

Experiment List for Programming Ability and Logic Building – 1

Given an integer array arr[] and an integer k, your task is to find and return the kth 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:

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

$1 \leq \text{arr}[i] \leq 10^5$

$1 \leq k \leq \text{arr.size()}$

1.

Solution-

```
Java (21) Start Timer
1- class Solution {
2-     public int kthSmallest(int[] arr, int k) {
3-         // Code here
4-         Arrays.sort(arr);
5-         return arr[k - 1];
6-     }
7- }
```

Test Cases Passed
1121 / 1121

Attempts : Correct / Total
1 / 1

Accuracy : 100%

Points Scored
4 / 4

Your Total Score: 4

Solve Next
Smallest Positive Missing, Valid Pair Sum, Optimal Array

Stay Ahead With:
Build 21 Projects in 21 Days
Build real-world ML, Deep Learning & Gen AI projects
Register Now

Custom Input, Compile & Run, Submit

2.

Given an array arr[] denoting heights of n towers and a positive integer k.

For each tower, you must perform exactly one of the following operations exactly once.

Increase the height of the tower by k

Decrease the height of the tower by k

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem here.

Note: It is compulsory to increase or decrease the height by k for each tower. After the operation, the resultant array should not contain any negative integers.

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 $17-6 = 11$.

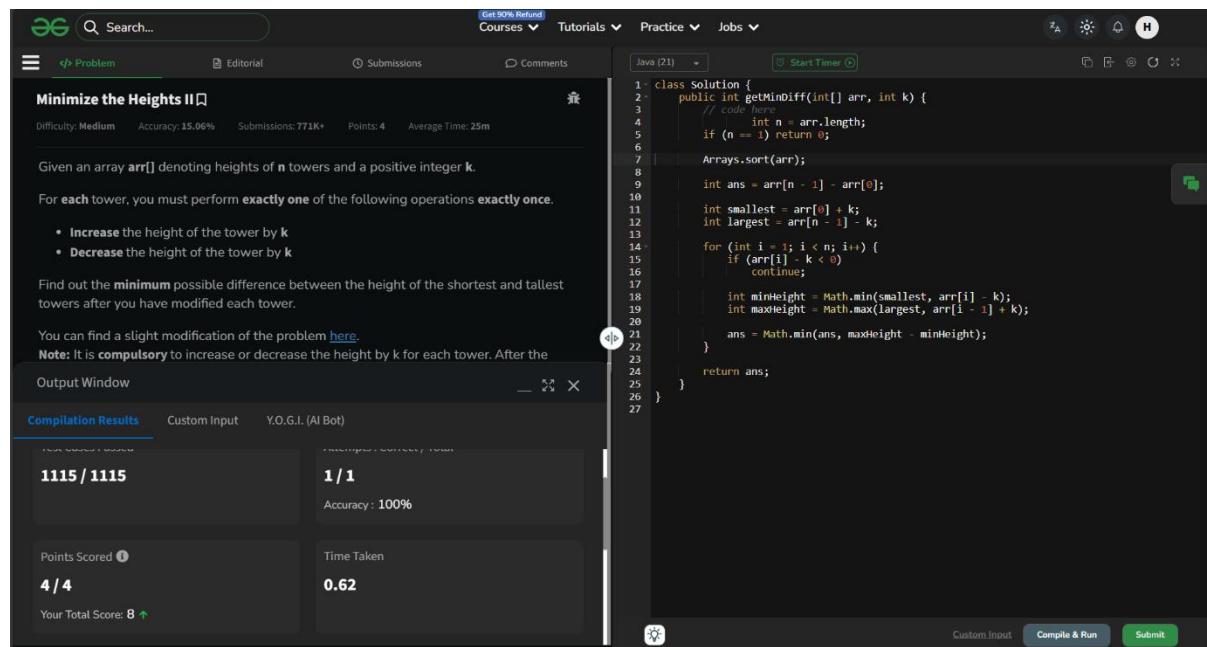
Constraints

$1 \leq k \leq 10^7$

$1 \leq n \leq 10^5$

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

Solution –



The screenshot shows the LeetCode platform interface for the 'Minimize the Heights II' problem. The problem details are as follows:

- Difficulty:** Medium
- Accuracy:** 15.06%
- Submissions:** 771K+
- Points:** 4
- Average Time:** 25m

The problem statement is identical to the one provided in the text above. The Java code solution is as follows:

```

1. class Solution {
2.     public int getMinDiff(int[] arr, int k) {
3.         // code here
4.         int n = arr.length;
5.         if (n == 1) return 0;
6.
7.         Arrays.sort(arr);
8.
9.         int ans = arr[n - 1] - arr[0];
10.
11.        int smallest = arr[0] + k;
12.        int largest = arr[n - 1] - k;
13.
14.        for (int i = 1; i < n; i++) {
15.            if (arr[i] - k < 0)
16.                continue;
17.
18.            int minHeight = Math.min(smallest, arr[i] - k);
19.            int maxHeight = Math.max(largest, arr[i - 1] + k);
20.
21.            ans = Math.min(ans, maxHeight - minHeight);
22.
23.        }
24.
25.        return ans;
26.    }
27. }

```

The 'Compilation Results' section shows a success message: **1115 / 1115** and **1 / 1**, with an accuracy of **100%**. The 'Time Taken' is listed as **0.62**.

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.

3.

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]

Output: -1

Explanation: We cannot go anywhere from the 1st element.

Constraints:

$2 \leq \text{arr.size()} \leq 10^5$

$0 \leq \text{arr}[i] \leq 10^5$

Solution -

The screenshot shows a LeetCode problem page for "Jump Game". The code editor contains the following Java solution:

```
1- class Solution {
2-     public int minJumps(int[] arr) {
3-         // code here
4-         int n = arr.length;
5-
6-         if (n <= 1)
7-             return 0;
8-         if (arr[0] == 0)
9-             return -1;
10-
11        int maxReach = arr[0];
12        int steps = arr[0];
13        int jumps = 1;
14-
15        for (int i = 1; i < n; i++) {
16            if (i == n - 1)
17                return jumps;
18            maxReach = Math.max(maxReach, i + arr[i]);
19            steps--;
20        }
21-
22        if (steps == 0) {
23            jumps++;
24            if (i >= maxReach)
25                return -1;
26            steps = maxReach - i;
27        }
28-
29    }
30}
31
32
33
34
35 }
```

The page displays the following statistics:

- Test Cases Passed: 1120 / 1120
- Attempts : Correct / Total: 1 / 1
- Accuracy: 100%
- Points Scored: 4 / 4
- Time Taken: 0.68
- Your Total Score: 12

Below the stats, there are buttons for "Maximum Index", "Jump Game", and "Wine Buying and Selling". A sidebar on the left says "Stay Ahead With:" and lists "Build 21 Projects in 21 Days" and "Build real-world ML, Deep Learning & Gen AI projects".

4.

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: $1! = 1$

Solution -

The screenshot shows a LeetCode problem page for "Factorial of a Number". The problem details are as follows:

- Problem:** Problem 6. Factorial of a Number
- Language:** Java
- Time:** 5m 32s
- Status:** Solved Successfully
- Attempts:** 1 / 1
- Accuracy:** 100%
- Score:** 4 / 4
- Total Score:** 16
- Time Taken:** 0.58

The code editor contains the following Java code:

```
1 // User function Template for Java
2
3 class Solution {
4     public static ArrayList<Integer> factorial(int n) {
5         ArrayList<Integer> ans = new ArrayList<Integer>();
6         ans.add(1);
7
8         for (int num = 2; num <= n; num++) {
9             int carry = 0;
10
11             for (int i = 0; i < ans.size(); i++) {
12                 int value = ans.get(i) * num + carry;
13
14                 ans.set(i, value % 10);
15                 carry = value / 10;
16             }
17
18             while (carry > 0) {
19                 ans.add(carry % 10);
20                 carry = carry / 10;
21             }
22
23         }
24
25         Collections.reverse(ans);
26         return ans;
27     }
28 }
29 }
```

At the bottom right, there are buttons for "Custom Input", "Compile & Run", and "Submit".

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 $a[]$

5.

Solution-

The screenshot shows a LeetCode problem interface. The top navigation bar includes 'Get 90% Refund', 'Courses', 'Tutorials', 'Practice', and 'Jobs'. The main area displays a Java solution for the 'isSubset' problem. The code uses a hashmap to store the frequency of each element in array 'a'. It then iterates through array 'b', checking if each element exists in the map or if its complement (target - current sum) does. If either condition fails, it returns false. Otherwise, it updates the map and continues. The solution is marked as successful with 1114/1114 test cases passed and 100% accuracy. The time taken was 0.58 seconds.

```
1 class Solution {
2     public boolean isSubset(int a[], int b[]) {
3         // Your code here
4         HashMap<Integer, Integer> map = new HashMap<>();
5
6         for (int num : a) {
7             map.put(num, map.getOrDefault(num, 0) + 1);
8         }
9
10        for (int num : b) {
11            if (!map.containsKey(num) || map.get(num) == 0) {
12                return false;
13            }
14            map.put(num, map.get(num) - 1);
15        }
16
17        return true;
18    }
19
20 }
```