Experiment- 8

Student Name: Kamal Mehta
UID: 22BET10097
Branch: B.E - IT
Section/Group: 22BET-701/A
Semester: 6th
Date of Performance: 28-03-25
Subject Name: AP Lab -2
Subject Code: 22ITP-351

Problem 1: Max Units on a Truck

1. **Problem Statement:** To determine the maximum number of units that can be loaded onto a truck given a set of boxes with different unit counts and a weight limit.

2. **Objective:**

I. To Implement a greedy algorithm to maximize the units loaded within the weight constraint.

II. To develop a strategy to count the necessary operations (increments) to achieve the desired array state.

III. To Create an efficient algorithm to calculate the maximum stones that can be retained.

IV. To Implement a dynamic programming or greedy approach to maximize the score.

V. To Formulate an algorithm that efficiently computes the required operations.

VI. To Develop a greedy or combinatorial approach to maximize task assignments.

3. Code:

```
class Solution:
    def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        heap = [[-units, -box] for box, units in boxTypes]
        heapify(heap)
        totalUnits = 0

        while heap and truckSize != 0:
            units, boxes = heappop(heap)
            units, boxes = -units, -boxes

            if boxes < truckSize:
                totalUnits += (boxes * units)
                truckSize -= boxes
            elif boxes >= truckSize:
                totalUnits += (truckSize * units)
                truckSize = 0

        return totalUnits
```

4. Output:

Fig 1: Output for Problem 1

Problem 2: Min Operations to make array increasing

1. **Problem Statement:** To find the minimum number of operations required to make an array strictly increasing.

2. **Code:**

```
class Solution:

    def minOperations(self, nums: List[int]) -> int:
        count = 0

        for i in range(1,len(nums)):
            if nums[i] <= nums[i-1]:

                x = nums[i]
                nums[i] += (nums[i-1] - nums[i]) + 1
                count += nums[i] – x

        return count
```

3. Output:

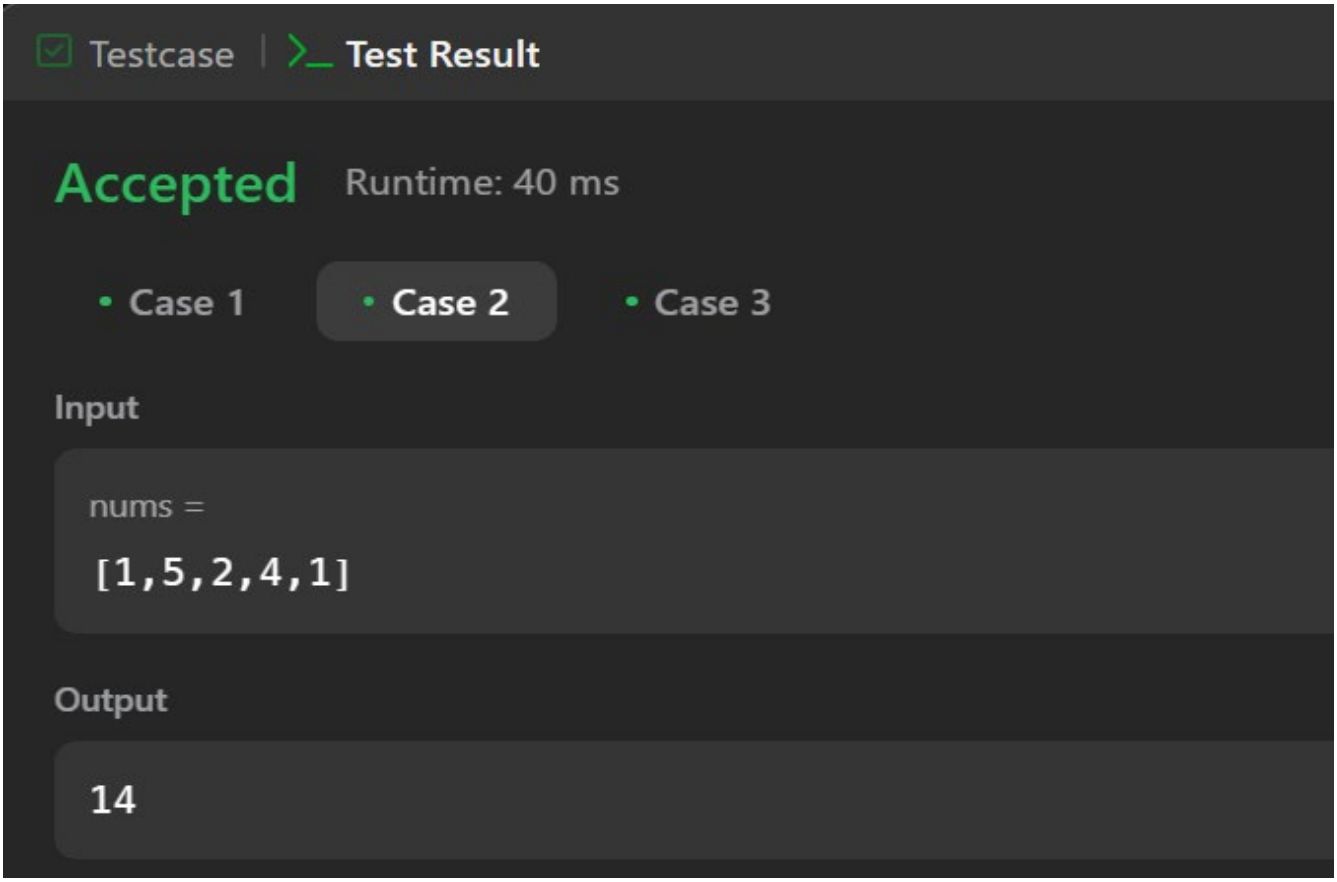

Fig 2: Output for Problem 2

Problem 3: Remove stones to Maximize total

1. **Problem Statement:** To maximize the total number of stones remaining after performing a series of removal operations.

2. **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)
```

3. Output:

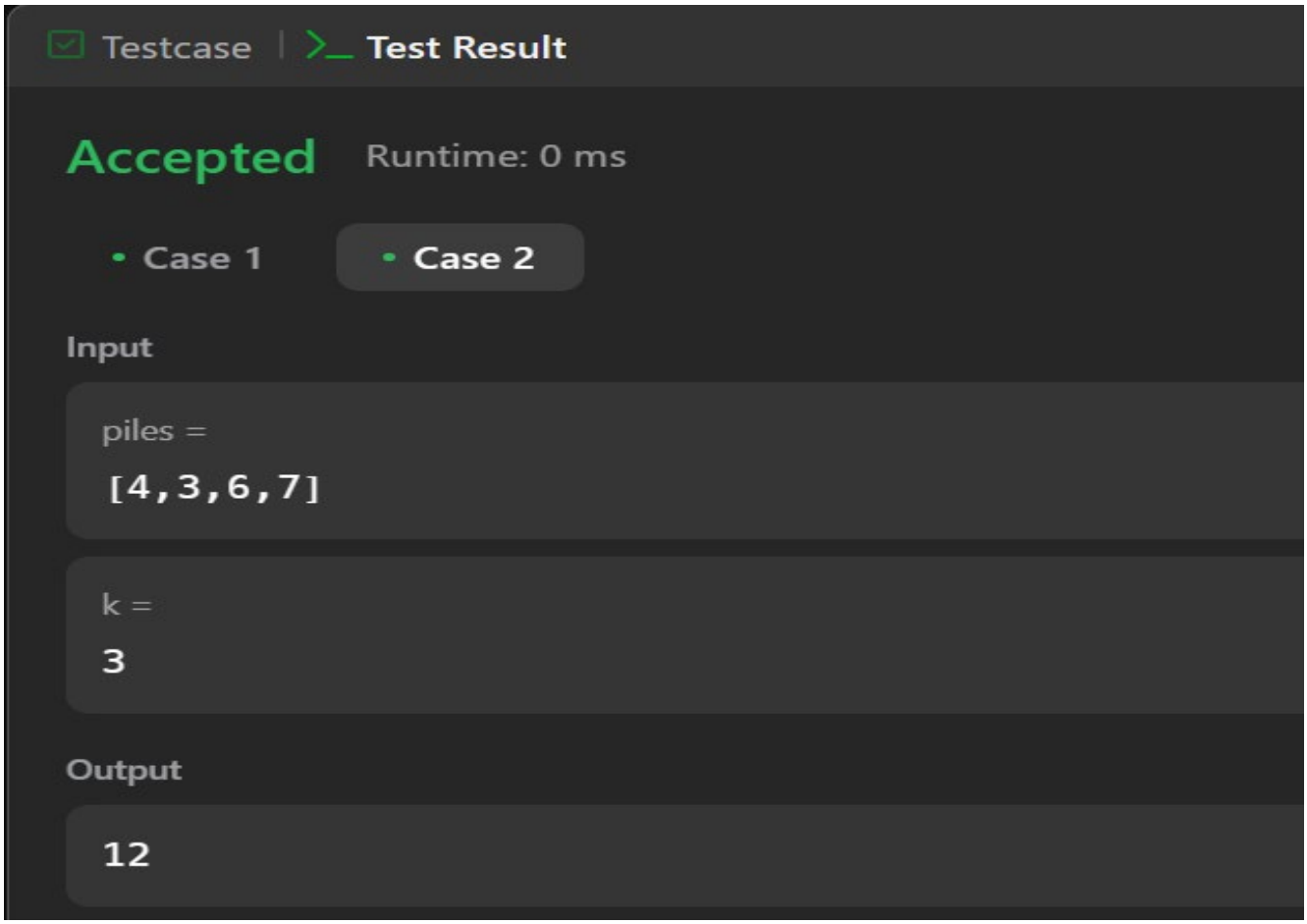

Fig 3: Output for Problem 3

Problem 4: Max Score from removing substrings

1. **Problem Statement:** To compute the maximum score obtainable by strategically removing substrings from a given string.

2. **Code:**

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

        if x >= y:
            s, points = remove_and_score(s, 'a', 'b', x)
            s, additional_points = remove_and_score(s, 'b', 'a', y)
            points += additional_points
        else:
```

```
        s, points = remove_and_score(s, 'b', 'a', y)
        s, additional_points = remove_and_score(s, 'a', 'b', x)
        points += additional_points

    return points
```

3. Output:

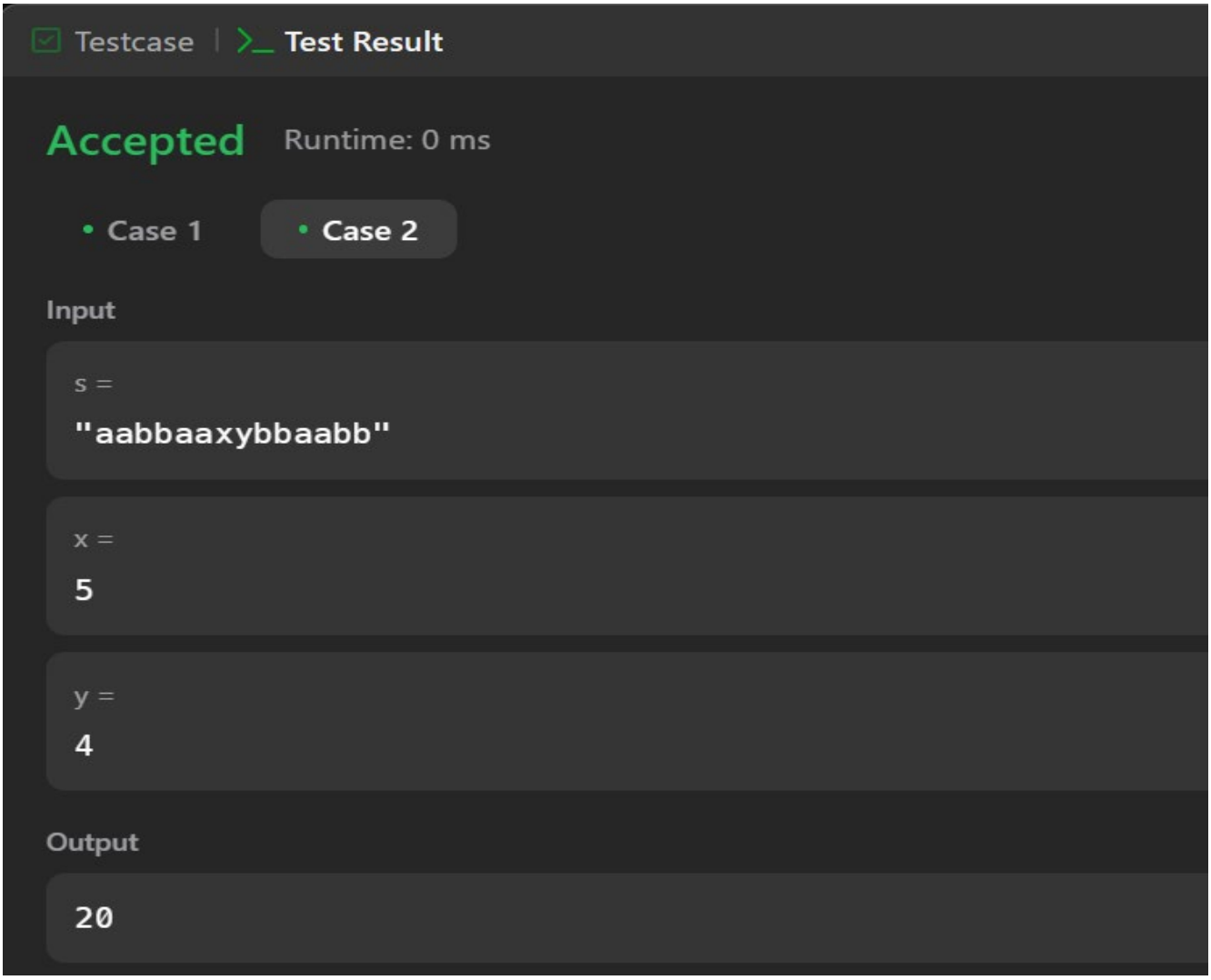

Fig 4: Output for Problem 4

Problem 5: Min operations to make a subsequence

1. **Problem Statement:** To determine the minimum number of operations needed to convert an array into a specific subsequence.

2. **Code:**

```
from bisect import bisect_left

class Solution:
    def minOperations(self, target: list[int], arr: list[int]) -> int:
        target_index_map = {num: i for i, num in enumerate(target)}

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

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

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

3. Output:

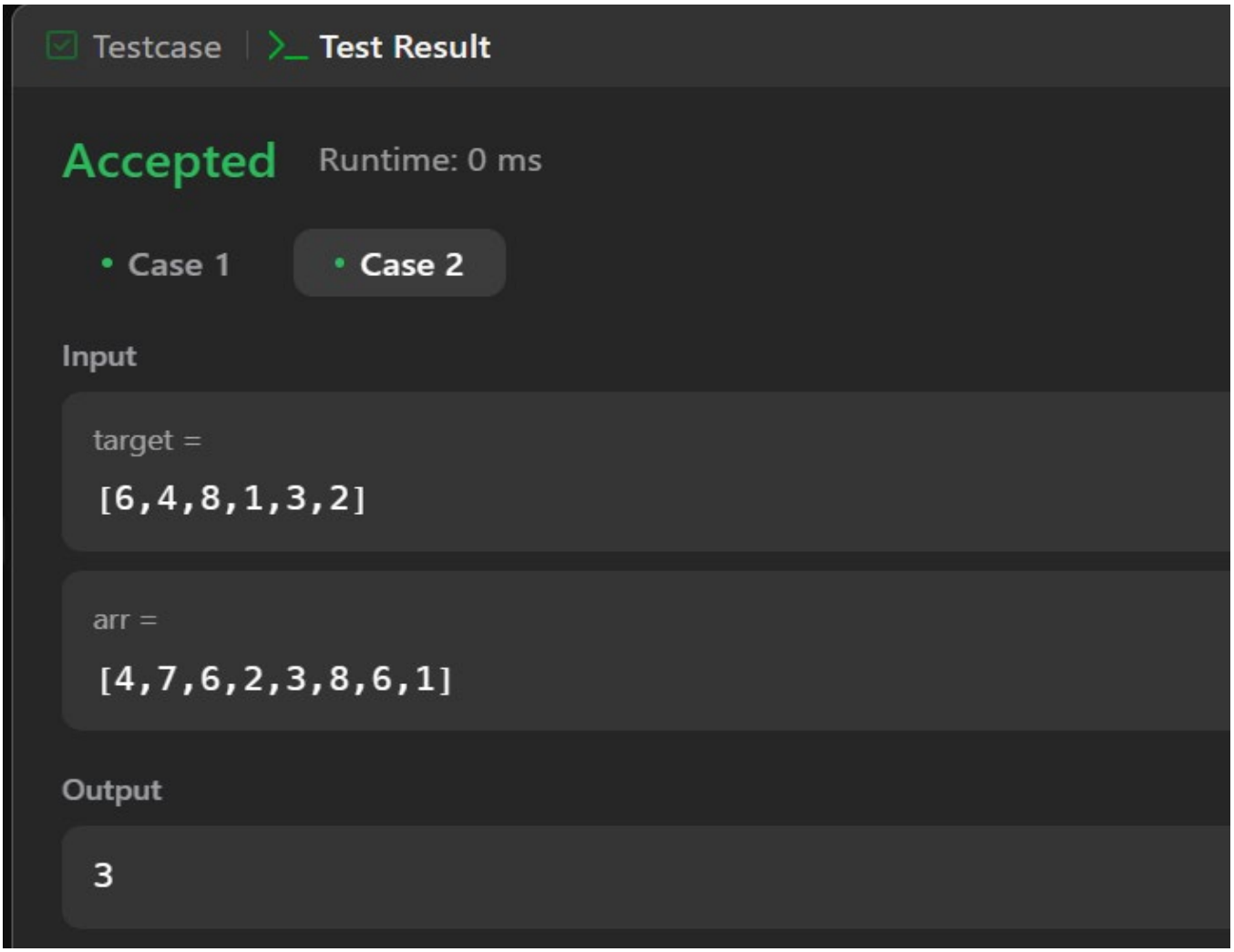

Fig 5: Output for Problem 5

Problem 6: Max number of tasks you can assign

1. **Problem Statement:** To find the maximum number of tasks that can be assigned to workers based on their capabilities and task requirements.

2. **Code:**

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

        def can_assign(n):
            task_i = 0
            task_temp = deque()
            n_pills = pills
            for i in range(n-1,-1,-1):
                while task_i < n and tasks[task_i] <= workers[i]+strength:
                    task_temp.append(tasks[task_i])
                    task_i += 1

                if len(task_temp) == 0:
                    return False
                if workers[i] >= task_temp[0]:
                    task_temp.popleft()
                elif n_pills > 0:
                    task_temp.pop()
                    n_pills -= 1
                else:
                    return False
```

```
        return True

    tasks.sort()
    workers.sort(reverse = True)

    l = 0
    r = min(len(tasks), len(workers))
    res = -1

    while l <= r:
        m = (l+r)//2
        if can_assign(m):
            res = m
            l = m+1
        else:
            r = m-1

    return res
```

3. Output:

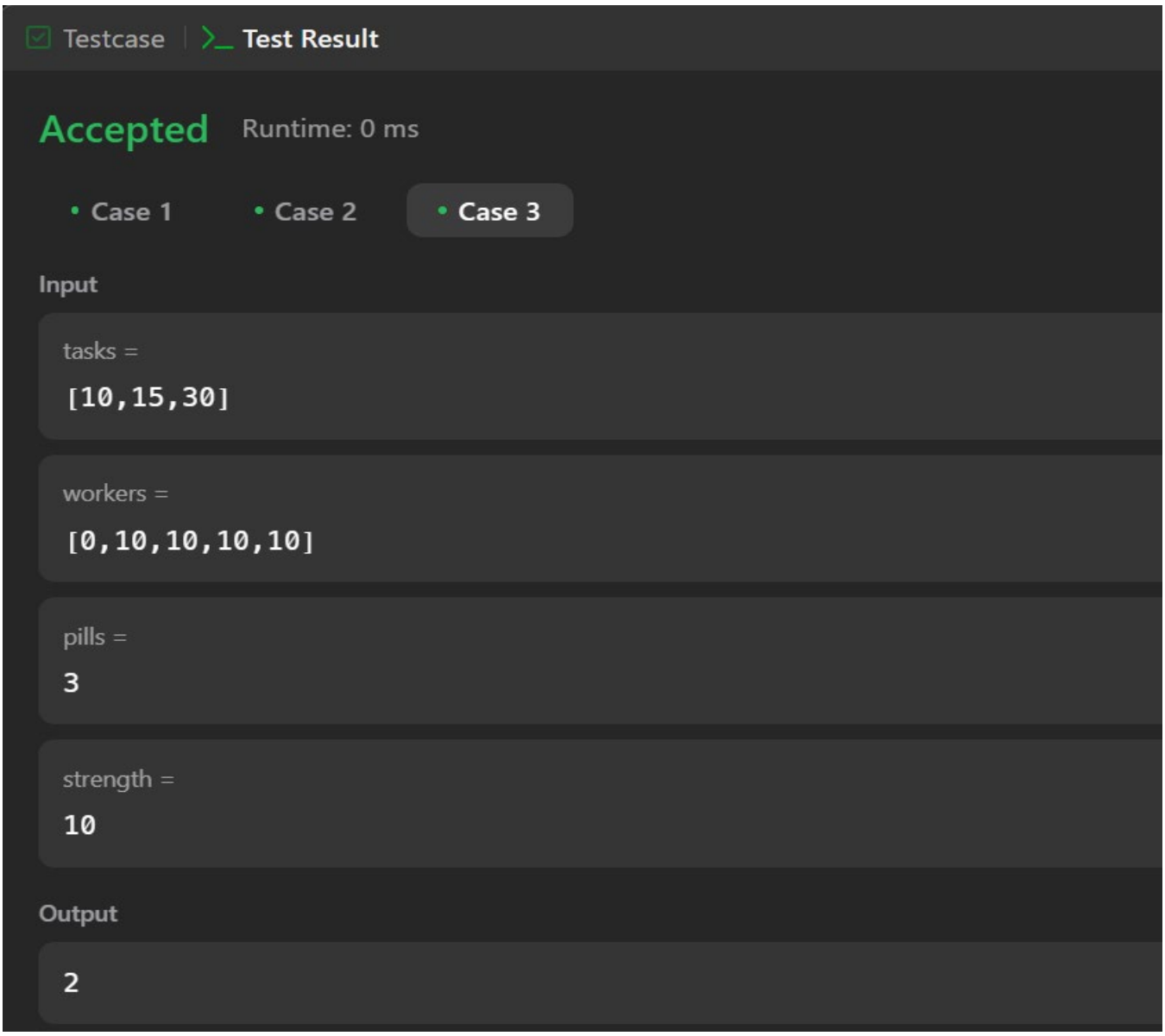

Fig 6: Output for Problem

4. Learning Outcome:

1. Enhance my problem-solving skills by analyzing complex problems and breaking them down into manageable components.
2. Develop a deeper understanding of various algorithmic paradigms, including greedy algorithms, dynamic programming, and combinatorial approaches.
3. Gained experience in utilizing and manipulating various data structures, such as arrays, lists, and heaps, to optimize algorithm performance.
4. Became proficient in analyzing the time and space complexity of my solutions, allowing me to evaluate algorithm efficiency and make informed decisions.
5. Improved my debugging skills and learn to create comprehensive test cases, including edge cases, to ensure the correctness of my solutions.