WORKSHEET 7

Student Name: Jayant Sharma **UID:** 22BCS16668

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] = [numberOfBoxesi, numberOfUnitsPerBoxi]:

   numberOfBoxesi is the number of boxes of type i.
   numberOfUnitsPerBoxi 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.

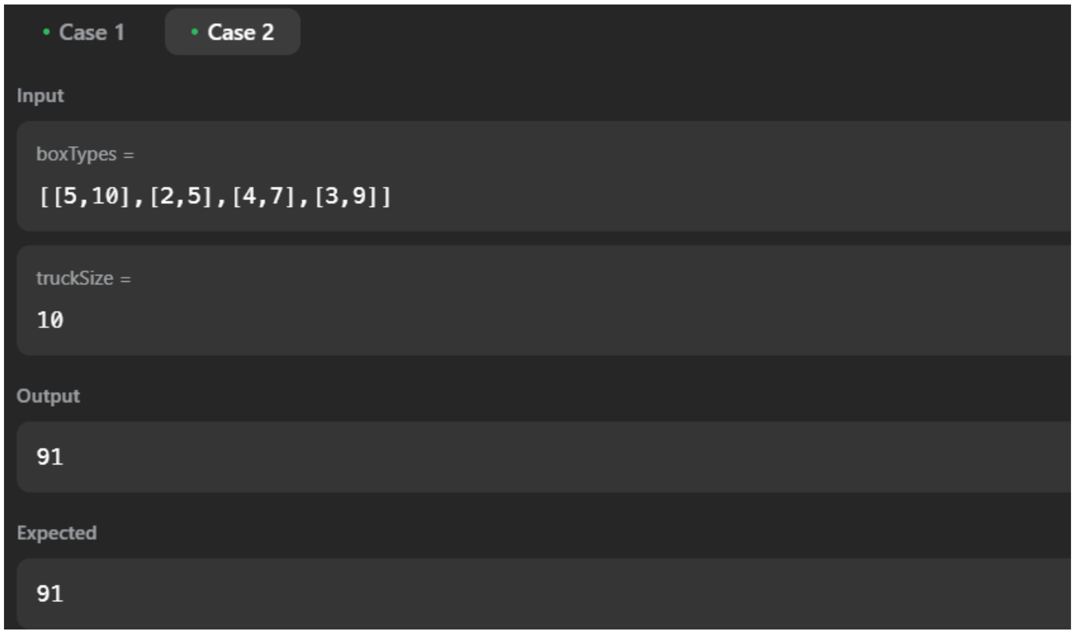
   Return the maximum total number of units that can be put on the truck

2. **Source Code:**

```
from typing import List

class Solution:
    def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        boxTypes.sort(key=lambda x: x[1], reverse=True)

        totalUnits = 0

        for box in boxTypes:
            boxCount, unitsPerBox = box
```

```
                boxesToTake = min(boxCount, truckSize)
                totalUnits += boxesToTake * unitsPerBox
                truckSize -= boxesToTake

                if truckSize == 0:
                    break

            return totalUnits
```

3. Screenshots of outputs:



```
Input
boxTypes =
[[5,10],[2,5],[4,7],[3,9]]
truckSize =
10
Output
91
Expected
91
```

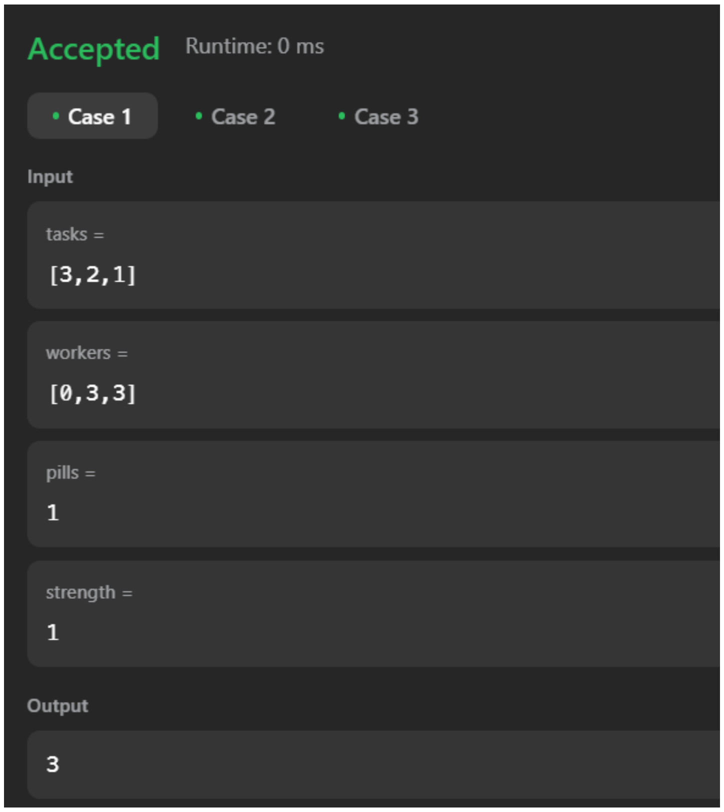
2. Aim: Maximum Number of Tasks You Can Assign

Source Code

```
from typing import List
```

```
class Solution:
    def maxTaskAssign(self, tasks: List[int], workers: List[int], pills: int, strength: int) -> int:

        tasks.sort()
        workers.sort()

        def canCompleteTasks(taskCount: int) -> bool:
            task_index = 0
            worker_index = 0
            used_pills = 0

            while task_index < taskCount and worker_index < len(workers):
                if workers[worker_index] >= tasks[task_index]:
                    task_index += 1
                elif workers[worker_index] + strength >= tasks[task_index] and used_pills < pills:
                    used_pills += 1
                    task_index += 1
                worker_index += 1

            return task_index == taskCount

        left, right = 0, min(len(tasks), len(workers))
        while left < right:
            mid = (left + right + 1) // 2
            if canCompleteTasks(mid):
                left = mid
            else:
                right = mid - 1
        return left
```

Screenshots of outputs:


Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tasks =
```
[3,2,1]
```
workers =
```
[0,3,3]
```
pills =
```
1
```
strength =
```
1
```
Output
```
3
```

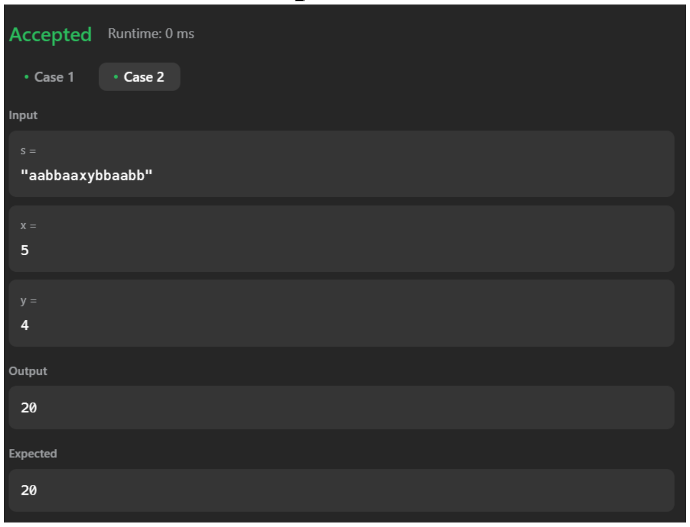
3. Aim: Maximum Score From Removing Substrings

Source Code:

```
class Solution:
    def maximumGain(self, s: str, x: int, y: int) -> int:
        def remove_substring(s, first, second, score):
            stack = []
            total_score = 0
            for char in s:
                if stack and stack[-1] == first and char == second:
                    stack.pop()
                    total_score += score
                else:
                    stack.append(char)
            return "".join(stack), total_score

        if x > y:
```

```
        s, score1 = remove_substring(s, 'a', 'b', x)
        _, score2 = remove_substring(s, 'b', 'a', y)
    else:
        s, score1 = remove_substring(s, 'b', 'a', y)
        _, score2 = remove_substring(s, 'a', 'b', x)

    return score1 + score2
```

Screenshots of outputs:



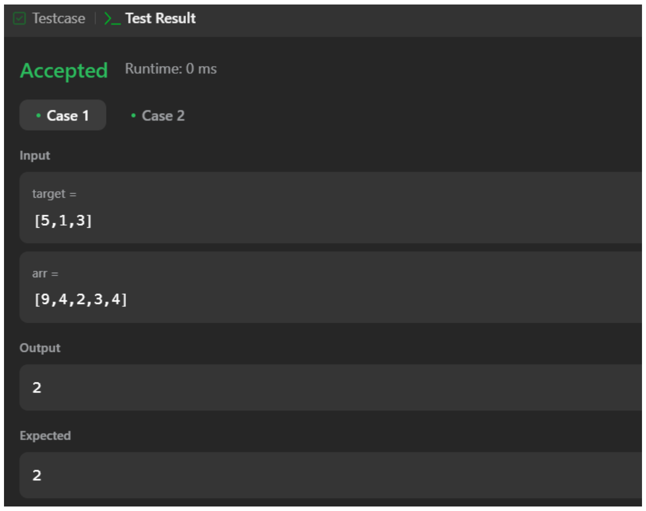
```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
s =
"aabbaaxybbaabb"
x =
5
y =
4
Output
20
Expected
20
```

4. Aim: Minimum Operations to Make a Subsequence

Source Code:

```
from bisect import bisect_left
from typing import List

class Solution:
    def minOperations(self, target: List[int], arr: List[int]) -> int:
        target_index_map = {value: index for index, value in enumerate(target)}

        arr_indices = []
        for num in arr:
            if num in target_index_map:
                arr_indices.append(target_index_map[num])

        lis = []
        for index in arr_indices:
            pos = bisect_left(lis, index)
            if pos == len(lis):
                lis.append(index)
            else:
                lis[pos] = index

        return len(target) - len(lis)
```

4. Screenshots of outputs:



```
Testcase | >_ Test Result
Accepted  Runtime: 0 ms
• Case 1   • Case 2
Input
target =
[5,1,3]
arr =
[9,4,2,3,4]
Output
2
Expected
2
```

Learning Outcomes:

- **Priority Queues (Heaps):** Learn how to use heaps for efficient retrieval and modification based on priority.
- **Greedy Algorithms:** Understand the greedy approach of making local optimal choices to achieve a global optimal solution.
- **Efficient Problem Solving with Heaps:** Apply heaps for dynamic data updates, optimizing insertions and removals in O(log n) time.
- **Binary Operations and Mathematical Modeling:** Use integer division and rounding down to reduce problem complexity.
- **Time and Space Complexity Analysis:** Analyze the time complexity of algorithms using heaps and evaluate space requirements.
- **Problem Decomposition:** Break down problems into manageable parts using efficient data structures and algorithms.