Experiment-6(A)

Student Name: Masud Alom **UID:** 22BCS16095
Branch: CSE **Section/Group:** NTPP_602-A
Semester: 6 **Date of Performance:** 20-02-25
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSH-359

1. **Title:** Dynamic Programming (Climbing Stairs)

2. **Objective:** The problem is to find the total number of distinct ways to reach the top of a staircase with n steps, where at each step, you can either climb 1 or 2 steps.

3. **Algorithm:**

- **Understanding the Problem:**
- For `n = 1`: Only 1 way → `[1]`
- For `n = 2`: Two ways → `[1,1]`, `[2]`
- For `n = 3`: Three ways → `[1,1,1]`, `[1,2]`, `[2,1]`
- For `n = 4`: Five ways → `[1,1,1,1]`, `[1,1,2]`, `[1,2,1]`, `[2,1,1]`, `[2,2]`
- **Pattern Identification:**
- The problem resembles the Fibonacci sequence: • **Ways(n) = Ways(n-1) + Ways(n-2)** • **Base cases:**
  `o Ways(1) = 1 o`
  `Ways(2) = 2`
- **Approach:**
- Initialize two variables `first = 1` (Ways to reach step 1) and `second = 2` (Ways to reach step 2).
- For steps from 3 to `n`, calculate the number of ways using:

```
python CopyEdit
current = first + second
first = second second =
current
```

- Return `second` as it will hold the answer for `n`.

4. Implementation/Code:

```
class Solution:     def climbStairs(self,
n: int) -> int:
        # Base cases
if n <= 2:
return n

        # Initialize the first two steps
first, second = 1, 2

        # Compute the ways to reach each step from 3 to n
for i in range(3, n + 1):                current = first +
second              first = second              second =
current

        return second
```

5. Output:

6. Time Complexity: O(N)

7. **Space Complexity:**O(1)

Discover. Learn. Empower.

Experiment 6(B)

1. **Title:** Jump Game

2. **Objective:** Determine if you can reach the last index of an array where each element represents the maximum jump length from that position.

3. **Algorithm:**

1.**Initialize:** A variable `maxReach` to `0`, representing the furthest index we can reach.
2.**Iterate:** Through each index `i` in the array:
- If `i > maxReach`, return `false` (i.e., current index is unreachable).
- Update `maxReach` as `max(maxReach, i + nums[i])`. o If `maxReach` is greater than or equal to the last index, return `true`.

Return: If the loop completes without returning, it means the last index is reachable, so return `true`.

4. **Implementation/Code:**

```
class Solution:     def canJump(self, nums:
list[int]) -> bool:
        maxReach = 0          for i in
range(len(nums)):              if i > maxReach:
return False              maxReach = max(maxReach,
i + nums[i])              if maxReach >= len(nums)
- 1:
                return True
return True
```

6. **Output:**

8. **Time Complexity:** O(N) 9.**Space Complexity:** O(1)

Experiment 6(C)

1. **Title:** Maximum Subarray

2. **Objective:** To find the contiguous subarray with the largest sum in a given integer array nums.

3. **Algorithm:**

- **Initialization:**
- `currentSum` = 0 (stores sum of the current subarray) • `maxSum` = `-infinity` (stores the maximum sum found so far)
- **Iteration through the array:**
- For each element `num` in `nums`:
  - Add `num` to `currentSum`. o Update `maxSum` to the maximum of `maxSum` and `currentSum`. o If `currentSum` becomes negative, reset it to 0 (discard the current subarray).
- **Result:**

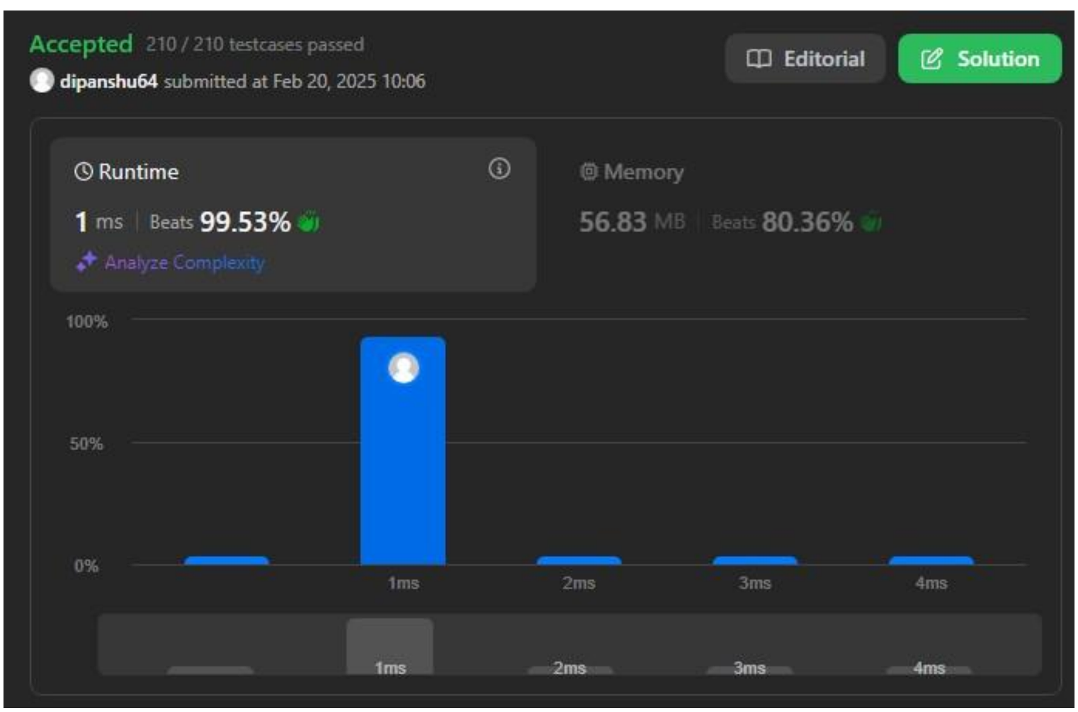
- Return `maxSum` as the maximum sum of the subarray.

5. Implementation/Code:

```
class Solution:     def maxSubArray(self, nums:
list[int]) -> int:
        currentSum = 0
maxSum = float('-inf')
                 for num
in nums:
            currentSum += num
            maxSum = max(maxSum, currentSum)
if currentSum < 0:
currentSum = 0

return maxSum
```

6. Output :



8. Time Complexity: O(N)

9. Space Complexity: O(1)

10. LearningOutcomes:

- **Kadane's Algorithm:** A powerful technique to solve maximum subarray problems in linear time.
- **Handling Negatives:** Resetting `currentSum` when it goes negative is key to maintaining the optimal subarray.
- **Optimized Approach:** Avoids nested loops, ensuring efficiency even for large input sizes.