

JAVA PRACTICE SHEET(26/01/26)

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
1121 / 1121

Attempts : Correct / Total
1 / 1

Accuracy : 100%

Points Scored ⓘ
4 / 4

Your Total Score: 14 ↗

Time Taken
0.67

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

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** ↗

Time Taken **0.71**

```

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

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.

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

For example:

If $\text{arr}[i] = 3$, you can jump to index $i + 1$, $i + 2$, or $i + 3$ from position i .

If $\text{arr}[i] = 0$, 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.

School of Computer Science and Engineering

Experiment List for Programming Ability and Logic Building - 1

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

Examples :

Input: $\text{arr}[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]$

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: $\text{arr} = [1, 4, 3, 2, 6, 7]$

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:

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

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

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

4.

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 <= 105

nums.length == n + 1

1 <= nums[i] <= n

All the integers in nums appear only once except for precisely one integer which appears two or more times.

Sol

The screenshot shows a code editor interface with a Java file named Solution.java. The code implements the Floyd's Tortoise and Hare algorithm to detect a cycle in the array indices. It includes a detailed docstring explaining the phases and logic. Below the code, there are performance metrics: Runtime 4 ms (Beats 90.96%) and Memory 83.17 MB (Beats 32.25%).

```
1 class Solution {
2     public int findDuplicate(int[] nums) {
3         // Phase 1: Detect the cycle
4         int slow = nums[0];
5         int fast = nums[0];
6
7         do {
8             slow = nums[slow];
9             fast = nums[nums[fast]];
10        } while (slow != fast);
11
12        slow = nums[0];
13        while (slow != fast) {
14            slow = nums[slow];
15            fast = nums[fast];
16        }
17
18        return slow;
19    }
}
```

5.

Given two sorted arrays a[] and b[] of size n and m respectively, the task is to merge them in sorted order without using any extra space. Modify a[] so that it contains the first n elements and modify b[] so that it contains the last m elements.

Examples:

Input: a[] = [2, 4, 7, 10], b[] = [2, 3]

Output: a[] = [2, 2, 3, 4], b[] = [7, 10]

Explanation: After merging the two non-decreasing arrays, we get, [2, 2, 3, 4, 7, 10]

Input: a[] = [1, 5, 9, 10, 15, 20], b[] = [2, 3, 8, 13]

Output: $a[] = [1, 2, 3, 5, 8, 9], b[] = [10, 13, 15, 20]$

Explanation: After merging two sorted arrays we get $[1, 2, 3, 5, 8, 9, 10, 13, 15, 20]$.

Input: $a[] = [0, 1], b[] = [2, 3]$

Output: $a[] = [0, 1], b[] = [2, 3]$

Explanation: After merging two sorted arrays we get $[0, 1, 2, 3]$.

Constraints:

$1 \leq n, m \leq 105$

$0 \leq a[i], b[i] \leq 10^7$

Sol

The screenshot shows a LeetCode problem solved successfully. The code is a Java implementation of the merge algorithm. It defines a class Solution with a public method mergeArrays that takes two integer arrays a and b as input. The code uses a gap-based merging strategy, starting with a gap of size n+m and decreasing it until it reaches 1. It handles three cases for each element pair (a[i], b[j]): if a[i] <= b[j], swap them; if a[i] > b[j], swap them and move the gap left; if they are equal, move both pointers and the gap left. A private helper method nextGap calculates the next gap size based on the current gap.

```
1. class Solution {
2.     public void mergeArrays(int[] a[], int[] b[]) {
3.         int n = a.length;
4.         int m = b.length;
5.         int gap = nextGap(n + m);
6.         while (gap > 0) {
7.             int i = 0, j = gap;
8.             while (j < n + gap) {
9.                 if (i < n && j >= n) {
10.                     if (a[i] > a[j]) {
11.                         int temp = a[i];
12.                         a[i] = a[j];
13.                         a[j] = temp;
14.                     }
15.                 } else if (i < n && j >= n) {
16.                     if (a[i] > b[j - n]) {
17.                         int temp = a[i];
18.                         a[i] = b[j - n];
19.                         b[j - n] = temp;
20.                     }
21.                 } else {
22.                     if (b[i - n] > b[j - n]) {
23.                         int temp = b[i - n];
24.                         b[i - n] = b[j - n];
25.                         b[j - n] = temp;
26.                     }
27.                 }
28.             }
29.             i++;
30.             j++;
31.         }
32.         gap = nextGap(gap);
33.     }
34. }
35. private int nextGap(int gap) {
36.     if (gap <= 1) return 0;
37.     return (gap / 2) + (gap % 2);
38. }
```

6.

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:

Input: $\text{intervals} = [[1,3],[2,6],[8,10],[15,18]]$

Output: $[[1,6],[8,10],[15,18]]$

Explanation: Since intervals $[1,3]$ and $[2,6]$ overlap, merge them into $[1,6]$.

Example 2:

Input: $\text{intervals} = [[1,4],[4,5]]$

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.

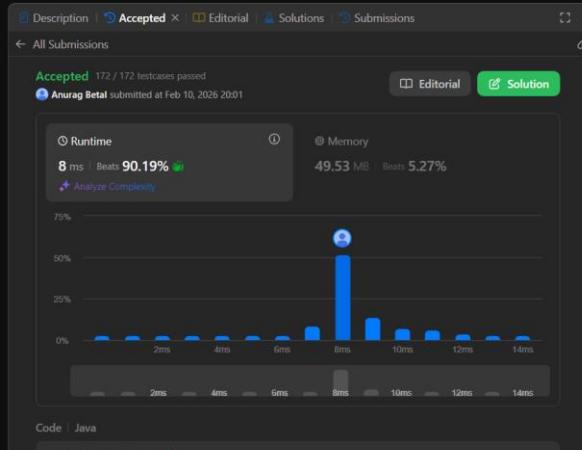
Constraints:

$1 \leq \text{intervals.length} \leq 10^4$

$\text{intervals}[i].length == 2$

$0 \leq \text{start}_i \leq \text{end}_i \leq 10^4$

Sol



The screenshot shows a LeetCode submission page. At the top, it says "Accepted" with 172 / 172 testcases passed, submitted by Anurag Betal at Feb 10, 2026 20:01. Below this are two performance metrics: "Runtime" (8 ms, 90.19% beats) and "Memory" (49.53 MB, 5.27% beats). A "Code" tab is open, displaying the Java solution. The code implements a merging algorithm for overlapping intervals. It first sorts the intervals by start time. Then, it iterates through the sorted intervals, merging overlapping ones. If no overlap, it adds the current interval to the merged list. The final merged list is returned as an array of arrays. The code is as follows:

```
1 import java.util.*;
2 class Solution {
3     public int[][] merge(int[][] intervals) {
4         if (intervals == null || intervals.length <= 1) {
5             return intervals;
6         }
7         Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
8         List<int[]> merged = new ArrayList<>();
9
10        int start = intervals[0][0];
11        int end = intervals[0][1];
12
13        for (int i = 1; i < intervals.length; i++) {
14            if (intervals[i][0] <= end) {
15                end = Math.max(end, intervals[i][1]);
16            } else {
17                merged.add(new int[]{start, end});
18                start = intervals[i][0];
19                end = intervals[i][1];
20            }
21        }
22        merged.add(new int[]{start, end});
23
24        return merged.toArray(new int[merged.size()][1]);
25    }
26
27
28
29 }
30
```

7.

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

Sol

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

- Output Window:** Displays "Compilation Results" and "Y.O.G.I. (AI Bot)".
- Problem Solved Successfully:** Shows "1215 / 1215" attempts correct.
- Attempts:** 1/1, Accuracy: 100%.
- Points Scored:** 2 / 2.
- Time Taken:** 3.35.
- Code:** A Java class named Solution with a method commonElements that takes three lists of integers and returns a list of common elements.

```
1* import java.util.*;
2* class Solution {
3*     // Function to find common elements in three arrays.
4*     public List<Integer> commonElements(List<Integer> arr1, List<Integer> arr2,
5*                                         List<Integer> arr3){
6*         int i = 0, j = 0, k = 0;
7*         List<Integer> result = new ArrayList<>();
8*         int n1 = arr1.size();
9*         int n2 = arr2.size();
10*        int n3 = arr3.size();
11*        while (i < n1 && j < n2 && k < n3) {
12*            int a = arr1.get(i);
13*            int b = arr2.get(j);
14*            int c = arr3.get(k);
15*            if (a == b && b == c) {
16*                result.add(a);
17*                while (i < n1 && arr1.get(i) == a) i++;
18*                while (j < n2 && arr2.get(j) == a) j++;
19*                while (k < n3 && arr3.get(k) == a) k++;
20*            }
21*        }
22*        else {
23*            int min = Math.min(a, Math.min(b, c));
24*            if (a == min) i++;
25*            else if (b == min) j++;
26*            else k++;
27*        }
28*    }
29*    if (result.isEmpty()) {
30*        result.add(-1);
31*    }
32*    return result;
33* }
```

- Solve Next:** Buttons for "Two Repeated Elements", "Sorted and Rotated Minimum", and "Sorted Insert Position".
- Stay Ahead With:** A button for "Build 21 Projects in 21 Days".
- Bottom Buttons:** "Custom Input", "Compile & Run", and "Submit".

8.

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: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

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

The screenshot shows a programming environment with the following details:

- Output Window:** Displays "Problem Solved Successfully" with a green checkmark.
- Compilation Results:** Shows "Test Cases Passed" as 1111 / 1111, "Attempts : Correct / Total" as 1 / 1, and "Accuracy : 100%".
- Code Editor:** Contains Java code for calculating the factorial of a number. The code uses an ArrayList to store intermediate results and iterates from 2 to n, multiplying each value by the current carry and taking the result modulo 10 to keep it within a single digit.
- Bottom Navigation:** Includes links for "Solve Next", "Large Factorial", "Number following a pattern", and "Rank The Permutations".
- Stay Ahead With:** A section for "Build 21 Projects in 21 Days" featuring a 3D cube icon.
- Buttons:** "Custom Input", "Compile & Run", and "Submit".

9.

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

```
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* 
```

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

10.

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.

Sol

```

1* import java.util.*;
2* class Solution {
3*     public boolean hasTripletSum(int arr[], int target) {
4*         // Write here
5*         int n = arr.length;
6*         if (n < 3) return false;
7*         Arrays.sort(arr);
8*         for (int i = 0; i < n - 2; i++) {
9*             int left = i + 1;
10*            int right = n - 1;
11*            while (left < right) {
12*                int sum = arr[i] + arr[left] + arr[right];
13*                if (sum == target) {
14*                    return true;
15*                } else if (sum < target) {
16*                    left++;
17*                } else {
18*                    right--;
19*                }
20*            }
21*        }
22*        return false;
23*    }
24* }
25* }
26* }
27* }

```

11.

Given an array $\text{arr}[]$ with non-negative integers representing the height of blocks.

If the width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.

Examples:

Input: $\text{arr}[] = [3, 0, 1, 0, 4, 0, 2]$

Output: 10

Explanation: Total water trapped = $0 + 3 + 2 + 3 + 0 + 2 + 0 = 10$ units.

Input: $\text{arr}[] = [3, 0, 2, 0, 4]$

Output: 7

Explanation: Total water trapped = $0 + 3 + 1 + 3 + 0 = 7$ units.

Input: $\text{arr}[] = [1, 2, 3, 4]$

Output: 0

Explanation: We cannot trap water as there is no height bound on both sides.

Input: $\text{arr}[] = [2, 1, 5, 3, 1, 0, 4]$

Output: 9

Explanation: Total water trapped = $0 + 1 + 0 + 1 + 3 + 4 + 0 = 9$ units.

Sol

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 8 / 8

Your Total Score: 45 ↗

Time Taken 0.26

Solve Next

Longest Arithmetic Subsequence | Rod Cutting | Jump Game

Stay Ahead With:

 Build 21 Projects in 21 Days

Build real-world ML, Deep Learning & Gen AI projects

Custom Input Compile & Run Submit

```
1- class Solution {
2-     public int maxWater(int arr[]) {
3-         // code here
4-         int n = arr.length;
5-         if (n == 0) return 0;
6-
7-         int left = 0, right = n - 1;
8-         int leftMax = 0, rightMax = 0;
9-         int water = 0;
10-
11-        while (left < right) {
12-            if (arr[left] <= arr[right]) {
13-                leftMax = arr[left];
14-            } else {
15-                water += leftMax - arr[left];
16-            }
17-            left++;
18-        } else {
19-            if (arr[right] >= arr[left]) {
20-                rightMax = arr[right];
21-            } else {
22-                water += rightMax - arr[right];
23-            }
24-            right--;
25-        }
26-
27-    }
28-
29-    return water;
30- }
31- }
32-
33- }
```