Problem | Editorial | Submissions | Comments

Kth Smallest

Difficulty: **Medium** Accuracy: **35.17%** Submissions: **739K+** Points: **4** Average Time: **25m**

Given an integer array **arr[]** and an integer **k**, your task is to find and return the **k^{th} smallest** element in the given array.

Note: The kth smallest element is determined based on the sorted order of the array.

Examples :

Input: arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10], k = 4
Output: 5
Explanation: 4th smallest element in the given array is 5.

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3
Output: 7
Explanation: 3rd smallest element in the given array is 7.

Constraints:

$1 \leq arr.size() \leq 10^5$

$1 \leq arr[i] \leq 10^5$

$1 \leq k \leq arr.size()$

Java (21) Start Timer

```java
class Solution {
    public int kthSmallest(int[] arr, int k) {
        Arrays.sort(arr);
        return arr[k - 1];
    }
}
```

Examples :

Input: k = 2, arr[] = [1, 5, 8, 10]
Output: 5
Explanation: The array can be modified as [1+k, 5-k, 8-k, 10-k] = [3, 3, 6, 8]. The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = [3, 9, 12, 16, 20]
Output: 11
Explanation: The array can be modified as [3+k, 9+k, 12-k, 16-k, 20-k] = [6, 12, 9, 13, 17]. The difference between the largest and the smallest is 17-6 = 11.

Constraints

$1 \leq k \leq 10^7$

$1 \leq n \leq 10^5$

$1 \leq arr[i] \leq 10^7$

Try more examples

Expected Complexities

```java
class Solution {
    public int getMinDiff(int[] arr, int k) {
        int n = arr.length;

        Arrays.sort(arr);

        int ans = arr[n - 1] - arr[0];

        int smallest = arr[0] + k;
        int largest = arr[n - 1] - k;

        for (int i = 1; i < n; i++) {

            if (arr[i] - k < 0)
                continue;

            int minHeight = Math.min(smallest, arr[i] - k);
            int maxHeight = Math.max(largest, arr[i - 1] + k);

            ans = Math.min(ans, maxHeight - minHeight);
        }

        return ans;
    }
}
```

Problem | Editorial | Submissions | Comments

Minimum Jumps

Difficulty: **Medium** Accuracy: **11.91%** Submissions: **1.1M** Points: **4**

You are given an array **arr[]** of non-negative numbers. Each number tells you the **maximum number of steps** you can jump forward from that position.

For example:

- If **arr[i] = 3**, you can jump to index **i + 1**, **i + 2**, or **i + 3** from position **i**.
- If **arr[i] = 0**, you **cannot jump forward** from that position.

Your task is to find the **minimum number of jumps** needed to move from the **first** position in the array to the **last** position.

Note: Return **-1** if you can't reach the end of the array.

Examples :

Input: arr[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]
Output: 3
Explanation: First jump from 1st element to 2nd element with value 3. From here we jump to 5th element with value 9, and from here we will jump to the last.

Input: arr = [1, 4, 3, 2, 6, 7]
Output: 2
Explanation: First we jump from the 1st to 2nd element and then jump to the last element.

Input: arr = [0, 10, 20]

Java (21) | Start Timer

```java
class Solution {
    public int minJumps(int[] arr) {
        int n = arr.length;

        if (n <= 1)
            return 0;

        if (arr[0] == 0)
            return -1;

        int jumps = 1;
        int maxReach = arr[0];
        int steps = arr[0];

        for (int i = 1; i < n; i++) {

            if (i == n - 1)
                return jumps;

            maxReach = Math.max(maxReach, i + arr[i]);
            steps--;

            if (steps == 0) {
                jumps++;

                if (i >= maxReach)
                    return -1;

                steps = maxReach - i;
            }
        }

        return -1;
    }
}
```

Problem List

Submit

Description | Editorial | Solutions | Submissions

287. Find the Duplicate Number

Medium | Topics | Companies

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

Example 1:

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

Example 2:

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

Example 3:

```
Input: nums = [3,3,3,3,3]
Output: 3
```

Code

C++ • Auto

```cpp
import java.util.Arrays;

class Solution {
    public int kthSmallest(int[] arr, int k) {
        Arrays.sort(arr);
        return arr[k - 1];
    }
}
```

Description | Accepted | Editorial | Solutions | Submissions

287. Find the Duplicate Number

Solved

Medium | Topics | Companies

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

Example 1:

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

Example 2:

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

Example 3:

```
Input: nums = [3,3,3,3,3]
Output: 3
```

Constraints:

- $1 <= n <= 10^5$

25.3K | 494 | 133 Online

Code

Java | Auto

```java
class Solution {
    public int findDuplicate(int[] nums) {

        int slow = nums[0];
        int fast = nums[0];

        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

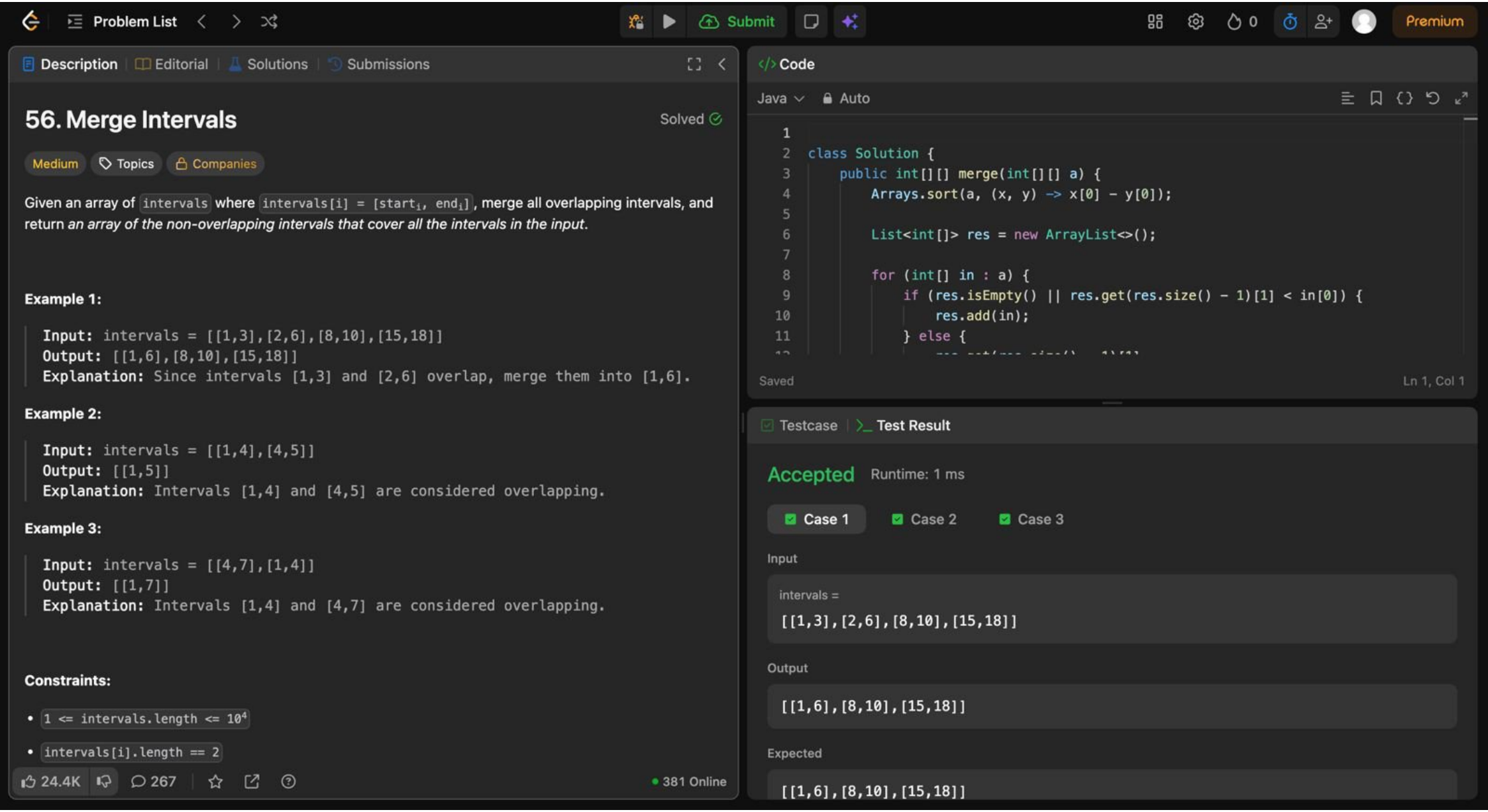
nums =
[1,3,4,2,2]

Output

2

Expected

2


Problem List
Submit
0
Premium
Description
Editorial
Solutions
Submissions
56. Merge Intervals
Solved
Medium
Topics
Companies
Given an array of intervals where intervals[i] = [start_i, end_i], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.
Example 1:
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
Example 2:
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
Example 3:
Input: intervals = [[4,7],[1,4]]
Output: [[1,7]]
Explanation: Intervals [1,4] and [4,7] are considered overlapping.
Constraints:
1 <= intervals.length <= 10^4
intervals[i].length == 2
24.4K
267
381 Online
Code
Java
Auto
class Solution {
public int[][] merge(int[][] a) {
Arrays.sort(a, (x, y) -> x[0] - y[0]);
List<int[]> res = new ArrayList<>();
for (int[] in : a) {
if (res.isEmpty() || res.get(res.size() - 1)[1] < in[0]) {
res.add(in);
} else {
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 1 ms
Case 1
Case 2
Case 3
Input
intervals =
[[1,3],[2,6],[8,10],[15,18]]
Output
[[1,6],[8,10],[15,18]]
Expected
[[1,6],[8,10],[15,18]]

Problem | Editorial | Submissions | Comments

Factorials of large numbers

Difficulty: **Medium** Accuracy: **36.57%** Submissions: **178K+** Points: **4** Average Time: **20m**

Given an integer **n,** find its factorial. Return a list of integers denoting the digits that make up the factorial of n.

Examples:

Input: n = 5
Output: [1, 2, 0]
Explanation: 5! = 1*2*3*4*5 = 120

Input: n = 10
Output: [3, 6, 2, 8, 8, 0, 0]
Explanation: 10! = 1*2*3*4*5*6*7*8*9*10 = 3628800

Input: n = 1
Output: [1]
Explanation: 1! = 1

Constraints:
$1 \leq n \leq 10^3$

Try more examples

Expected Complexities

Java (21) Start Timer

```java
// User function Template for Java

class Solution {
    public static ArrayList<Integer> factorial(int n) {

        ArrayList<Integer> res = new ArrayList<>();
        res.add(1);

        for (int x = 2; x <= n; x++) {
            int carry = 0;

            for (int i = 0; i < res.size(); i++) {
                int val = res.get(i) * x + carry;
                res.set(i, val % 10);
                carry = val / 10;
            }

            while (carry > 0) {
                res.add(carry % 10);
                carry /= 10;
            }
        }

        Collections.reverse(res);
        return res;
    }
}

```

Common in 3 Sorted Arrays

Difficulty: **Easy** Accuracy: **22.16%** Submissions: **440K+** Points: **2**

Given three sorted arrays in **non-decreasing** order, print all common elements in **non-decreasing** order across these arrays. If there are no such elements return an empty array. In this case, the output will be -1.

Note: can you handle the duplicates without using any additional Data Structure?

Examples :

Input: arr1 = [1, 5, 10, 20, 40, 80] , arr2 = [6, 7, 20, 80, 100] , arr3 = [3, 4, 15, 20, 30, 70, 80, 120]
Output: [20, 80]
Explanation: 20 and 80 are the only common elements in arr1, arr2 and arr3.

Input: arr1 = [1, 2, 3, 4, 5] , arr2 = [6, 7] , arr3 = [8,9,10]
Output: [-1]
Explanation: There are no common elements in arr1, arr2 and arr3.

Input: arr1 = [1, 1, 1, 2, 2, 2], arr2 = [1, 1, 2, 2, 2], arr3 = [1, 1, 1, 1, 2, 2, 2, 2]
Output: [1, 2]
Explanation: We do not need to consider duplicates

Constraints:

$1 <= arr1.size(), arr2.size(), arr3.size() <= 10^5$

$-10^5 <= arr1_i , arr2_i , 1arr3_i <= 10^5$

```java
// User function Template for Java
import java.util.*;

class Solution {

    public List<Integer> commonElements(List<Integer> arr1, List<Integer> arr2,
                                        List<Integer> arr3) {

        List<Integer> res = new ArrayList<>();
        int i = 0, j = 0, k = 0;

        while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {

            int a = arr1.get(i);
            int b = arr2.get(j);
            int c = arr3.get(k);

            if (a == b && b == c) {
                if (res.isEmpty() || res.get(res.size() - 1) != a)
                    res.add(a);
                i++; j++; k++;
            }
            else if (a < b) i++;
            else if (b < c) j++;
            else k++;
        }

        if (res.isEmpty())
            res.add(-1);

        return res;
    }
}
```

Problem | Editorial | Submissions | Comments

Array Subset

Difficulty: **Basic** Accuracy: **44.05%** Submissions: **521K+** Points: **1** Average Time: **20m**

Given two arrays **a[]** and **b[]**, your task is to determine whether **b[]** is a subset of **a[]**.

Examples:

Input: a[] = [11, 7, 1, 13, 21, 3, 7, 3], b[] = [11, 3, 7, 1, 7]
Output: true
Explanation: b[] is a subset of a[]

Input: a[] = [1, 2, 3, 4, 4, 5, 6], b[] = [1, 2, 4]
Output: true
Explanation: b[] is a subset of a[]

Input: a[] = [10, 5, 2, 23, 19], b[] = [19, 5, 3]
Output: false
Explanation: b[] is not a subset of a[]

Constraints:

$1 <= a.size(), b.size() <= 10^5$

$1 <= a[i], b[j] <= 10^6$

Try more examples

Expected Complexities

Java (21) | Start Timer

```java
class Solution {
    public boolean isSubset(int a[], int b[]) {

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


        for (int x : a) {
            map.put(x, map.getOrDefault(x, 0) + 1);
        }


        for (int x : b) {
            if (!map.containsKey(x) || map.get(x) == 0)
                return false;
            map.put(x, map.get(x) - 1);
        }

        return true;
    }
}
```

Custom Input | Compile & Run | Submit

Triplet Sum in Array

Difficulty: **Medium** Accuracy: **35.0%** Submissions: **362K+** Points: **4** Average Time: **15m**

Given an array **arr[]** and an integer **target**, determine if there exists a triplet in the array whose sum equals the given **target**.

Return **true** if such a triplet exists, otherwise, return **false**.

Examples:

Input: arr[] = [1, 4, 45, 6, 10, 8], target = 13
Output: true
Explanation: The triplet {1, 4, 8} sums up to 13.

Input: arr[] = [1, 2, 4, 3, 6, 7], target = 10
Output: true
Explanation: The triplets {1, 3, 6} and {1, 2, 7} both sum to 10.

Input: arr[] = [40, 20, 10, 3, 6, 7], target = 24
Output: false
Explanation: No triplet in the array sums to 24.

Constraints:

$3 \leq arr.size() \leq 5*10^3$

$0 \leq arr[i], target \leq 10^5$

Java (21) Start Timer

```java
import java.util.Arrays;

public boolean hasTripletSum(int arr[], int target) {
    Arrays.sort(arr);
    int n = arr.length;

    for (int i = 0; i < n - 2; i++) {
        int l = i + 1, r = n - 1;

        while (l < r) {
            int sum = arr[i] + arr[l] + arr[r];

            if (sum == target) return true;
            else if (sum < target) l++;
            else r--;
        }
    }
    return false;
}
```

Problem | Editorial | Submissions | Comments

Input: arr[] = [1, 2, 3, 4]
Output: 0
Explanation: We cannot trap water as there is no height bound on both sides.

Input: arr[] = [2, 1, 5, 3, 1, 0, 4]
Output: 9
Explanation: Total water trapped = 0 + 1 + 0 + 1 + 3 + 4 + 0 = 9 units.

Constraints:

$1 \leq arr.size() \leq 10^5$

$0 \leq arr[i] \leq 10^3$

Try more examples

Expected Complexities

Company Tags

Flipkart | Amazon | Microsoft | Google | Goldman Sachs

Topic Tags

Related Interview Experiences

Related Articles

Java (21) | Start Timer

```java
class Solution {
    public int maxWater(int arr[]) {
        int left = 0, right = arr.length - 1;
        int leftMax = 0, rightMax = 0, water = 0;

        while (left < right) {
            if (arr[left] <= arr[right]) {
                if (arr[left] >= leftMax)
                    leftMax = arr[left];
                else
                    water += leftMax - arr[left];
                left++;
            } else {
                if (arr[right] >= rightMax)
                    rightMax = arr[right];
                else
                    water += rightMax - arr[right];
                right--;
            }
        }
        return water;
    }
}
```

Custom Input | Compile & Run | Submit