

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 `k`th smallest element in the given array.

Note: The `k`th 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-

The screenshot displays a coding platform interface with the following components:

- Header:** Includes a search bar, navigation links for Courses, Tutorials, Practice, and Jobs, and user profile icons.
- Problem Window:** Shows the problem name 'kthSmallest' and a 'Start Timer' button.
- Compilation Results:** A green banner indicates 'Problem Solved Successfully' with a checkmark.
- Test Cases Passed:** A box shows '1121 / 1121'.
- Attempts:** A box shows '1 / 1' with 'Correct / Total' and 'Accuracy: 100%'.
- Points Scored:** A box shows '4 / 4' with 'Your Total Score: 4' and an upward arrow.
- Time Taken:** A box shows '0.74'.
- Solve Next:** A row of buttons for 'Smallest Positive Missing', 'Valid Pair Sum', and 'Optimal Array'.
- Stay Ahead With:** A promotional banner for 'Build 21 Projects in 21 Days' with a 'Register Now' link.
- Code Editor:** Displays a Java solution for the 'kthSmallest' problem. The code is as follows:

```
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 }
8
```
- Footer:** Includes a 'Custom Input' field and 'Compile & Run' and 'Submit' buttons.

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 displays a coding platform interface for the problem "Minimize the Heights II". The problem description on the left states: 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 by `k` or decrease the height by `k`. The goal is to find the minimum possible difference between the height of the shortest and tallest towers after modification. A note specifies that it is compulsory to increase or decrease the height by `k` for each tower, and the resultant array must not contain negative integers.

The solution is implemented in Java in the editor on the right. The code defines a `Solution` class with a `getMinDiff` method. It sorts the array and then iterates through it, calculating the minimum and maximum possible heights for each tower after the operation. The final result is the minimum difference between the maximum and minimum heights.

```
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             ans = Math.min(ans, maxHeight - minHeight);
21         }
22         return ans;
23     }
24 }
25
26
27
```

The bottom left of the interface shows the "Compilation Results" section, indicating that the solution was successful with an accuracy of 100% and a time taken of 0.62 seconds. The overall score is 4/4 points.

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 displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.J. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Test cases passed are 1120/1120, attempts are 1/1, accuracy is 100%, points scored are 4/4, and time taken is 0.68. Below this, 'Solve Next' buttons for 'Maximum Index', 'Jump Game', and 'Wine Buying and Selling' are visible. At the bottom left, a banner promotes 'Build 21 Projects in 21 Days'. The main area on the right shows a Java code editor with a solution for the 'Jump Game' problem. The code defines a 'Solution' class with a 'minJumps' method that uses a greedy approach to find the minimum number of jumps. The code is as follows:

```
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
17             if (i == n - 1)
18                 return jumps;
19
20             maxReach = Math.max(maxReach, i + arr[i]);
21             steps--;
22
23             if (steps == 0) {
24                 jumps++;
25
26                 if (i >= maxReach)
27                     return -1;
28
29                 steps = maxReach - i;
30             }
31         }
32
33         return -1;
34     }
35 }
```

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

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 displays a coding platform interface with a dark theme. On the left, a sidebar shows the 'Problem' tab selected, with options for 'Editorial', 'Submissions', and 'Comments'. The main area on the left contains a 'Problem Solved Successfully' message with a green checkmark. Below this, statistics are shown: 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.58'. A 'Solve Next' section offers links to 'Large Factorial', 'Number following a pattern', and 'Rank The Permutations'. At the bottom left, a banner promotes 'Build 21 Projects in 21 Days'.

The right side of the interface shows a code editor with a Java solution. The code is as follows:

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

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-

```
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        return true;
17    }
18 }
19
20
21
22
23
```