

## **Experiment List for Programming Ability and Logic Building-2**

Name : Ikshu Goel

Batch : 2CSE5

Roll Number : 2410030774

Week : 26/01/26 to 31/01/26

Lecture : 1 & 2

### **EXPERIMENT 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 Java code editor interface on a web browser. The code is a simple solution to find the kth smallest element in an array by sorting the array and returning the element at index k-1.

```
1- import java.util.*;
2-
3- class Solution {
4-     public static int kthSmallest(int[] arr, int k) {
5-         Arrays.sort(arr);
6-         return arr[k - 1];
7-     }
8-
9- }
```

The code editor has tabs for "Java (21)", "Start Timer", and "Custom Input". The "Custom Input" tab is selected. The input fields show the array `arr[] = [7, 10, 4, 3, 20, 15]` and the value `k = 3`. The output field shows the result `7`.

# EXPERIMENT 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 web-based Java compiler and debugger interface. At the top, there's a navigation bar with links like 'Kth Smallest', 'Minim', 'Find', 'Merge', 'Comm', 'Facto', 'Array', 'Triple', and 'Traps'. Below the navigation is a search bar and a header with 'Courses', 'Tutorials', 'Practice', and 'Jobs' dropdowns. The main area is titled 'Problem' and shows 'Output Window' and 'Compilation Results' tabs. Under 'Compilation Completed', it says 'Case 1' and displays input fields for 'k' (containing '2') and 'arr[]' (containing '1 5 8 10'). It also shows 'Your Output' as '5' and 'Expected Output' as '5'. On the right, the code editor contains Java code for calculating the minimum difference between elements in an array after performing at most k jumps:

```
1 import java.util.*;
2
3 class Solution {
4
5     public int getMinDiff(int[] arr, int k) {
6
7         int n = arr.length;
8         Arrays.sort(arr);
9
10        int min = arr[0] + k;
11        int max = arr[n - 1] - k;
12
13        if (min < 0)
14            min = 0;
15
16        return max - min;
17    }
18
19
20 }
```

# EXPERIMENT 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.

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:  $\text{arr} = [0, 10, 20]$

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 on the GeeksforGeeks platform. The code is a solution for the problem "Minimum Number of Jumps". The code uses a greedy algorithm to calculate the minimum number of jumps required to reach the end of the array. It iterates through the array, keeping track of the current index and the maximum index it can reach. If the current index is greater than or equal to the last index, it returns the total jumps. Otherwise, it increments the jump count and updates the maximum index reached.

```
1+ class Solution {  
2+     public int minJumps(int[] arr) {  
3+         int n = arr.length;  
4+         if (n <= 1) return 0;  
5+         if (arr[0] == 0) return -1;  
6+         int jumps = 0;  
7+         int i = 0;  
8+         while (i < n - 1) {  
9+             if (arr[i] == 0) return -1;  
10+            i = i + arr[i];  
11+            jumps++;  
12+            if (i >= n - 1) return jumps;  
13+        }  
14+    }  
15+    return -1;  
16+ }
```

# EXPERIMENT 4

Given an array of integers  $\text{nums}$  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:

$1 \leq n \leq 105$

$\text{nums.length} == n + 1$

$1 \leq \text{nums}[i] \leq n$

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

The screenshot shows a LeetCode problem page for "287. Find the Duplicate Number". The problem description states: "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." The Java code implements the Floyd's Tortoise and Hare algorithm to find the duplicate. The code is as follows:

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

The code is saved and the test result shows it is accepted with a runtime of 0 ms. The input is [1,3,4,2,2] and the output is 2.

# EXPERIMENT 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:

$1 \leq n, m \leq 105$

$0 \leq a[i], b[i] \leq 107$

The screenshot shows a Java IDE interface on geeksforgeeks.org. The code editor displays a Java class named Solution with a static method mergeArrays. The method takes two integer arrays, a and b, as parameters and returns nothing. It uses two pointers, i and j, to iterate through the arrays from the end towards the beginning. If a[i] is greater than b[j], it swaps them and increments j. Both pointers are then decremented. This process continues until all elements have been merged. Finally, both arrays are sorted again using Arrays.sort().

```
1+ import java.util.*;
2
3+ class Solution {
4+     void mergeArrays(int[] a, int[] b) {
5+         int n = a.length;
6+         int m = b.length;
7+
8+         int i = n - 1;
9+         int j = 0;
10+
11+        while (i >= 0 && j < m) {
12+            if (a[i] > b[j]) {
13+                int temp = a[i];
14+                a[i] = b[j];
15+                b[j] = temp;
16+            }
17+            i--;
18+            j++;
19+        }
20+
21+        Arrays.sort(a);
22+        Arrays.sort(b);
23+    }
24+
25+
26+ }
```

The left panel shows the 'Output Window' with 'Compilation Results' and 'Custom Input' sections. Under 'Custom Input', there are input fields for arrays a and b, and a 'Your Output:' field containing the expected result. The 'Expected Output:' field also contains the same result. The right panel shows the Java compiler interface with tabs for 'Courses', 'Tutorials', 'Practice', and 'Jobs'. The status bar at the bottom indicates 'Custom Input', 'Compile & Run', and 'Submit' buttons.

# EXPERIMENT 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:  $\text{intervals} = [[4,7],[1,4]]$

Output:  $[[1,7]]$

Explanation: Intervals [1,4] and [4,7] are considered overlapping.

Constraints:

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

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

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

The screenshot shows the LeetCode platform with the problem "56. Merge Intervals".

**Description:** 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.

**Java Code:**

```
1 import java.util.*;
2 class Solution {
3     public int[][] merge(int[][] intervals) {
4         Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
5         List<int[]> result = new ArrayList<>();
6         int[] current = intervals[0];
7         result.add(current);
8
9         for (int[] interval : intervals) {
10            if (interval[0] <= current[1]) {
11                current[1] = Math.max(current[1], interval[1]);
12            } else {
13                current = interval;
14                result.add(current);
15            }
16        }
17        return result.toArray(new int[result.size()][]);
18    }
19 }
```

**Constraints:**

- $1 \leq \text{intervals.length} \leq 10^4$
- $\text{intervals}[i].length == 2$

**Testcase:** Accepted. Runtime: 0 ms. Cases: Case 1, Case 2, Case 3.

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

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

# EXPERIMENT 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

```

3+ class Solution {
4-
5-     ArrayList<Integer> commonElements(int[] A, int[] B, int[] C) {
6-
7-         ArrayList<Integer> res = new ArrayList<()>();
8-
9-         int i = 0, j = 0, k = 0;
10-
11-        while (i < A.length && j < B.length && k < C.length) {
12-
13-            if (A[i] == B[j] && B[j] == C[k]) {
14-
15-                if (res.size() == 0 || res.get(res.size() - 1) != A[i]) {
16-                    res.add(A[i]);
17-
18-                }
19-
20-                i++;
21-                j++;
22-                k++;
23-
24-            } else if (A[i] < B[j]) {
25-                i++;
26-            } else if (B[j] < C[k]) {
27-                j++;
28-            } else {
29-                k++;
30-            }
31-
32-            if (res.size() == 0) {
33-                res.add(-1);
34-            }
35-
36-
37-        }
38-
39-    }

```

**Common in 3 Sorted Arrays**

Difficulty: Easy Accuracy: 22.16% Submissions: 441K+ Points: 2

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, 2, 2, 2, 2]  
**Output:** [1, 2]  
**Explanation:** We do not need to consider duplicates

**Constraints:**  
 $1 \leq \text{arr1.size(), arr2.size(), arr3.size()} \leq 10^5$   
 $-10^5 \leq \text{arr1}_i, \text{arr2}_i, \text{arr3}_i \leq 10^5$

Try more examples

# EXPERIMENT 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:**  $5! = 1 * 2 * 3 * 4 * 5 = 120$

**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 browser window for geeksforgeeks.org/problems/factorials-of-large-numbers2508/1. The page title is "geeksforgeeks.org/problems/factorials-of-large-numbers2508/1". The main content area has tabs for "Problem", "Editorial", "Submissions", and "Comments". On the left, there's an "Output Window" and a "Compilation Results" section. The "Compilation Completed" section shows a case with input "n" set to "5" and output "120". The "Expected Output" is also "120". On the right, a Java code editor displays the following code:

```
1+ import java.util.*;
2
3+ class Solution {
4+     public static ArrayList<Integer> factorial(int n) {
5+         ArrayList<Integer> res = new ArrayList<O>;
6+         res.add(1);
7+
8+         for (int i = 2; i <= n; i++) {
9+             int carry = 0;
10+            for (int j = 0; j < res.size(); j++) {
11+                int val = res.get(j) * i + carry;
12+                res.set(j, val % 10);
13+                carry = val / 10;
14+            }
15+            while (carry > 0) {
16+                res.add(carry % 10);
17+                carry /= 10;
18+            }
19+        }
20+        Collections.reverse(res);
21+        return res;
22+    }
23+
24+}
```

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

# EXPERIMENT 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[]

The screenshot shows a browser window on the GeeksforGeeks website. The URL is [geeksforgeeks.org/problems/array-subset-of-another-array2317/1](https://www.geeksforgeeks.org/problems/array-subset-of-another-array2317/1). The page title is "GeeksForGeeks". The main content area displays a Java code editor with the following code:

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

To the left of the code editor is the "Output Window" section, which includes "Compilation Results" and "Custom Input". It shows the following input and output:

**Input:**  
a[] =  
11 7 1 13 21 3 7 3  
  
b[] =  
11 3 7 1 7

**Your Output:**  
true

**Expected Output:**  
true

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

## EXPERIMENT 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.

The screenshot shows a Java code editor on the GeeksforGeeks platform. The code implements a solution to find a triplet in an array whose sum equals a given target. It uses a three-pointer approach after sorting the array.

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

The input provided in the editor is:

```
arr[] = 1 4 45 6 10 8
target = 13
```

The output section shows:

Your Output:  
true

Expected Output:  
true

# EXPERIMENT 11

Given an array **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:** arr[] = [3, 0, 1, 0, 4, 0 2]

**Output:** 10

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

2

Take

Home

**Input:** arr[] = [3, 0, 2, 0, 4]

**Output:** 7

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

**Input:** arr[] = [1, 2, 3, 4]

**Output:** 0

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

**Input:** arr[] = [2, 1, 5, 3, 1, 0, 4]

**Output:** 9

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

Kth Smallest Element | Minim... | Minim... | Minim... | Find t... | Merge... | Comm... | Facto... | Array... | Triple... | Trap... | + | Paused | ...

geeksforgeeks.org/problems/trapping-rain-water-1587115621/

Get 90% Refund Courses Tutorials Practice Jobs

Problem Editorial Submissions Comments

Output Window X

Compilation Results Custom Input

Compilation Completed

Case 1

Input: arr[] = 3 0 1 0 4 0 2

Your Output: 10

Expected Output: 10

Java (21) Start Timer

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

Custom Input Compile & Run Submit