Assignment_01

July 19, 2024

1 Assignment 01

1.0.1 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.

Example: Input: nums = [2,7,11,15], target = 9 Output0 [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1][

[ ]:
```java
public class Q1_TwoSum {

    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[] {-1, -1};
    }

    public static int[] twoSum2(int[] nums, int target) {
        int sum=0;
        int[] index=new int[2];
        for(int i=0;i<nums.length;i++){
            for(int j=i+1;j<nums.length;j++){
                sum=nums[i]+nums[j];
                if(target==sum){

                    index[0]=i;
                    index[1]=j;

                }
            }
        }
```

```
        return index;
}
    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
        int[] result = twoSum(nums, target);
        System.out.println(result);
    }
}
```

1.0.2 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.

Example : Input: nums = [3,2,2,3], val = 3 Output: 2, nums = [2,2,_*,_*]

[ ]:
```
public class Q2_Remove_Elements {
    public static int removeElements(int nums[], int val){
        int end=nums.length-1;
        int start=0;
        while(start<=end){
        if(nums[end]==val){
            end--;
        }
        else{
            if(nums[start]==val){
                int temp=nums[end];
                nums[end]=nums[start];
                nums[start]=temp;
                start++;
                end--;
            }
            else{
                start++;
            }
        }
        }
        return end+1;
    }
    public static void main(String[] args) {
        int[] nums = {0,1,2,2,3,0,4,2};
```

```
        int val = 2;
        int result = removeElements(nums, val);
        System.out.println(result);
    }
}
```

1.0.3 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.

Example 1: Input: nums = [1,3,5,6], target = 5

Output: 2

[ ]:
```
public class Q3_Search_Insert_Elements {
    public int searchInsert(int[] nums, int target) {
        // Last and First indexes
        int start = 0;
        int end = nums.length - 1;

         // Traverse an array
        while( start <= end ) {

            int mid = (start + end) / 2;

            //if target value found.
            if(nums[mid] == target) {
                return mid;
            }

            // If target value is greater then mid elements's value
            else if (target > nums[mid]) {
                start = mid + 1;
            }

            //otherwise target value is less,
             else {
                end = mid - 1;
            }
        }
        // Return the insertion position
        return end + 1;
    }
    public static void main(String[] args) {
        int[] nums = {1,3,5,6};
        int target = 5;
```

```
        Q3_Search_Insert_Elements obj = new Q3_Search_Insert_Elements();
        int result = obj.searchInsert(nums, target);
        System.out.println(result);
    }
}
```

1.0.4 Q4. You are given a large integer represented as an integer array digits, where each digits[i] is the ith 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.

Example 1: Input: digits = [1,2,3] Output: [1,2,4]

[ ]:
```
public class Q4_Plus_One {
    public int[] plusOne(int[] digits) {
        for (int i = digits.length - 1; i >= 0; i--) {
        if (digits[i] < 9) {
                digits[i]++;
                return digits;
        }
        digits[i] = 0;
}

digits = new int[digits.length + 1];
digits[0] = 1;
return digits;
    }
    public static void main(String[] args) {
        int[] digits = {1,2,3};
        Q4_Plus_One obj = new Q4_Plus_One();
        int[] result = obj.plusOne(digits);
        System.out.println(result);
    }
}
```

1.0.5 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.

Example 1: Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6]

[ ]:
```java
public class Q5_Merge_SortedArray {
    private static void merge(Integer[] nums1, int m, Integer[] nums2, int n) {
        // Set p1 and p2 to point to the end of their respective arrays.
        int p1 = m - 1;
        int p2 = n - 1;
        // And move p backwards through the array, each time writing
        // the smallest value pointed at by p1 or p2.
        for (int p = m + n - 1; p >= 0; p--) {
            if (p2 < 0) {
                break;
            }
            if (p1 >= 0 && nums1[p1] > nums2[p2]) {
                nums1[p] = nums1[p1--];
            } else {
                nums1[p] = nums2[p2--];
            }
        }
    }

    public static void main(String[] args) {
        Integer nums1[] = new Integer[] { 1, 2, 3, 0, 0, 0 };
        Integer nums2[] = new Integer[] { 2, 5, 6 };
        int m = 3;
        int n = 3;
        merge(nums1, m, nums2, n);
        for (int i = 0; i < m + n; i++) {
            System.out.print(nums1[i] + " ");
        }
    }
}
```

1.0.6 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.

Example 1: Input: nums = [1,2,3,1]

[ ]:
```java
import java.util.Arrays;

public class Q6_Contains_Duplicate {

    public static boolean containsDuplicate(int[] nums) {
        Arrays.sort(nums);
        for (int i = 0; i < nums.length - 1; i++) {
                if (nums[i] == nums[i+1]) {
                    return true;
                }
        }
```

```
        return false;
    }
public static void main(String[] args) {
     int nums[]={1,2,3,1};
     System.out.println(containsDuplicate(nums));
}
}
```

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

Note that you must do this in-place without making a copy of the array.

Example 1: Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]

[ ]:
```
import java.util.Arrays;

public class Q7_Move_Zeros {

     public static 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;
            }
        }
    }
public static void main(String[] args) {
    int nums[]={0,1,0,3,12};
    moveZeroes(nums);
        System.out.println(Arrays.toString(nums));
}
}
```

1.0.8 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.

Example 1: Input: nums = [1,2,2,4] Output: [2,3]

[ ]:
```java
public class Q8_Set_MissMatch {
    public static int[] findErrorNums(int[] nums) {
        int ans[] = {-1,-1};
        for(int i=0;i<nums.length;i++)
        {
            int curr=Math.abs(nums[i])-1;
            if(nums[curr]<0)
                ans[0] = curr + 1;
            else
                nums[curr] = nums[curr] * -1;
        }
        for(int i=0;i<nums.length;i++)
        {
            if(nums[i]>0)
                ans[1] = i + 1;
        }
        return ans;
    }
    public static void main(String[] args) {
        int[] arr = new int[] { 1,2,2,4 };
        int[] repeatingNum = findErrorNums(arr);


        System.out.println(repeatingNum);
    }
}
```

2 Thank You!