

Experiment 2

Student Name: ANMOLPREET SINGH
Branch: BE-CSE
Semester: 5TH
Subject Name: -Advance Programming lab-2

UID: -22BCS15759
Section/Group: Ntp-602-A
Date of Performance: 15-1-25
Subject Code: -22CSP-351

Aim: -To solve the "Two Sum" and "Implement Queue Using Stacks" problems by designing efficient algorithms that meet the problem requirements while ensuring correct and optimal performance.

Objective: -The objective is to develop solutions for the "Two Sum" and "Queue Using Stacks" problems. For "Two Sum", implement an algorithm that finds two numbers adding up to a target. For "Queue Using Stacks", design a queue with push, pop, peek, and empty operations using two stacks, ensuring efficient FIFO behavior.

Apparatus Used:

1. **Software:** -Leetcode
2. **Hardware:** Computer with 4 GB RAM and keyboard.

Problem Statement(2.1): 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.

Algorithm for the Two Sum Problem:

1. **Input:** An array `nums[]` of integers and an integer `target`.
2. **Output:** Return a pair of indices `i` and `j` such that `nums[i] + nums[j] = target`.

Steps:

1. Start by determining the size `n` of the `nums[]` array.
2. Loop through the array using the first index `i` (from 0 to `n-2`).
3. Inside the first loop, use a second loop with index `j` (starting from `i + 1` to `n-1`).
4. For each pair of indices `(i, j)`, check if the sum of the elements at these indices, `nums[i] + nums[j]`, is equal to `target`.
5. If the sum is equal to `target`, return the indices `[i, j]`.
6. If no pair is found after checking all possible pairs, return an empty array `[]`.

Code:

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }
        return {};
    }
}
```

Time Complexity:

- The time complexity of this solution is $O(n^2)$ because of the nested loops where each element is compared with every other element in the array.

Space Complexity:

- The space complexity is $O(1)$ because we are using only a constant amount of space to store the result.

Output- All the test cases passed

Description | Accepted × | Editorial | Solutions | Submissions

← All Submissions

Accepted 63 / 63 testcases passed

22BSCS15759 submitted at Feb 24, 2025 10:48

Editorial Solution

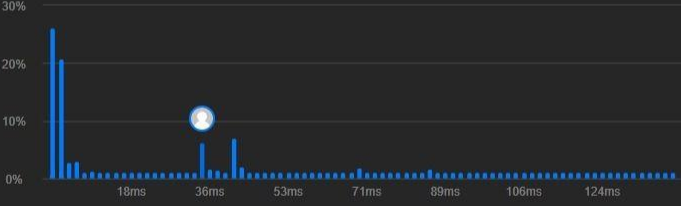
Runtime

34 ms | Beats 43.77%

Analyze Complexity

Memory

14.10 MB | Beats 76.83%



Code | C++

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
```

</> Code

C++ Auto

```
3 vector<int> twoSum(vector<int>& nums, int target) {
4     int n = nums.size();
5     for (int i = 0; i < n - 1; i++) {
6         for (int j = i + 1; j < n; j++) {
7             if (nums[i] + nums[j] == target) {
8                 return {i, j}; }
9         }
10    return {};
11 }
12 }
```

Saved Ln 12, Col 3

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =

[2, 7, 11, 15]

target =

9

Output

Problem Statement: You are given a 0-indexed array `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`. Return the minimum number of jumps to reach `nums[n - 1]`.

Algorithm for Implementing Queue Using Stacks:

1. Input:

A stack `s1` used for queue operations (FIFO behavior).

A stack `s2` used temporarily for rearranging elements in the right order.

Operations: `push(x)`, `pop()`, `peek()`, and `empty()`.

2. Operations:

Push Operation (`push(x)`):-While `s1` is not empty, pop all elements from `s1` and push them onto `s2` to reverse their order. Push the element `x` onto `s2`. While `s2` is not empty, pop all elements from `s2` and push them back onto `s1`. This restores the correct order for the queue (FIFO).

Pop Operation (`pop()`):-The front element of the queue is at the top of `s1`, so pop and return the top element of `s1`.

Peek Operation (`peek()`):-The front element of the queue is still at the top of `s1`, return the top element of `s1` without popping.

Empty Operation (`empty()`):-Return true if `s1` is empty, otherwise return false.

Code-

```
class Solution:

    def canJump(self, nums):

        farthest = 0

        for i in range(len(nums)):

            # If we can't reach this index, return False

            if i > farthest:

                return False

            # Update the farthest index we can reach

            farthest = max(farthest, i + nums[i])

            # If we can reach or exceed the last index, return True
            if farthest >= len(nums) - 1:

                return True

        return False

nums = [2, 3, 1, 1, 4]
result = Solution().canJump(nums)
print(result) # Output: True

nums = [3, 2, 1, 0, 4]
result = Solution().canJump(nums)
print(result) # Output: False
```

Time Complexity:

Push Operation: $O(n)$, where n is the number of elements in the queue, because elements are moved between $s1$ and $s2$.

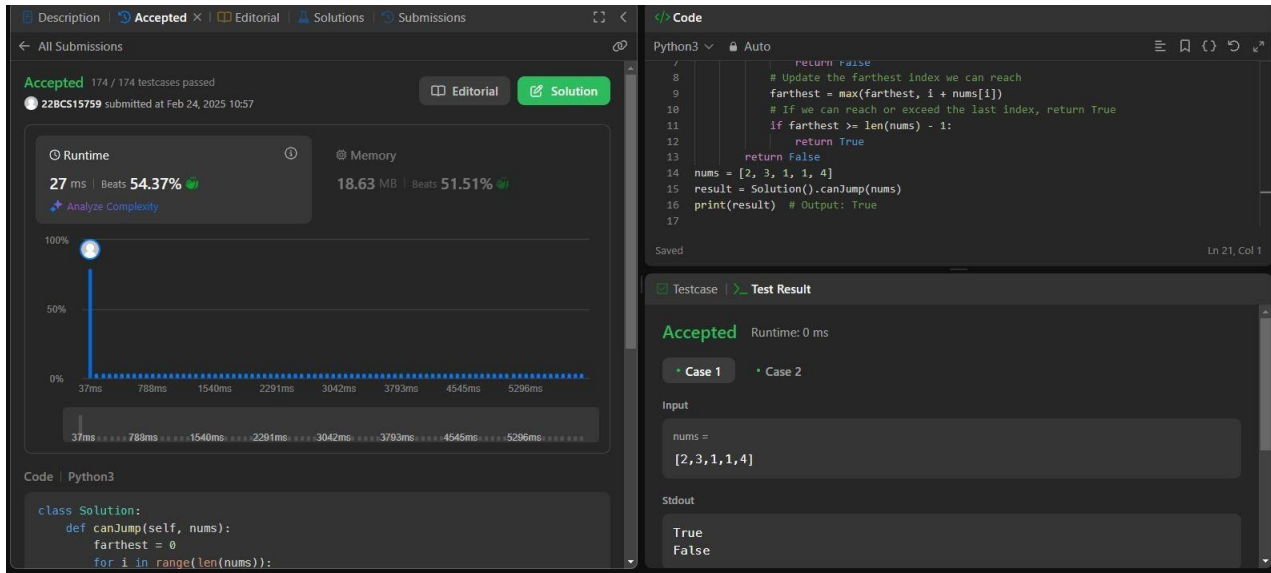
Pop Operation: $O(1)$, as it only involves popping the top element of $s1$.

Peek Operation: $O(1)$, as it only involves looking at the top element of $s1$.

Empty Operation: $O(1)$, as it checks if $s1$ is empty.

Space Complexity: $O(n)$, where n is the number of elements in the queue, because both stacks $s1$ and $s2$ hold the elements in the queue at any time.

Result-All test cases passes



Learning Outcomes:

1. Understand the use of two stacks to implement a FIFO queue and array syntaxes.
2. Learn how to manage stack operations to mimic queue behavior.
3. Develop skills in designing efficient algorithms for data structure manipulation.
4. Analyze the time and space complexity of array solutions.
5. Apply problem-solving techniques to implement standard queue operations (push, pop, peek, empty).