Experiment 1 A

Student Name: PARDEEP SINGH **UID: 22BCS16692**

Branch: CSE **Section/Group: Ntpp 602-A**

Semester: 6TH **Date of Performance:20/01/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Two SUM

2. AIM:

Given an array of integers nums and an integer target, return the indices of the two numbers such that they add up to target. Each input has exactly one solution, and you cannot use the same element twice.

3. Algorithm

- Create an empty dictionary num_map.
- Loop through nums, get each number num and its index i.
- For each num, calculate complement as target - num.
- If complement exists in num_map, return the indices;
- otherwise, update num_map with num and i.

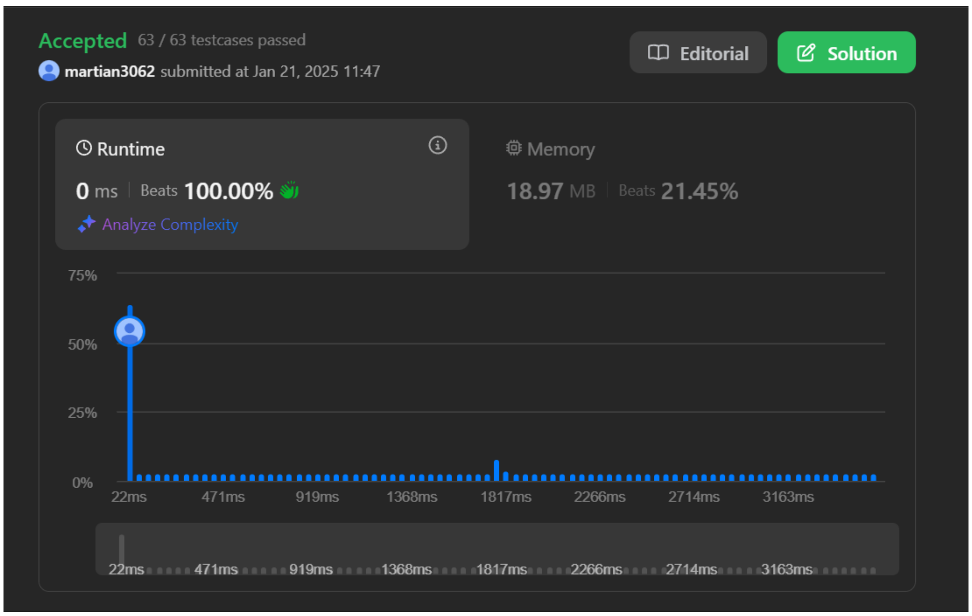
Implemetation/Code

```
class Solution:
    def twoSum(self, nums, target):
        num_map = {}
```

```
for i, num in enumerate(nums):
    complement = target - num
    if complement in num_map:

        return [num_map[complement], i]
        num_map[num] = i
```

Output



Time Complexity : O(n)

Space Complexity : O(n)

Learning Outcomes:-

- Understand dictionary operations for efficient data lookup.
- Develop skills in creating single-pass algorithms for problem solving.

Experiment 1 B

Student Name: PARDEEP SINGH **UID: 22BCS16692**

Branch: CSE **Section/Group: Ntpp 602-A**

Semester: 6TH **Date of Performance:20/01/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Jump Game II

2. AIM:

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

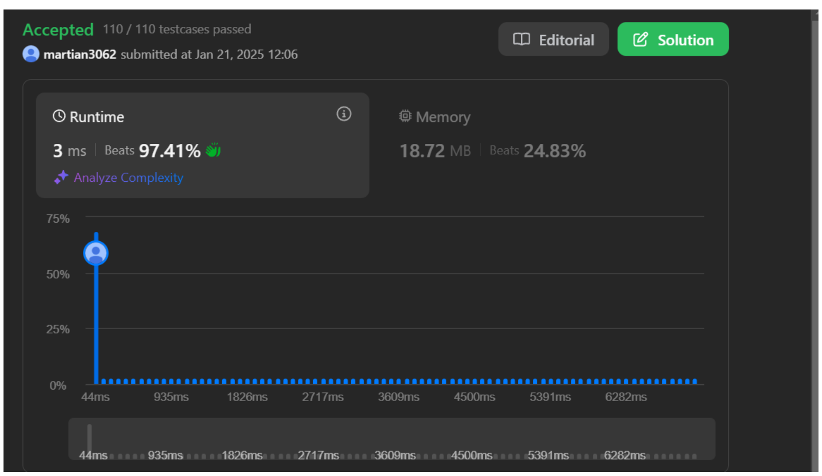
3. Algorithm

1. Handle the base case where n == 1 by checking if x == 1.
2. Initialize DP states dpX and dpNotX based on whether x is 1.
3. Iterate from position 2 to n, updating dpX and dpNotX using transition rules.
4. At each step, set dpX_new to dpNotX and dpNotX_new to (dpX * (k - 1) + dpNotX * (k - 2)) % MOD.
5. Return dpX as the final count of valid arrays ending with x.

Implemetation/Code:

```
class Solution:
    def jump(self, nums):
        n = len(nums)
        if n <= 1:
            return 0
        jumps = 0
        current = 0
        farthest = 0
        for i in range(n):
            farthest = max(farthest, i + nums[i])
            if i == current:
                jumps += 1
                current = farthest
                if current >= n - 1:
                    break
        return jumps
```

Output



Time Complexity : O(n)

Space Complexity : O(1)

Learning Outcomes:-

- Utilize greedy strategies for efficient problem-solving.
- Achieving optimal time and space complexities in algorithms.