Description | Accepted | Editorial | Solutions | Submissions

42. Trapping Rain Water

Solved

Hard | Topics | 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
```

35.9K | 538 | 536 Online

Code

Java | Auto

```java
class Solution {
    public int trap(int[] h) {
        int l = 0, r = h.length - 1;
        int lmax = 0, rmax = 0;
        int water = 0;

        while (l < r) {
            if (h[l] < h[r]) {
                if (h[l] >= lmax)
                    lmax = h[l];
                else
                    water += lmax - h[l];
                l++;
            } else {
                if (h[r] >= rmax)
                    rmax = h[r];
```

Saved | Ln 21, Col 10

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

height =
[0,1,0,2,1,0,1,3,2,1,2,1]

Output

6

Description | Accepted | Editorial | Solutions | Submissions

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

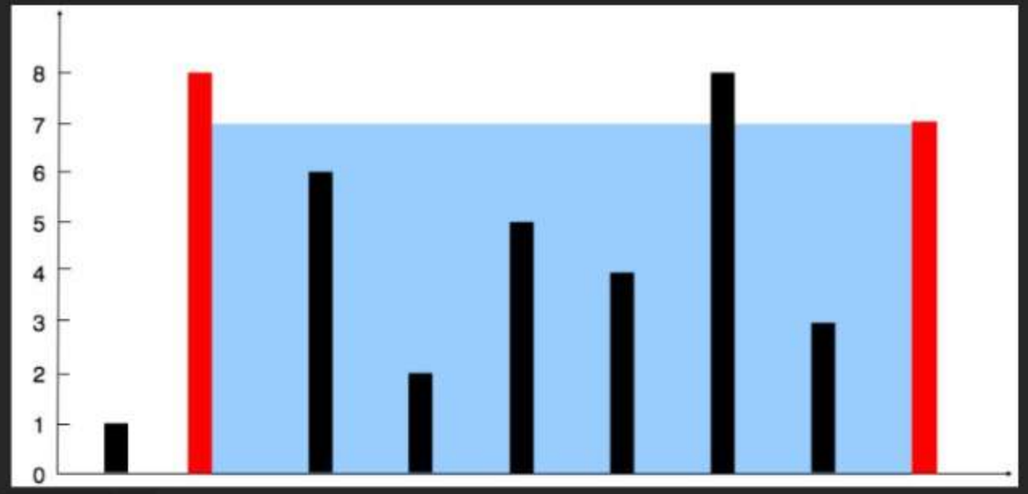
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store.*

Notice that you may not slant the container.

Example 1:



33.6K | 840 | 595 Online

Code

Java | Auto

```java
class Solution {
    public int maxArea(int[] h) {
        int l = 0, r = h.length - 1;
        int max = 0;

        while (l < r) {
            int area = Math.min(h[l], h[r]) * (r - l);
            max = Math.max(max, area);

            if (h[l] < h[r])
                l++;
            else
                r--;
        }
        return max;
    }
}
```

Saved | Ln 17, Col 2

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

height =
[1,8,6,2,5,4,8,3,7]

Output

49

Description | Accepted | Editorial | Solutions | Submissions

26. Remove Duplicates from Sorted Array

Solved

Easy | Topics | Companies | Hint

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Consider the number of *unique elements* in `nums` to be `k`. After removing duplicates, return the number of unique elements `k`.

The first `k` elements of `nums` should contain the unique numbers in **sorted order**. The remaining elements beyond index `k - 1` can be ignored.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Code

Java | Auto

```java
class Solution {
    public int removeDuplicates(int[] a) {
        if (a.length == 0) return 0;

        int j = 0;

        for (int i = 1; i < a.length; i++) {
            if (a[i] != a[j]) {
                j++;
                a[j] = a[i];
            }
        }
        return j + 1;
    }
}

```

Saved | Ln 16, Col 1

Testcase | Test Result

nums =

[1,1,2]

Output

[1,2]

Expected

[1,2]

Contribute a testcase