

# EXPERIMENT LIST FOR PROGRAMMING ABILITY AND LOGIC BUILDING – 1

(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:

$1 \leq \text{arr.size()} \leq 105$

$1 \leq \text{arr}[i] \leq 105$

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

The screenshot shows a programming environment with the following details:

- Problem Solved Successfully**: The user has solved the problem successfully.
- Test Cases Passed**: 1121 / 1121
- Attempts : Correct / Total**: 2 / 2
- Accuracy**: 100%
- Time Taken**: 0.81
- Note**: You get marks only for the first correct submission if you solve the problem without viewing the full solution.
- Solve Next**: Options include Smallest Positive Missing, Valid Pair Sum, and Optimal Array.
- Code Area**:

```
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 }
```
- Toolbars**: Includes tabs for Courses, Tutorials, Practice, and Jobs, along with various icons for file operations.

(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

$1 \leq k \leq 107$

$1 \leq n \leq 105$

$1 \leq arr[i] \leq 107$

The screenshot shows a browser window for the GeeksforGeeks 'Minimize the Heights II' practice problem. The page includes navigation tabs for Courses, Tutorials, Practice, and Jobs. The main area displays Java code for the solution, which sorts the array and calculates the minimum difference. The 'Compilation Results' section shows 'Custom Input' and 'Y.O.G.I. (AI Bot)' results. The 'Problem Solved Successfully' section shows 1115/1115 test cases passed, 1/1 attempts correct, 100% accuracy, 4/4 points scored, and a total score of 8. The 'Time Taken' is 0.56 seconds. At the bottom, there are links to 'Minimum Jumps', 'A difference of values and indexes', and 'Minimize the Heights I'. A 'Solve Next' button is also visible.

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

(3) 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]

Output: -1

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

Constraints:

$2 \leq \text{arr.size()} \leq 105$

$0 \leq \text{arr}[i] \leq 105$

The screenshot shows a Java code editor with the following code:

```
1+ class Solution {
2+     public int minJumps(int[] arr) {
3+         // code here
4+         int n = arr.length;
5+         if (n == 1)
6+             return 0;
7+
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+             maxReach = Math.max(maxReach, i + arr[i]);
20+
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+         }
34+         return jumps;
35+     }
}
```

The interface displays the following statistics:

- Problem Solved Successfully: 1120 / 1120
- Attempts: Correct / Total: 1 / 1
- Accuracy: 100%
- Points Scored: 4 / 4
- Time Taken: 0.62
- Total Score: 12

Below the stats, there are links for "Solve Next" (Maximum Index, Jump Game, Wine Buying and Selling) and "Stay Ahead With:".

(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:  $1! = 1$

The screenshot shows a web browser window for the GeeksforGeeks Practice platform. The URL is https://www.geeksforgeeks.org/problems/factorials-of-large-numbers2508/1. The page displays a Java code editor with a solved problem. The code implements a factorial function that returns a list of digits. It uses a for loop to calculate the product of numbers from 2 to n, adding each result to a list and keeping track of a carry value. The output window shows 1111/1111 test cases passed, 1/1 attempts correct, 4/4 points scored, and a time taken of 0.56 seconds. The code editor has syntax highlighting and line numbers. Below the editor are solve next links and stay ahead with suggestions.

```
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             int carry = 0;
11
12             for (int i = 0; i < ans.size(); i++) {
13                 int value = ans.get(i) * num + carry;
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
30 }
```