

## 607. Two Sum - Data structure design ☆



Description

Notes

Testcase

Judge

Design and implement a TwoSum class. It should support the following operations:

**add** and **find**.

**add** - Add the number to an internal data structure.

**find** - Find if there exists any pair of numbers which sum is equal to the value.

Have you met this question in a real interview?

### Example

```
add(1); add(3); add(5);  
find(4) // return true  
find(7) // return false
```

```

public class TwoSum {

    private List<Integer> list = null;
    private Map<Integer, Integer> map = null;
    public TwoSum() {
        list = new ArrayList<Integer>();
        map = new HashMap<Integer, Integer>();
    }

    // Add the number to an internal data structure.
    public void add(int number) {
        // Write your code here
        if (map.containsKey(number)) {
            map.put(number, map.get(number) + 1);
        } else {
            map.put(number, 1);
            list.add(number);
        }
    }

    // Find if there exists any pair of numbers which sum is equal to the value.
    public boolean find(int value) {
        // Write your code here
        for (int i = 0; i < list.size(); i++) {
            int num1 = list.get(i), num2 = value - num1;
            if ((num1 == num2 && map.get(num1) > 1) ||
                (num1 != num2 && map.containsKey(num2)))
                return true;
        }
        return false;
    }
}

// Your TwoSum object will be instantiated and called as such:
// TwoSum twoSum = new TwoSum();
// twoSum.add(number);
// twoSum.find(value);

```

## 521. Remove Duplicate Numbers in Array ☆



Description

Notes

>\_ Testcase

Judge

Given an array of integers, remove the duplicate numbers in it.

You should:

1. Do it in place in the array.
2. Move the unique numbers to the front of the array.
3. Return the total number of the unique numbers.

### Notice

You don't need to keep the original order of the integers.

Have you met this question in a real interview?

### Example

Given *nums* = [1,3,1,4,4,2], you should:

1. Move duplicate integers to the tail of *nums* => *nums* = [1,3,4,2,?,?].
2. Return the number of unique integers in *nums* => 4.

Actually we don't care about what you place in ?, we only care about the part which has no duplicate integers.

```

// O(n) time, O(n) space
public class Solution {
    /**
     * @param nums an array of integers
     * @return the number of unique integers
     */
    public int deduplication(int[] nums) {
        // Write your code here
        HashMap<Integer, Boolean> mp = new HashMap<Integer, Boolean>();
        for (int i = 0; i < nums.length; ++i)
            mp.put(nums[i], true);

        int result = 0;
        for (Map.Entry<Integer, Boolean> entry : mp.entrySet())
            nums[result++] = entry.getKey();
        return result;
    }
}

```

```

// O(nlogn) time, O(1) extra space
public class Solution {
    /**
     * @param nums an array of integers
     * @return the number of unique integers
     */
    public int deduplication(int[] nums) {
        if (nums.length == 0) {
            return 0;
        }

        Arrays.sort(nums);
        int len = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != nums[len]) {
                nums[++len] = nums[i];
            }
        }
        return len + 1;
    }
}

```

## 608. Two Sum - Input array is sorted ☆



Description

Notes

>\_ Testcase

⚖ Judge

Given an array of integers that is already *sorted in ascending order*, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

### Notice

You may assume that each input would have exactly one solution.

Have you met this question in a real interview?

### Example

Given nums = [2, 7, 11, 15], target = 9  
return [1, 2]

Tags

```

public class Solution {
    /*
    * @param nums an array of Integer
    * @param target = nums[index1] + nums[index2]
    * @return [index1 + 1, index2 + 1] (index1 < index2)
    */
    public int[] twoSum(int[] nums, int target) {
        if (nums == null || nums.length < 2) {
            return null;
        }

        int start = 0, end = nums.length - 1;
        while (start < end) {
            if (nums[start] + nums[end] == target) {
                int[] pair = new int[2];
                pair[0] = start + 1;
                pair[1] = end + 1;
                return pair;
            }
            if (nums[start] + nums[end] < target) {
                start++;
            } else {
                end--;
            }
        }

        return null;
    }
}

```

## 609. Two Sum - Less than or equal to target ☆

[Description](#)[Notes](#)[Testcase](#)[Judge](#)

Given an array of integers, find how many pairs in the array such that their sum is **less than or equal to** a specific target number. Please return the number of pairs.

Have you met this question in a real interview?

### Example

Given nums = **[2, 7, 11, 15]**, target = **24**.

Return **5**.

$2 + 7 < 24$

$2 + 11 < 24$

$2 + 15 < 24$

$7 + 11 < 24$

$7 + 15 < 25$

Tags ▼

```

public class Solution {
    /**
     * @param nums an array of integer
     * @param target an integer
     * @return an integer
     */
    public int twoSum5(int[] nums, int target) {
        // Write your code here
        if (nums == null || nums.length < 2)
            return 0;

        Arrays.sort(nums);
        int cnt = 0;
        int left = 0, right = nums.length - 1;
        while (left < right) {
            int v = nums[left] + nums[right];
            if (v > target) {
                right--;
            } else {
                cnt += right - left;
                left++;
            }
        }
        return cnt;
    }
}

```



## 587. Two Sum - Unique pairs ☆

[Description](#)[Notes](#)[\\_ Testcase](#)[Judge](#)

Given an array of integers, find how many **unique pairs** in the array such that their sum is equal to a specific target number. Please return the number of pairs.

Have you met this question in a real interview?

### Example

Given nums = **[1,1,2,45,46,46]** , target = **47**  
return **2**

1 + 46 = 47

2 + 45 = 47

Tags ▾



```

public class Solution {
    /**
     * @param nums an array of integer
     * @param target an integer
     * @return an integer
     */
    public int twoSum6(int[] nums, int target) {
        // Write your code here
        if (nums == null || nums.length < 2)
            return 0;

        Arrays.sort(nums);
        int cnt = 0;
        int left = 0, right = nums.length - 1;
        while (left < right) {
            int v = nums[left] + nums[right];
            if (v == target) {
                cnt++;
                left++;
                right--;
                while (left < right && nums[right] == nums[right + 1])
                    right--;
                while (left < right && nums[left] == nums[left - 1])
                    left++;
            } else if (v > target) {
                right--;
            } else {
                left++;
            }
        }
        return cnt;
    }
}

```

## 533. Two Sum - Closest to target ☆



Description

Notes

Testcase

Judge

Given an array `nums` of  $n$  integers, find two integers in `nums` such that the sum is closest to a given number, `target`.

Return the difference between the sum of the two integers and the target.

Have you met this question in a real interview?

### Example

Given array `nums` = `[-1, 2, 1, -4]`, and `target` = `4`.

The minimum difference is `1`. ( $4 - (2 + 1) = 1$ ).

...



```

public class Solution {
    /**
     * @param nums an integer array
     * @param target an integer
     * @return the difference between the sum and the target
     */
    public int twoSumClosest(int[] nums, int target) {
        if (nums == null || nums.length < 2) {
            return -1;
        }

        Arrays.sort(nums);

        int left = 0, right = nums.length - 1;
        int diff = Integer.MAX_VALUE;

        while (left < right) {
            if (nums[left] + nums[right] < target) {
                diff = Math.min(diff, target - nums[left] - nums[right]);
                left++;
            } else {
                diff = Math.min(diff, nums[left] + nums[right] - target);
                right--;
            }
        }

        return diff;
    }
}

```

## 148. Sort Colors ☆



Description

Notes

Testcase

Judge

Given an array with  $n$  objects colored *red*, *white* or *blue*, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers **0**, **1**, and **2** to represent the color red, white, and blue respectively.

### Notice

You are not suppose to use the library's sort function for this problem.  
You should do it in-place (sort numbers in the original array).

Have you met this question in a real interview?

### Example

Given **[1, 0, 1, 2]**, sort it in-place to **[0, 1, 1, 2]**.

**Challenge** ▼

```

public class Solution {
    public void sortColors(int[] a) {
        if (a == null || a.length <= 1) {
            return;
        }


        int pl = 0;
        int pr = a.length - 1;
        int i = 0;
        while (i <= pr) {
            if (a[i] == 0) {
                swap(a, pl, i);
                pl++;
                i++;
            } else if (a[i] == 1) {
                i++;
            } else {
                swap(a, pr, i);
                pr--;
            }
        }
    }

    private void swap(int[] a, int i, int j) {
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }
}

```

## 143. Sort Colors II ☆

 Description

 Notes

 Testcase

 Judge

Given an array of  $n$  objects with  $k$  different colors (numbered from 1 to  $k$ ), sort them so that objects of the same color are adjacent, with the colors in the order 1, 2, ...  $k$ .

### Notice

You are not suppose to use the library's sort function for this problem.

$k \leq n$

Have you met this question in a real interview?

### Example

Given colors= [3, 2, 2, 1, 4] ,  $k=4$  , your code should sort colors in-place to [1, 2, 2, 3, 4] .

### Challenge

// version 1:  $O(n \log k)$ , the best algorithm based on comparing

class Solution {

/\*\*

\* @param colors: A list of integer

\* @param k: An integer

\* @return: nothing

\*/

public void sortColors2(int[] colors, int k) {

if (colors == null || colors.length == 0) {

return;

}

rainbowSort(colors, 0, colors.length - 1, 1, k);

}

public void rainbowSort(int[] colors,

int left,

int right,

int colorFrom,

int colorTo) {

if (colorFrom == colorTo) {

return;

}

if (left >= right) {

return;

}

int colorMid = (colorFrom + colorTo) / 2;

int l = left, r = right;

while (l <= r) {

while (l <= r && colors[l] <= colorMid) {

l++;

}

while (l <= r && colors[r] > colorMid) {

r--;

}

if (l <= r) {

int temp = colors[l];

colors[l] = colors[r];

colors[r] = temp;

l++;

r--;

}

}



```
rainbowSort(colors, left, r, colorFrom, colorMid);  
rainbowSort(colors, l, right, colorMid + 1, colorTo);  
}  
}
```

## 57. 3Sum ☆



Description

Notes

Testcase

Judge

Given an array  $S$  of  $n$  integers, are there elements  $a, b, c$  in  $S$  such that  $a + b + c = 0$ ? Find all unique triplets in the array which gives the sum of zero.

### Notice

Elements in a triplet  $(a, b, c)$  must be in non-descending order. (ie,  $a \leq b \leq c$ )

The solution set must not contain duplicate triplets.

Have you met this question in a real interview?

### Example

For example, given array  $S = \{-1, 0, 1, 2, -1, -4\}$ , A solution set is:

$(-1, 0, 1)$   
 $(-1, -1, 2)$

```

public class Solution {
    /**
     * @param nums : Give an array numbers of n integer
     * @return : Find all unique triplets in the array which gives the sum of zero.
     */
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> results = new ArrayList<>();

        if (nums == null || nums.length < 3) {
            return results;
        }

        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 2; i++) {
            // skip duplicate triples with the same first numebr
            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }

            int left = i + 1, right = nums.length - 1;
            int target = -nums[i];

            twoSum(nums, left, right, target, results);
        }

        return results;
    }

    public void twoSum(int[] nums,
        int left,
        int right,
        int target,
        List<List<Integer>> results) {
        while (left < right) {
            if (nums[left] + nums[right] == target) {
                ArrayList<Integer> triple = new ArrayList<>();
                triple.add(-target);
                triple.add(nums[left]);
                triple.add(nums[right]);
                results.add(triple);

                left++;
                right--;
                // skip duplicate pairs with the same left
            }
        }
    }
}


```

```
while (left < right && nums[left] == nums[left - 1]) {  
    left++;  
}  
// skip duplicate pairs with the same right  
while (left < right && nums[right] == nums[right + 1]) {  
    right--;  
}  
} else if (nums[left] + nums[right] < target) {  
    left++;  
} else {  
    right--;  
}  
}  
}  
}
```

## 31. Partition Array

### 31. Partition Array ☆

 Description

 Notes

 \_ Testcase

 Judge

Given an array `nums` of integers and an int `k`, partition the array (i.e move the elements in "nums") such that:

- All elements  $< k$  are moved to the *left*
- All elements  $\geq k$  are moved to the *right*

Return the partitioning index, i.e the first index  $i$   $nums[i] \geq k$ .

#### Notice

You should do really partition in array `nums` instead of just counting the numbers of integers smaller than `k`.

If all elements in `nums` are smaller than `k`, then return `nums.length`

Have you met this question in a real interview?

### Example

If `nums = [3, 2, 2, 1]` and `k=2`, a valid answer is `1`.

```

public class Solution {
    /**
     * @param nums: The integer array you should partition
     * @param k: As description
     * return: The index after partition
     */
    public int partitionArray(int[] nums, int k) {
        if(nums == null || nums.length == 0){
            return 0;
        }

        int left = 0, right = nums.length - 1;
        while (left <= right) {

            while (left <= right && nums[left] < k) {
                left++;
            }

            while (left <= right && nums[right] >= k) {
                right--;
            }

            if (left <= right) {
                int temp = nums[left];
                nums[left] = nums[right];
                nums[right] = temp;

                left++;
                right--;
            }
        }
        return left;
    }
}

```

