

# Assignment 1 (Arrays) Solution

**Q1.** Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Solution:

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

```
    // O(N)
```

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

```
        int key = nums[i];
```

```
        int value = i;
```

```
        map.put(key,value);
```

```
    }
```

```
    // O(N)
```

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

```
        int x = nums[i];
```

```
        int y = target - x;
```

```
        if(map.containsKey(y)){
```

```
            int result = map.get(y);
```

```
            // checking if index of y is not same as x (x and y should be different elements)
```

```
            if(result != i){
```

```
                int[] arr = {i,result};
```

```

        return arr;
    }

}

}

int[] barr = {-1,-1};

return barr;

}

```

**Q2.** Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` in-place. The order of the elements may be changed. Then return the number of elements in `nums` which are not equal to `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Solution:

```

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

    int i = 0;

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

        if (nums[j] != val) {

            int temp = nums[i];

            nums[i] = nums[j];

            nums[j] = temp;

            i++;

        }

    }

}

```

```
    return i;
}
```

**Q3.** Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with  $O(\log n)$  runtime complexity.

Solution:

```
public int searchInsert(int[] nums, int target) {
    int low = 0;
    int high = nums.length-1;
    int mid = (low + high )/2;
    while(low<=high){
        mid = (low + high)/2;
        if(nums[mid] == target){
            return mid;
        }
        if(nums[mid] > target){
            high = mid - 1;
        }
        if(nums[mid] < target){
            low = mid + 1;
        }
    }
    if(nums[mid] > nums[nums.length - 1]){
```

```

        return nums.length;
    }
    if(nums[mid] < target){
        return ++mid;
    }
    return mid;
}

```

**Q4.** You are given a large integer represented as an integer array `digits`, where each `digits[i]` is the *i*th digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Solution:

```

public int[] plusOne(int[] digits) {

    if(digits[digits.length - 1] == 9){

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

            if(digits[i] == 9){

                digits[i] = 0;

            }else{

```

```
        digits[i]++;  
        return digits;  
    }  
}
```

```
if(digits[0] == 0){  
    int[] arr = new int[digits.length + 1];  
    arr[0] = 1;  
    for(int i = 1;i<arr.length;i++){  
        arr[i] = 0;  
    }  
    return arr;  
}
```

```
}else{  
    digits[digits.length -1]++;  
    return digits;  
}
```

```
return digits;  
}
```

**Q5.** You are given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to 0 and should be ignored. `nums2` has a length of `n`.

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--];  
        }  
    }  
}
```

Q6. Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Solution: public boolean containsDuplicate(int[] nums) {

```
    HashSet<Integer> set = new HashSet<>();
```

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

```
        if(set.contains(nums[i])){
```

```
            return true;
```

```
        }
```

```
        set.add(nums[i]);
```

```
    }
```

```
    return false;
```

```
}
```

Q7. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero elements.

Solution:

```
public void moveZeroes(int[] nums) {
```

```
    int n = nums.length;
```

```
    int index = n-1;
```

```
    for(int i = n-1;i>=0;i++){
```

```
        if(nums[i] == 0){
```

```
            if(nums[index] != 0){
```

```

        index--;

    }else{

        int temp = nums[i];

        nums[i] = nums[index];

        nums[index] = temp;

        index--;

    }

}

}

}

```

**Q8.** You have a set of integers *s*, which originally contains all the numbers from 1 to *n*. Unfortunately, due to some error, one of the numbers in *s* got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array *nums* representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

Solution:

```

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

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

    int[] result = {-1,-1};

    for(int i : nums){

        arr[i-1]++;

    }

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

```



```
    if(arr[i] == 2){  
        result[0] = i+1;  
    }  
    if(arr[i] == 0){  
        result[1] = i+1;  
    }  
}  
return result;  
}
```