



Experiment-2

Name: Tushar Pareenja

Branch: BE-CSE

Semester: 6th

Subject Name: AP LAB-II

UID: 22BCS16825

Section/Group: 627/A

Date of Performance 24/01/25

Subject Code: 22CSP-351

1. **Aim:** Two Sum

2. **Objective:** 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. You can return the answer in any order. Example 1:

Input: nums = [2,7,11,15], target = 9 Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

3. **Algorithm:**

- Initialize an empty hash map (dict).
- Iterate through the nums array:
- For each element num, calculate the complement: complement = target - num.
- Check if the complement exists in the hash map:
 - If it does, return the indices of the complement and the current number.
 - If it doesn't, add the current number and its index to the hash map.
- Return the indices of the two numbers that add up to the target.

4. **Implementation/Code:**

```
class Solution {
public int[] twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[i] + nums[j] == target) {
                return new int[] {i, j};
            }
        }
    }
    return new int[] {};
}
```

5. Output:

```
Testcase | Test Result
Accepted Runtime: 0 ms
• Case 1 • Case 2 • Case 3
Input
nums =
[2,7,11,15]
target =
9
Output
[0,1]
Expected
[0,1]
```

```
Testcase | Test Result
Accepted Runtime: 0 ms
• Case 1 • Case 2 • Case 3
Input
nums =
[3,2,4]
target =
6
Output
[1,2]
Expected
[1,2]
```

```
Testcase | Test Result
Accepted Runtime: 0 ms
• Case 1 • Case 2 • Case 3
Input
nums =
[3,3]
target =
6
Output
[0,1]
Expected
[0,1]
```

6. Learning Outcome:

- We Learn About the use of mapping.
 - We Learn About the use of vector.
 - We Learn About the use of indexing.
 - We learn About the ordered map.
 - We Learn About the Calling For the function.
7. **Time Complexity:** $O(n)$, where n is the number of elements in the `nums` array. This is because we only iterate through the array once.
8. **Space Complexity:** $O(n)$, since we store each number and its index in the hash map.

Question 2

1. **Aim:** Jump Game II.

2. **Objective:** 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: $0 \leq j \leq \text{nums}[i]$ 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]`.

Example 1:

Input: `nums = [2,3,1,1,4]`

Output: 2

Explanation: The minimum number of jumps to reach the last index is 2.

Jump 1 step from index 0 to 1, then 3 steps to the last index.

3. **Algorithm:**

- Initialize `jumps = 0`, `current_end = 0`, `farthest = 0`.
- Loop through indices `i` to `n - 2`, update `farthest = max(farthest, i + nums[i])`.
- If `i == current_end`, increment `jumps` and set `current_end = farthest`.
- Return `jumps`.

4. **Implementation/Code:**

```
class Solution {
    public int jump(int[] nums) {
        int jumps = 0, currentEnd = 0, farthest = 0;
        for (int i = 0; i < nums.length - 1; i++) {
            farthest = Math.max(farthest, i + nums[i]);
            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;
            }
        }
        return jumps;
    }
}
```

5. Output:

• Case 1 • Case 2

Input

nums =
[2,3,1,1,4]

Output

2

Expected

2

• Case 1 • Case 2

Input

nums =
[2,3,0,1,4]

Output

2

Expected

2

6. Learning Outcome:

- We Learn About the use of Running vector.
- We Learn About the use of vector.
- We Learn About the use of max function.
- We learn About the jumping into int indexes.
- We Learn About the Calling For the libraries.

7. Complexity:

- The time complexity of the algorithm is $O(n)$. The loop iterates through the array once, performing constant-time operations at each index.
- The space complexity of the algorithm is $O(1)$. It uses a constant amount of extra space for variables like jumps, current_end, and farthest.