WORKSHEET 7

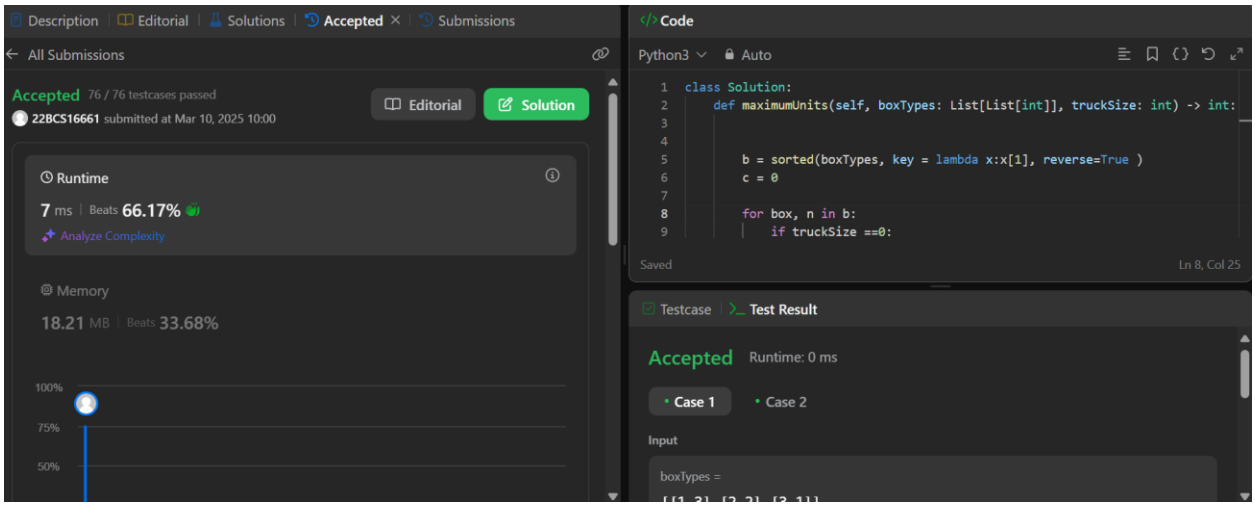
Student Name: Parth Arora
UID: 22BCS16661
Branch: BE-CSE
Section/Group: 22BCS_NTPP-602-A
Semester: 6th
Date of Performance: 10/03/2025
Subject Name: AP LAB - II
Subject Code: 22CSP-351

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$_i$ 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.*

2. **Source Code:**

```
class Solution:
    def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:

        b = sorted(boxTypes, key = lambda x:x[1], reverse=True )
        c = 0

        for box, n in b:
            if truckSize ==0:
                return c

            boxes = min(box, truckSize)
            c += boxes * n

            truckSize -= boxes
        return c
```

3. Screenshots of outputs:



2.

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:

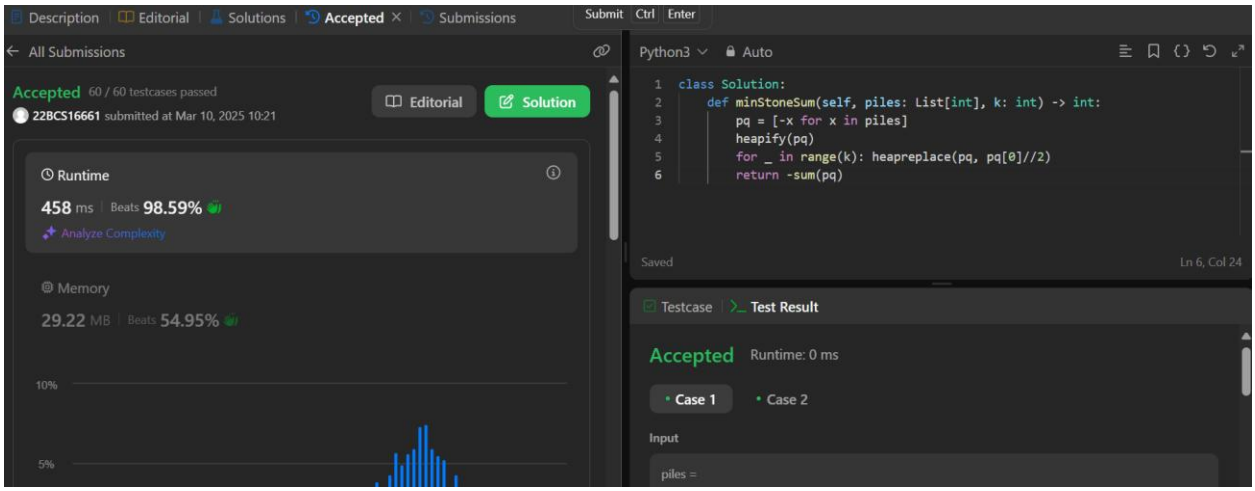
- Choose any piles[i] and **remove** floor(piles[i] / 2) stones from it.

Notice that you can apply the operation on the **same** pile more than once.
Return *the* ***minimum*** *possible total number of stones remaining after applying the* k *operations.*
floor(x) is the **greatest** integer that is **smaller** than or **equal** to x (i.e., rounds x down).

Source Code:

```
class Solution:
    def minStoneSum(self, piles: List[int], k: int) -> int:
        pq = [-x for x in piles]
        heapify(pq)
        for _ in range(k): heapreplace(pq, pq[0]//2)
        return -sum(pq)
```

Screenshots of outputs:



3.

Aim: 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.

Source Code:

```
class Solution:
    def minOperations(self, target: List[int], arr: List[int]) -> int:
        dic = {num: i for i, num in enumerate(target)}
        A = []
        for num in arr:
            if num in dic:
                A.append(dic[num])
        return len(target) - self.lengthOfLIS(A)

    def lengthOfLIS(self, nums):
        if not nums: return 0
        piles = []
        for num in nums:
            index = bisect.bisect_left(piles, num)
```

```
        if index == len(piles):
            piles.append(num)
        else:
            piles[index] = num
    return len(piles)
```

4. Screenshots of outputs: