```
// 1. Solution

//Time= O(n)
//Space=O(n)

importjava.util.HashMap;
importjava.util.Map;

publicclassSolution {
    publicint[] twoSum(int[] nums, inttarget) {
        Map<Integer, Integer>map=newHashMap<>();
        for (inti=0; i<nums.length; i++) {
            map.put(nums[i], i);
        }
        for (inti=0; i<nums.length; i++) {
            intcomplement=target-nums[i];
            if (map.containsKey(complement) &&map.get(complement) !=i) {
                returnnewint[] { i, map.get(complement) };
            }
        }
        // In case there is no solution, we'll just return null
        returnnull;
    }
}
```

```
//2. Solution

//Time= O(n)
//Space=O(n)

classSolution {
    publicintremoveElement(int[] nums, intval) {
        // Counter for keeping track of elements other than val
        intcount=0;
        // Loop through all the elements of the array
        for (inti=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)

classSolution {
    publicintsearchInsert(int[] nums, inttarget) {
        intstart=0, end=nums.length-1;
        intans=nums.length; // Default answer when target is greater than all
elements

        while (start<=end) {
            intmid=start+ (end-start) /2;

            if (nums[mid] ==target) {
                returnmid;
            } elseif (nums[mid] <target) {
                start=mid+1;
            } else {
                ans=mid; // Update the answer to the current index
                end=mid-1;
            }
        }

        returnans;
    }
}
```

```
//5. Solution

//Time : O(m+n)
//Space: O(1)

classSolution {
    publicvoidmerge(int[] nums1, intm, int[] nums2, intn) {
        inti=m-1;
        intj=n-1;
        intk=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 (inti=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 =newint[digits.length+1];
digits[0] =1;
return digits;
```

```
6.Solution

//Time= O(n)
//Space= O(n)

classSolution {
    publicbooleancontainsDuplicate(int[] nums) {
        HashMap<Integer,Integer>map=newHashMap<>();
        for (inti=0; i<nums.length; i++) {
            if (map.containsKey(nums[i])) {
                returntrue;
            }
            map.put(nums[i],1);
        }
        returnfalse;
    }
}
```

```
//7.Solution

//Time= O(n)

classSolution {
     publicvoidmoveZeroes(int[] nums) {
        intsnowBallSize=0;
        for (inti=0;i<nums.length;i++){
            if (nums[i]==0){
                snowBallSize++;
            }
            elseif (snowBallSize>0) {
                intt=nums[i];
                nums[i]=0;
                nums[i-snowBallSize]=t;
            }
        }
    }
}
```

```
//8. Solution
//Time= O(n)
//Space=O(1)

publicclassSolution {
    publicint[] findErrorNums(int[] nums) {
        int[] arr=newint[nums.length+1];
        intdup=-1, missing=1;
        for (inti=0; i<nums.length; i++) {
            arr[nums[i]] +=1;
        }
        for (inti=1; i<arr.length; i++) {
            if (arr[i] ==0)
                missing=i;
            elseif (arr[i] ==2)
                dup=i;
        }
        returnnewint[]{dup, missing};
    }
}
```