

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:

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

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

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

Sol

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

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓ Suggest Feedback

Test Cases Passed Attempts : Correct / Total

1121 / 1121 1 / 1

Accuracy : 100%

Points Scored ⓘ Time Taken

4 / 4 0.67

Your Total Score: 14 ↗

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

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

$1 \leq k \leq 107$

$1 \leq n \leq 105$

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

Sol

```

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

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓ Suggest Feedback

Test Cases Passed 1115 / 1115

Attempts : Correct / Total 1 / 1 Accuracy : 100%

Points Scored 4 / 4 Your Total Score: 18 ↗

Solve Next Minimum Jumps A difference of values and indexes Minimize the Heights I

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

Custom Input Compile & Run Submit

3.

Given an array $\text{arr}[]$ 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: $k = 2$, $\text{arr}[] = [1, 5, 8, 10]$

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: $k = 3$, $\text{arr}[] = [3, 9, 12, 16, 20]$

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

$1 \leq k \leq 107$

$1 \leq n \leq 105$

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

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

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: $1! = 1$

Sol

```

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

```

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓ Suggest Feedback

Test Cases Passed 1111 / 1111 Attempts : Correct / Total 1 / 1 Accuracy : 100%

Points Scored 4 / 4 Time Taken 0.56

Your Total Score: 26 ↗

Solve Next Large Factorial Number following a pattern Rank The Permutations

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

Custom Input Compile & Run Submit

5.

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 $a[]$

Sol

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed **1114 / 1114**

Attempts : Correct / Total **1 / 2**

Accuracy : 50%

Points Scored **1 / 1**

Time Taken **0.64**

Your Total Score: 27

Solve Next

[Counting elements in two arrays](#) [Union of 2 Sorted Arrays](#)

[Left most and right most index](#)

Stay Ahead With:

Custom Input **Compile & Run** **Submit**

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