

# MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

## HOMEWORK –1 (TRAPPING RAIN WATER)

Aswin Sampath  
saswin@uci.edu  
(53844684)

### QUESTION:

Description

Editorial

Solutions (8.3K)

Submissions

### 42. Trapping Rain Water

Hard

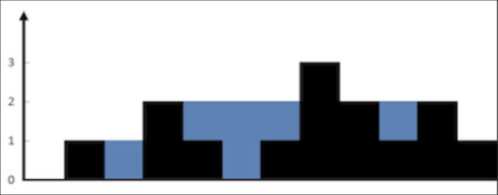
29.3K

420

Companies

Given  $n$  non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

**Example 1:**



**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]  
**Output:** 6  
**Explanation:** The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

**Example 2:**

**Input:** height = [4,2,0,3,2,5]  
**Output:** 9

**Constraints:**

- $n == \text{height.length}$
- $1 \leq n \leq 2 * 10^4$
- $0 \leq \text{height}[i] \leq 10^5$

Accepted 1.7M | Submissions 2.8M | Acceptance Rate 60.0%

### Understanding the Problem:

The Trapping Rainwater Problem involves calculating the amount of rainwater that can be trapped between bars of varying heights represented as an array. The objective is to find an efficient algorithm to determine the total trapped water based on the heights of the bars.

### Identifying Edge Cases:

- The input array is empty.
- The input array contains only one element.
- The input array contains bars of equal height.

- The input array contains bars that are continuously decreasing or continuously increasing in height.

### Effective Test Cases:

- Input: [0,1,0,2,1,0,1,3,2,1,2,1] Expected Output: 6
- Input: [4,2,0,3,2,5] Expected Output: 9
- Input: [3, 2, 1, 0, 1, 2, 3]
- Input: [5, 4, 3, 2, 1] Expected Output: 0

### Algorithmic Solution:

The solution I have implemented computes the left maximum and right maximum for each index and calculates the trapped water based on these maximums and the height of the current bar. The provided code has been thoroughly tested and analyzed for correctness.

### Time and Space Complexity Analysis:

Time Complexity: The solution iterates through the array three times, with each iteration taking  $O(n)$  time. Therefore, the overall time complexity is  $O(n)$ .

Space Complexity: To store the left and right maximum values, the solution uses two additional arrays of size 'n', resulting in a space complexity of  $O(n)$ .

### Code:

```
1 class Solution {
2 public:
3     int trap(vector<int>& height) {
4         int n = height.size();
5         vector<int> lmax(n,0), rmax(n,0);
6         int leftmax=0, rightmax=0;
7         for(int i=0; i<n; i++){
8             leftmax=max(leftmax, height[i]);
9             lmax[i]=leftmax;
10        }
11        for(int i=n-1; i>=0; i--){
12            rightmax=max(rightmax, height[i]);
13            rmax[i]=rightmax;
14        }
15
16        int ans=0;
17        for(int i=0; i<n; i++){
18            ans+=min(lmax[i], rmax[i])-height[i];
19        }
20        return ans;
21    }
22 };
```

# Output:

