EXPERIMENT 8

Name : Vatsala Singh

UID: 22BCS10028

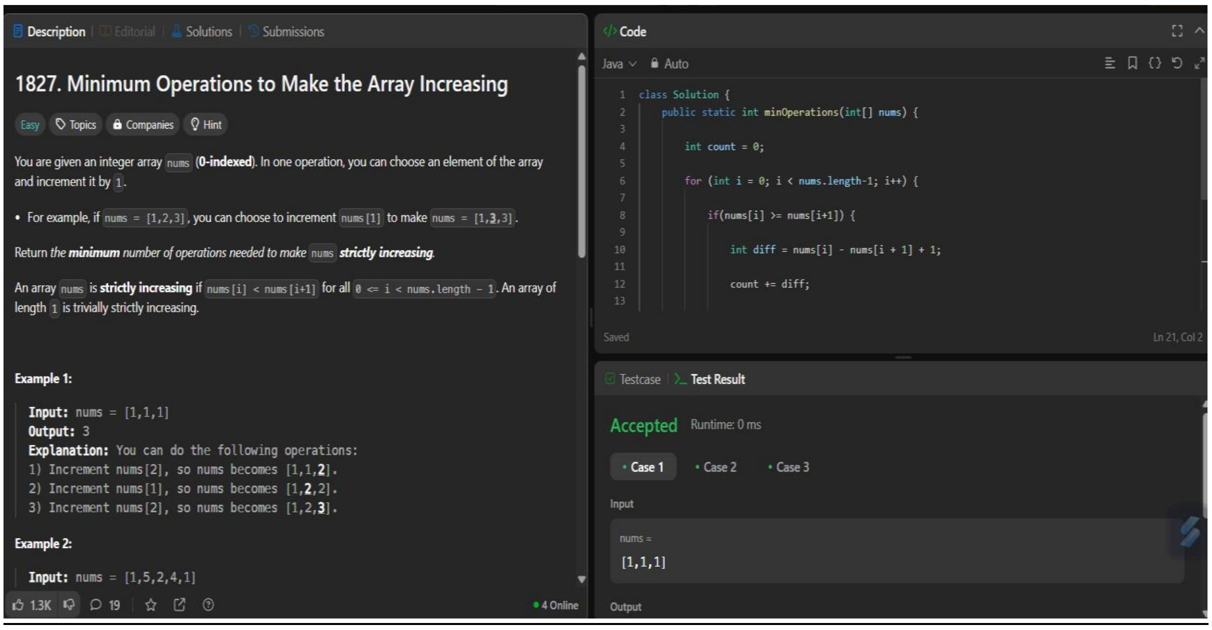
Section/Group: 605-A

Question 1

Minimum Operations to Make the Array Increasing

Solution

```
class Solution {
public static int minOperations(int[] nums) {

int count = 0;

for (int i = 0; i < nums.length-1; i++) {

if(nums[i] >= nums[i+1]) {

int diff = nums[i] - nums[i + 1] + 1;

count += diff;


nums[i+1] += diff;
}

}
return count;
}
}
```



1827. Minimum Operations to Make the Array Increasing

You are given an integer array `nums` (**0-indexed**). In one operation, you can choose an element of the array and increment it by `1`.

- For example, if `nums = [1,2,3]`, you can choose to increment `nums[1]` to make `nums = [1,3,3]`.

Return *the* ***minimum*** *number of operations needed to make* `nums` ***strictly increasing***.

An array `nums` is **strictly increasing** if `nums[i] < nums[i+1]` for all `0 <= i < nums.length - 1`. An array of length `1` is trivially strictly increasing.

Example 1:

```
Input: nums = [1,1,1]
Output: 3
Explanation: You can do the following operations:
1) Increment nums[2], so nums becomes [1,1,2].
2) Increment nums[1], so nums becomes [1,2,2].
3) Increment nums[2], so nums becomes [1,2,3].
```

Example 2:

```
Input: nums = [1,5,2,4,1]
```

```java
class Solution {
    public static int minOperations(int[] nums) {

        int count = 0;

        for (int i = 0; i < nums.length-1; i++) {

            if(nums[i] >= nums[i+1]) {

                int diff = nums[i] - nums[i + 1] + 1;

                count += diff;

```

Accepted Runtime: 0 ms

Input

```
nums =
[1,1,1]
```

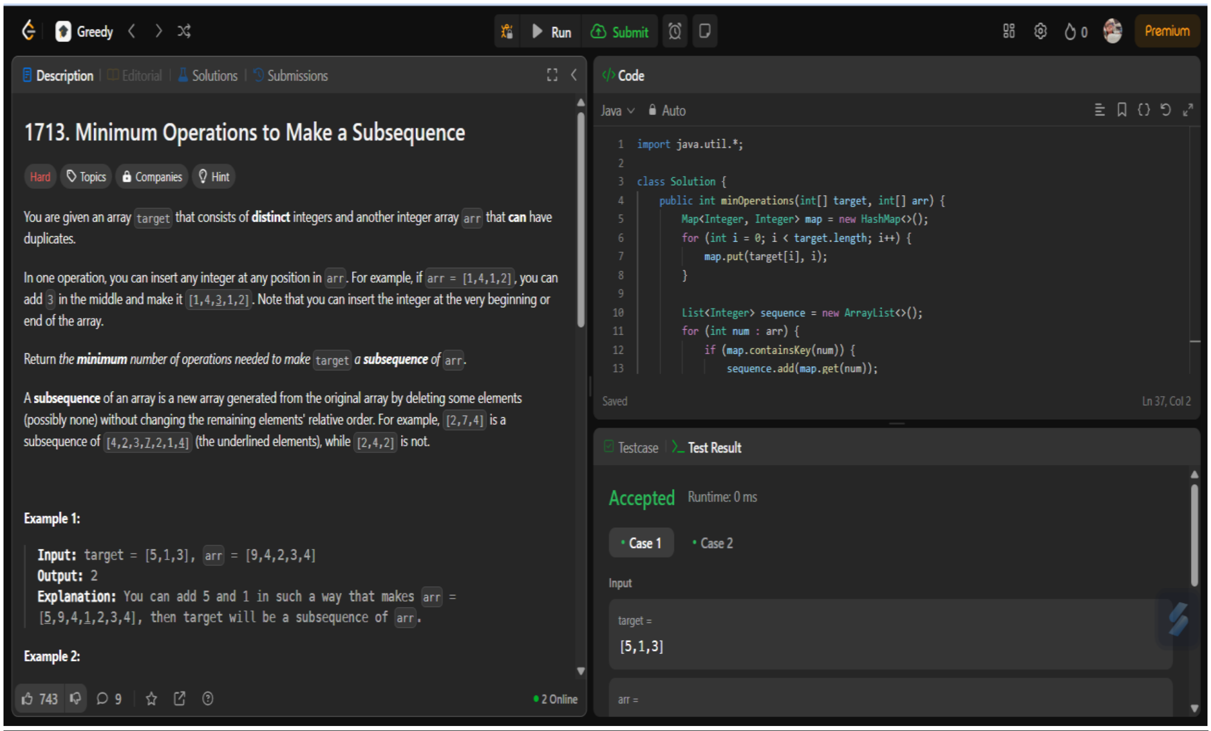
Output

Question 2

Minimum Operations to Make a Subsequence

```java
import java.util.*;

class Solution {
public int minOperations(int[] target, int[] arr) {
Map<Integer, Integer> map = new HashMap<>();
for (int i = 0; i < target.length; i++) {
map.put(target[i], i);
}

List<Integer> sequence = new ArrayList<>();
for (int num : arr) {
if (map.containsKey(num)) {
sequence.add(map.get(num));
}
}

int maxSubsequence = lis(sequence);
return target.length - maxSubsequence;
}

private int lis(List<Integer> list) {
```

```
List<Integer> max = new ArrayList<>();
for (int num : list) {
if (max.isEmpty() || num > max.get(max.size() - 1)) {
max.add(num);
} else {
int pos = Collections.binarySearch(max, num);
if (pos >= 0) {
max.set(pos, num);
} else {
max.set(-(pos + 1), num);
}
}
}
return max.size();
}
}
```



1713. Minimum Operations to Make a Subsequence

You are given an array `target` that consists of **distinct** integers and another integer array `arr` that **can** have duplicates.

In one operation, you can insert any integer at any position in `arr`. For example, if `arr = [1,4,1,2]`, you can add `3` in the middle and make it `[1,4,3,1,2]`. Note that you can insert the integer at the very beginning or end of the array.

Return *the **minimum** number of operations needed to make* `target` *a **subsequence** of* `arr`.

A **subsequence** of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the remaining elements' relative order. For example, `[2,7,4]` is a subsequence of `[4,2,3,7,2,1,4]` (the underlined elements), while `[2,4,2]` is not.

Example 1:

```
Input: target = [5,1,3], arr = [9,4,2,3,4]
Output: 2
Explanation: You can add 5 and 1 in such a way that makes arr =
[5,9,4,1,2,3,4], then target will be a subsequence of arr.
```

Example 2:

```java
import java.util.*;

class Solution {
    public int minOperations(int[] target, int[] arr) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < target.length; i++) {
            map.put(target[i], i);
        }

        List<Integer> sequence = new ArrayList<>();
        for (int num : arr) {
            if (map.containsKey(num)) {
                sequence.add(map.get(num));
```

Accepted Runtime: 0 ms

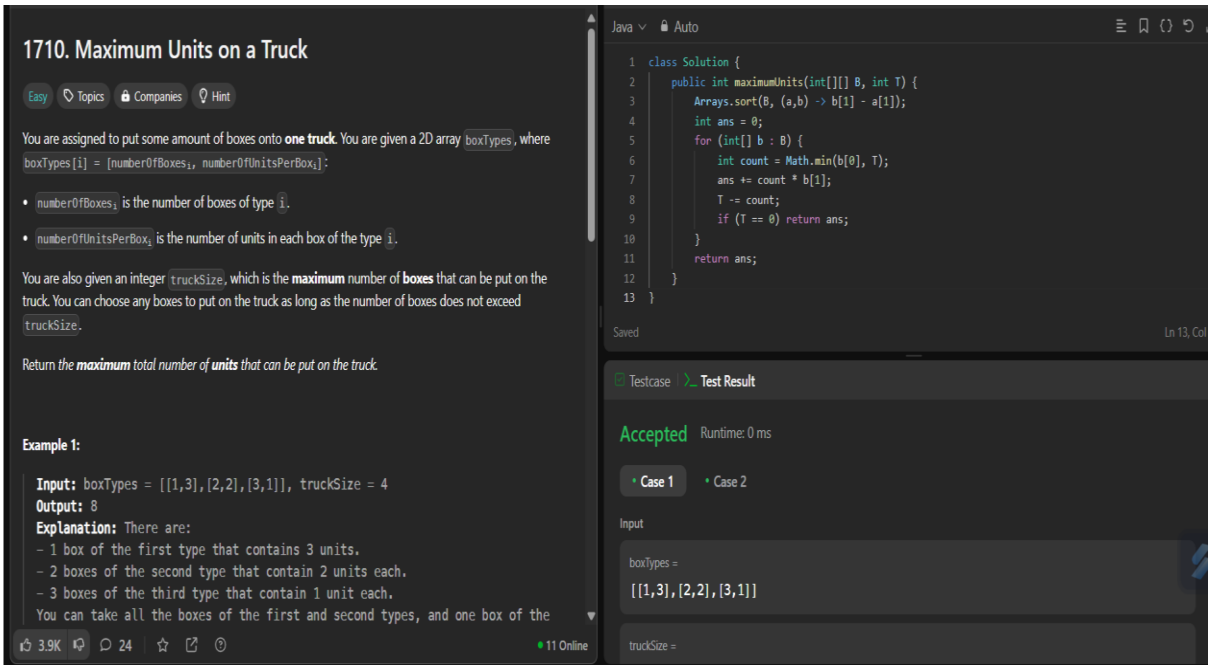
target =
[5,1,3]

arr =

Question 3

Maximum Units on a Truck

```
class Solution {
public int maximumUnits(int[][] B, int T) {
Arrays.sort(B, (a,b) -> b[1] - a[1]);
int ans = 0;

for (int[] b : B) {

int count = Math.min(b[0], T);

ans += count * b[1];

T -= count;

if (T == 0) return ans;

}

return ans;

}

}
```



1710. Maximum Units on a Truck

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array `boxTypes`, where `boxTypes[i] = [numberOfBoxes_i, numberOfUnitsPerBox_i]`:

- `numberOfBoxes_i` is the number of boxes of type `i`.
- `numberOfUnitsPerBox_i` is the number of units in each box of the type `i`.

You are also given an integer `truckSize`, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed `truckSize`.

Return *the **maximum** total number of **units** that can be put on the truck.*

Example 1:

```
Input: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second types, and one box of the
```

```java
class Solution {
    public int maximumUnits(int[][] B, int T) {
        Arrays.sort(B, (a,b) -> b[1] - a[1]);
        int ans = 0;
        for (int[] b : B) {
            int count = Math.min(b[0], T);
            ans += count * b[1];
            T -= count;
            if (T == 0) return ans;
        }
        return ans;
    }
}
```

Accepted Runtime: 0 ms

Input

boxTypes =
[[1,3],[2,2],[3,1]]

truckSize =