Experiment - 7

Student Name: Nitil Jakhar
Branch: BE-CSE
Semester: 6th
Subject Name: AP Lab

UID: 22BCS17300
Section/Group:22BCS_NTPP_IOT-602/A
Subject Code: 22CSP-351
Date: 28/03/2025

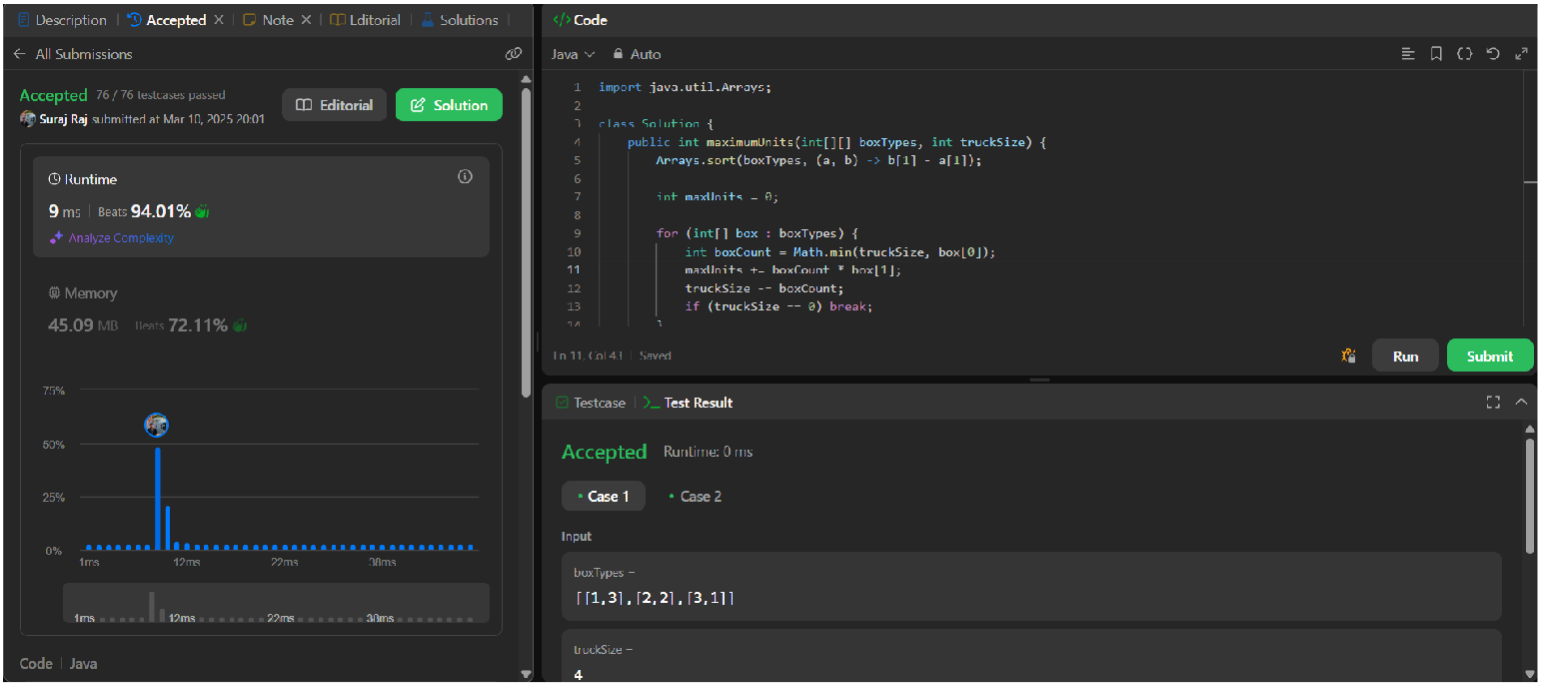
A. Maximum Units on a Truck

1. **Aim:** 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 is the number of boxes of type i.
   - numberOfUnitsPerBox$_i$ is the number of units in each box of the type i.

2. **Code**

```
import java.util.Arrays;

class Solution { public int maximumUnits(int[][] boxTypes,
   int truckSize) {
      Arrays.sort(boxTypes, (a, b) -> b[1] - a[1]); int

      maxUnits = 0;

      for (int[] box : boxTypes) { int boxCount =
      Math.min(truckSize, box[0]); maxUnits +=
      boxCount * box[1]; truckSize -= boxCount; if
      (truckSize == 0) break; }

      return maxUnits;
   }
}
```

3. **Output:**



B. Minimum Operations to Make the Array Increasing

1. **Aim:** 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.

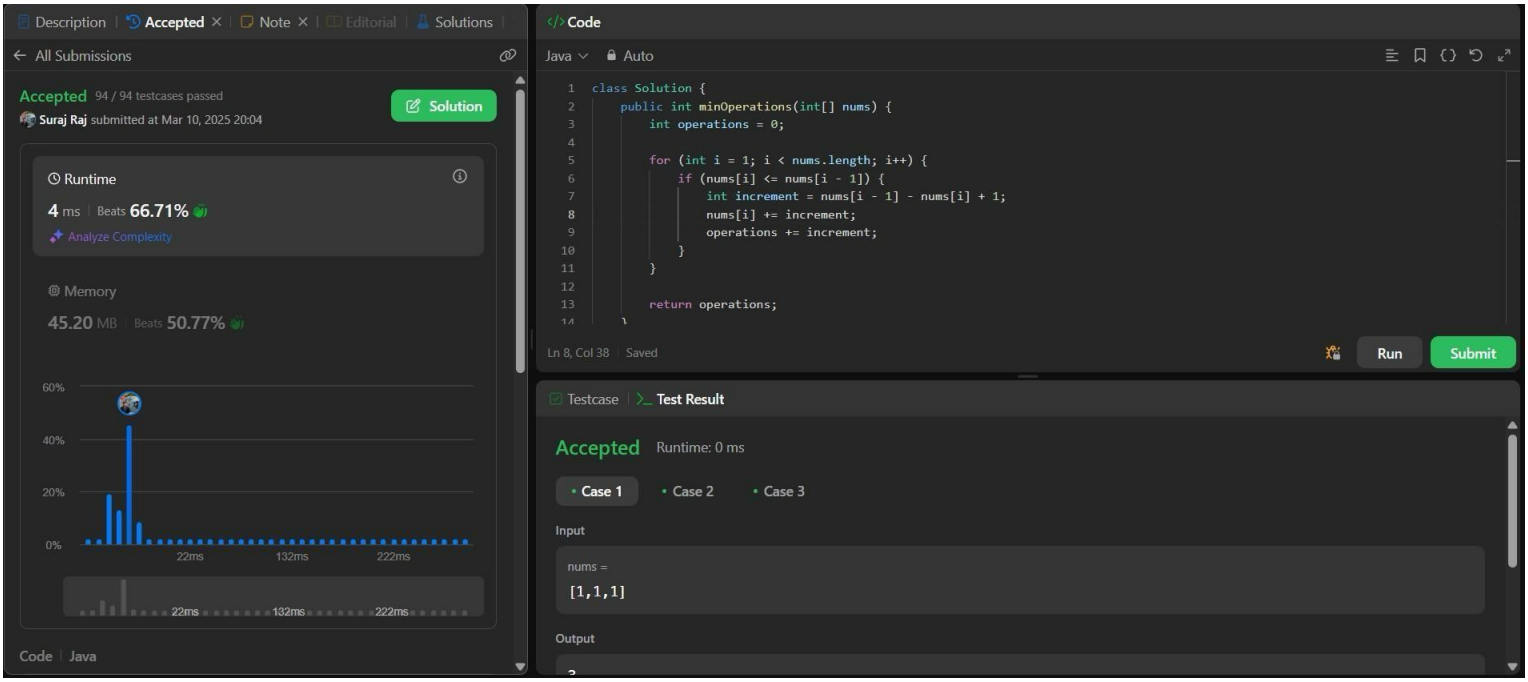
- For example, if nums = [1,2,3], you can choose to increment nums[1] to make nums = [1,**3**,3].
- Return *the **minimum** number of operations needed to make* nums ***strictly increasing***.

2. **Code:**

```
class Solution { public int minOperations(int[] nums) { int operations = 0;

    for (int i = 1; i < nums.length; i++) { if (nums[i] <= nums[i - 1]) { int increment = nums[i - 1] - nums[i] + 1; nums[i] += increment;
        operations += increment; }
    }

    return operations;
}
```

}

3. Output:



C. Remove Stones to Minimize the Total

1. **Aim:** You are given a 0-indexed integer array piles, where piles[i] represents the number of stones in the i^{th} pile, and an integer k. You should apply the following operation exactly k times:

2. **Code:**

```
import java.util.PriorityQueue;

class Solution { public int minStoneSum(int[]
   piles, int k) {
      PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) -> b - a); int

      totalStones = 0;

      for (int pile : piles) { maxHeap.add(pile)
         ;       totalStones        +=
         pile;
      }
```

```
        for (int i = 0; i < k; i++) { int largestPile = maxHeap.poll();
                int removedStones = largestPile / 2;
                totalStones -= removedStones;
            maxHeap.add(largestPile - removedStones);
        }

        return totalStones;
    }
}
```

3. Output: