

Experiment: 9

Student Name: Shagun Sharma

UID: 22BCS11241

Branch: CSE

Section/Group: IOT-628/B

Semester: 6th

Date of Performance: 21/04/2025

Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSP-351

Problem -1

1. Aim: Group the People Given the Group Size They Belong To.

2. Objective: There are n people, each with a unique ID from 0 to $n-1$. You are given an integer array `groupSizes`, where `groupSizes[i]` represents the size of the group that person i is in. For example, if `groupSizes[1] = 3`, then person 1 must be in a group of size 3. Return a list of groups such that each person i is in a group of size `groupSizes[i]`. Each person should appear in exactly one group. If there are multiple answers, return any of them.

3. Algorithm:

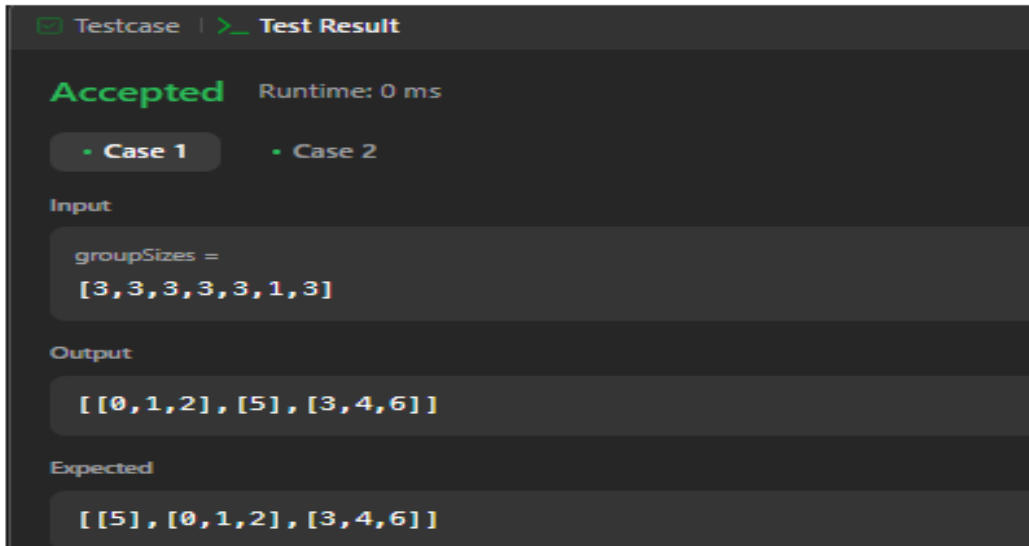
- Step 1: Iterate through the `groupSizes` and group people by their group size.
- Step 2: Once a group reaches the desired size, add it to the result and start a new group.

4. Implementation/Code:

```
Code | Accepted x
Python3 v Auto

1 class Solution:
2     def groupThePeople(self, groupSizes: List[int]) -> List[List[int]]:
3         from collections import defaultdict
4
5         size_to_people = defaultdict(list)
6         result = []
7
8         for person, size in enumerate(groupSizes):
9             size_to_people[size].append(person)
10            # If enough people are collected for this group size, form the group
11            if len(size_to_people[size]) == size:
12                result.append(size_to_people[size])
13                size_to_people[size] = []
14
15        return result
16
```

5. Output:



The screenshot shows a test result interface with a dark theme. At the top, it says 'Testcase' and 'Test Result'. Below that, it says 'Accepted' in green and 'Runtime: 0 ms'. There are two tabs: 'Case 1' and 'Case 2'. Under 'Case 1', the 'Input' is 'groupSizes = [3,3,3,3,3,1,3]' and the 'Output' is '[[0,1,2],[5],[3,4,6]]'. The 'Expected' output is '[[5],[0,1,2],[3,4,6]]'.

Problem-2

1. **Aim:** Find the Difference

2. **Objective:** You are given two strings ss and tt. String tt is generated by randomly shuffling string ss and then adding one more letter at a random position. Return the letter that was added to tt.

3. **Algorithm:**

- Step 1: Use XOR to find the character that was added.
- Step 2: XOR all characters from both strings. The result will be the extra character.

4. **Time and Space Complexity:**

- $O(n)$, where n is the length of the string.
- $O(1)$, as we are using only a constant amount of space.

5. Implementation/Code:

```
Code | Accepted X
Python Auto
1 class Solution(object):
2     def findTheDifference(self, s, t):
3         """
4         :type s: str
5         :type t: str
6         :rtype: str
7         """
8         result = 0
9         for char in s:
10             result ^= ord(char)
11         for char in t:
12             result ^= ord(char)
13         return chr(result)
14
```

6. Output:

```
Testcase | Test Result
Accepted Runtime: 0 ms
• Case 1 • Case 2
Input
s =
"abcd"
t =
"abcde"
Output
"e"
Expected
"e"
```

Problem: - 3

1. **Aim:** Predict the Winner

2. **Objective:** You are given an integer array `nums`. Two players are playing a game with this array: player 1 and player 2. Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array (i.e., `nums[0]` or `nums[nums.length - 1]`) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array. Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

3. **Algorithm:**

Step 1: Use dynamic programming to store the optimal score difference at each subarray.

Step 2: Return whether player 1's score difference is greater than or equal to zero at the full array level.

4. **Time and Space Complexity:**

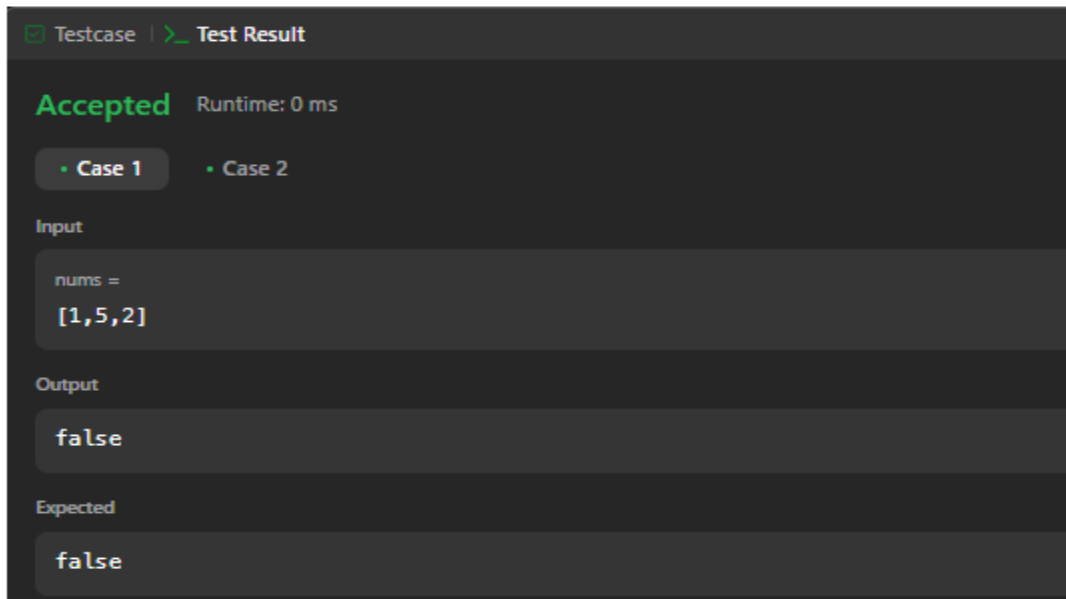
- $O(n^2)$, due to the DP approach filling in an $n \times n$ table.
- $O(n^2)$, for the DP table.

5. **Implementation/Code:**

```
</> Code | Accepted x
Python3 v Auto

1 class Solution:
2     def predictTheWinner(self, nums: List[int]) -> bool:
3         from functools import lru_cache
4
5         @lru_cache(None)
6         def dp(start, end):
7             if start == end:
8                 return nums[start]
9             # Player chooses either start or end and minimizes the opponent's advantage
10            pick_start = nums[start] - dp(start + 1, end)
11            pick_end = nums[end] - dp(start, end - 1)
12            return max(pick_start, pick_end)
13
14        return dp(0, len(nums) - 1) >= 0
15
```

6. Output:



7. Learning Outcomes (All 3 Problems):

1. Understand and apply greedy, hash-based, and dynamic programming techniques to solve algorithmic problems efficiently.
2. Enhance logical reasoning and problem decomposition for implementing game theory and optimization strategies.
3. Develop the ability to manipulate data structures like arrays, strings, and hash maps for customized outputs.
4. Improve proficiency in designing algorithms that satisfy constraints like group formation, optimal choices, and string analysis.
5. Strengthen coding skills by solving diverse problems involving simulation, decision-making, and pattern recognition.