// 1. Solution

//Time= O(n)

//Space=O(n)

import java.util.HashMap;

import java.util.Map;

public class Solution {

    public int[] twoSum(int[] nums, int target) {

        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {

            map.put(nums[i], i);

        }

        for (int i = 0; i < nums.length; i++) {

            int complement = target - nums[i];

            if (map.containsKey(complement) && map.get(complement) != i) {

                return new int[] { i, map.get(complement) };

            }

        }

        // In case there is no solution, we'll just return null

        return null;

    }

}

//2. Solution

//Time= O(n)

//Space=O(n)

class Solution {

    public int removeElement(int[] nums, int val) {

        // Counter for keeping track of elements other than val

        int count = 0;

        // Loop through all the elements of the array

        for (int i = 0; i < nums.length; i++) {

            // If the element is not val

            if (nums[i] != val) {

                nums[count++] = nums[i];

            }

        }

        return count;

    }

}

//3. Solution

//TC = O(N)

//Space: O(1)

class Solution {

    public int searchInsert(int[] nums, int target) {

        int start = 0, end = nums.length - 1;

        int ans = nums.length; // Default answer when target is greater than all elements

        while (start <= end) {

            int mid = start + (end - start) / 2;

            if (nums[mid] == target) {

                return mid;

            } else if (nums[mid] < target) {

                start = mid + 1;

            } else {

                ans = mid; // Update the answer to the current index

                end = mid - 1;

            }

        }

        return ans;

    }

}

//5. Solution

//Time : O(m+n)

//Space: O(1)

class Solution {

    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int i = m - 1;

        int j = n - 1;

        int k = m + n - 1;

        while (j >= 0) {

            if (i >= 0 && nums1[i] > nums2[j]) {

                nums1[k--] = nums1[i--];

            } else {

                nums1[k--] = nums2[j--];

            }

        }

    }

}

//4. Solution

//Time= O(n)

//Space=O(n)

for (int i = digits.length - 1; i >= 0; i--) {

    if (digits[i] < 9) {

        digits[i]++;

        return digits;

        // starting from extreme right--> if array[i] is less than 9 means can be added with 1

        // i.e. [ 5,8 ]-->[ 5,9 ] or

        //      [ 9,4 ]-->[ 9,5 ] or

        //      [ 6,0 ]-->[ 6,1 ]

        // and will directly return array

    }

    digits[i] = 0;

    // if array[i] is not less than 9, means it have to be 9 only then digit is changed to 0,

    // and we again revolve around loop to check for number if less than 9 or not

    // i.e. [ 5,9 ]-->[ 5,0 ]-loop->[ 6,0 ] or

    //      [ 1,9,9 ]-->[ 1,9,0 ]-loop->[ 1,0,0 ]-loop->[ 2,0,0 ]

    // and will directly return array

}

// if all number inside array are 9

// i.e. [ 9,9,9,9 ] than according to above loop it will become [ 0,0,0,0 ]

// but we have to make it like this [ 9,9,9,9 ]-->[ 1,0,0,0,0 ]

// to make like above we need to make new array of length--> n+1

// by default new array values are set to -->0 only

// thus just changed first value of array to 1 and return the array

digits = new int[digits.length + 1];

digits[0] = 1;

return digits;

6.Solution

//Time= O(n)

//Space= O(n)

class Solution {

    public boolean containsDuplicate(int[] nums) {

        HashMap<Integer,Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {

            if (map.containsKey(nums[i])) {

                return true;

            }

            map.put(nums[i],1);

        }

        return false;

    }

}

//7.Solution

//Time= O(n)

class Solution {

     public void moveZeroes(int[] nums) {

        int snowBallSize = 0;

        for (int i=0;i<nums.length;i++){

            if (nums[i]==0){

                snowBallSize++;

            }

            else if (snowBallSize > 0) {

                int t = nums[i];

                nums[i]=0;

                nums[i-snowBallSize]=t;

            }

        }

    }

}

//8. Solution

//Time= O(n)

//Space=O(1)

public class Solution {

    public int[] findErrorNums(int[] nums) {

        int[] arr = new int[nums.length + 1];

        int dup = -1, missing = 1;

        for (int i = 0; i < nums.length; i++) {

            arr[nums[i]] += 1;

        }

        for (int i = 1; i < arr.length; i++) {

            if (arr[i] == 0)

                missing = i;

            else if (arr[i] == 2)

                dup = i;

        }

        return new int[]{dup, missing};

    }

}