Find Minimum in Rotated Sorted Array
class Solution {
    public int findMin(int[] arr) {
        int n = arr.length;
        int low = 0, high = n - 1;
        int ans = Integer.MAX_VALUE;
        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[low] <= arr[high]) {
                ans = Math.min(ans, arr[low]);
                break;
            }
            if (arr[low] <= arr[mid]) {
                ans = Math.min(ans, arr[low]);
                low = mid + 1;
            } else {
                ans = Math.min(ans, arr[mid]);
                high = mid - 1;
            }
        }

        return ans;
    }
}

// BS7--Find out how many times array is rotated
class Solution {
    int findKRotation(int arr[], int n) {
        // code here
        int low = 0, high = n - 1;
        int ans = Integer.MAX_VALUE;
        int rotation = 0;
        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[low] <= arr[high]) {
                if (arr[low] < ans) {
                    ans = arr[low];
                    rotation = low;
                }
                break;
            }
            if (arr[low] <= arr[mid]) {
                if (arr[low] < ans) {
                    ans = arr[low];
                    rotation = low;
                }
                low = mid + 1;
            } else {
                if (arr[mid] < ans) {
                    ans = arr[low];
                    rotation = mid;
                }
                high = mid - 1;
            }
        }

        return rotation;
    }
}

// BS8--Single Elemeent in Sorted Array
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int n = nums.length;
        if (n == 1) {
            return nums[0];
        }
        if (nums[0] != nums[1]) {
            return nums[0];
        }
        if (nums[n - 1] != nums[n - 2]) {
            return nums[n - 1];
        }
        int low = 1, high = n - 2;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] != nums[mid + 1] && nums[mid] != nums[mid - 1]) {
                return nums[mid];
            } else if ((mid % 2 == 1) && (nums[mid] == nums[mid - 1]) ||
                        (mid % 2 == 0) && (nums[mid] == nums[mid + 1])) {
                    low = mid + 1;
                } else {
                    high = mid - 1;
                }
            }
            return -1;
        }
    }
}
```