Previous Next

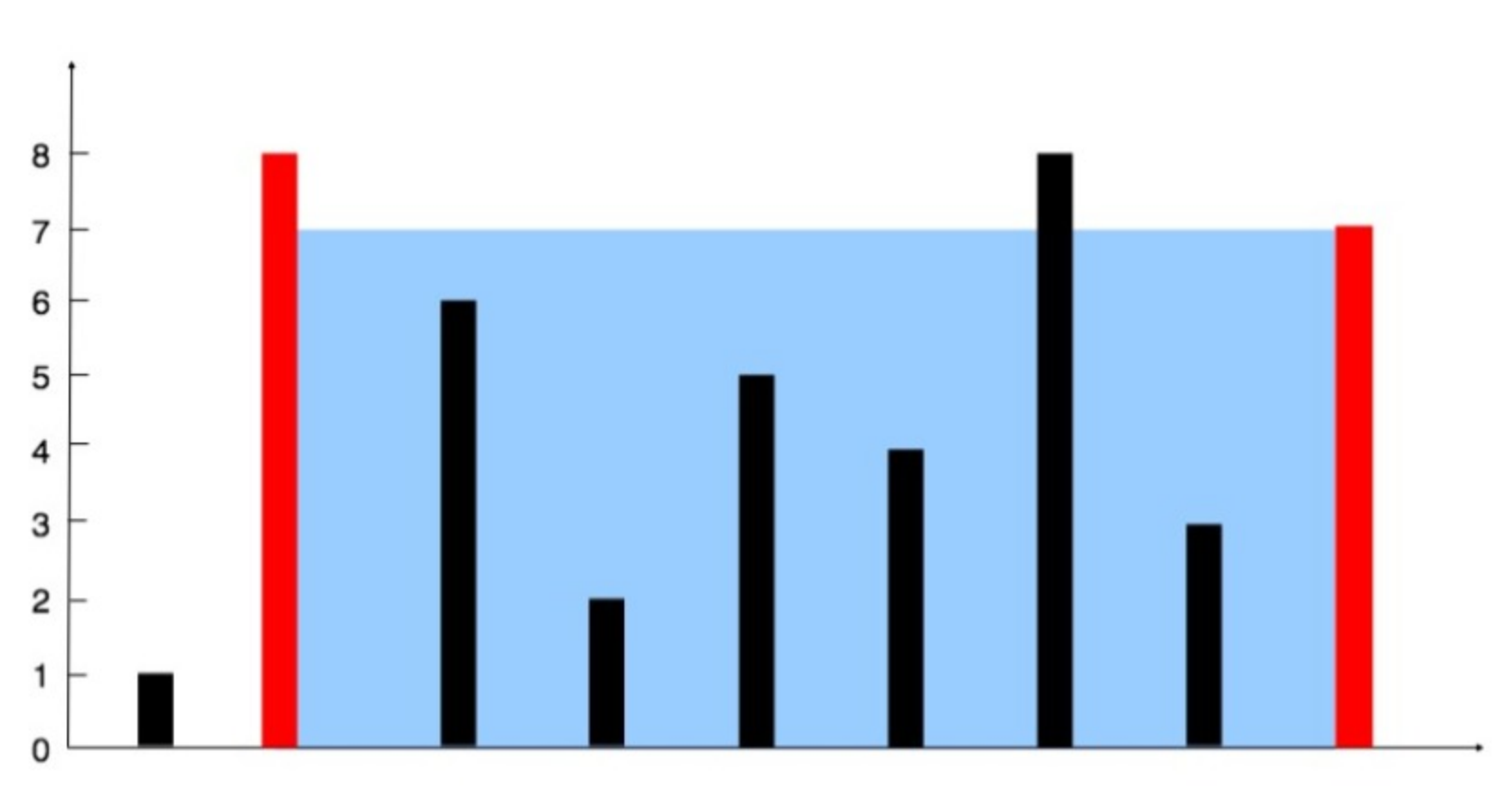
11. Container With Most Water

July 15, 2016 | 432.1K views

Average Rating: 4.78 (305 votes)

Given n non-negative integers $a_1, a_2, ..., a_n$, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.



The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example:

```
Input: [1,8,6,2,5,4,8,3,7]
Output: 49
```

Summary

We have to maximize the Area that can be formed between the vertical lines using the shorter line as length and the distance between the lines as the width of the rectangle forming the area.

Solution

Approach 1: Brute Force

Algorithm

In this case, we will simply consider the area for every possible pair of the lines and find out the maximum area out of those.

Java | Copy

```java
public class Solution {
    public int maxArea(int[] height) {
        int maxarea = 0;
        for (int i = 0; i < height.length; i++)
            for (int j = i + 1; j < height.length; j++)
                maxarea = Math.max(maxarea, Math.min(height[i], height[j]) * (j - i));
        return maxarea;
    }
}
```

Complexity Analysis

- Time complexity : $O(n^2)$. Calculating area for all $\frac{n(n-1)}{2}$ height pairs.
- Space complexity : $O(1)$. Constant extra space is used.

Approach 2: Two Pointer Approach

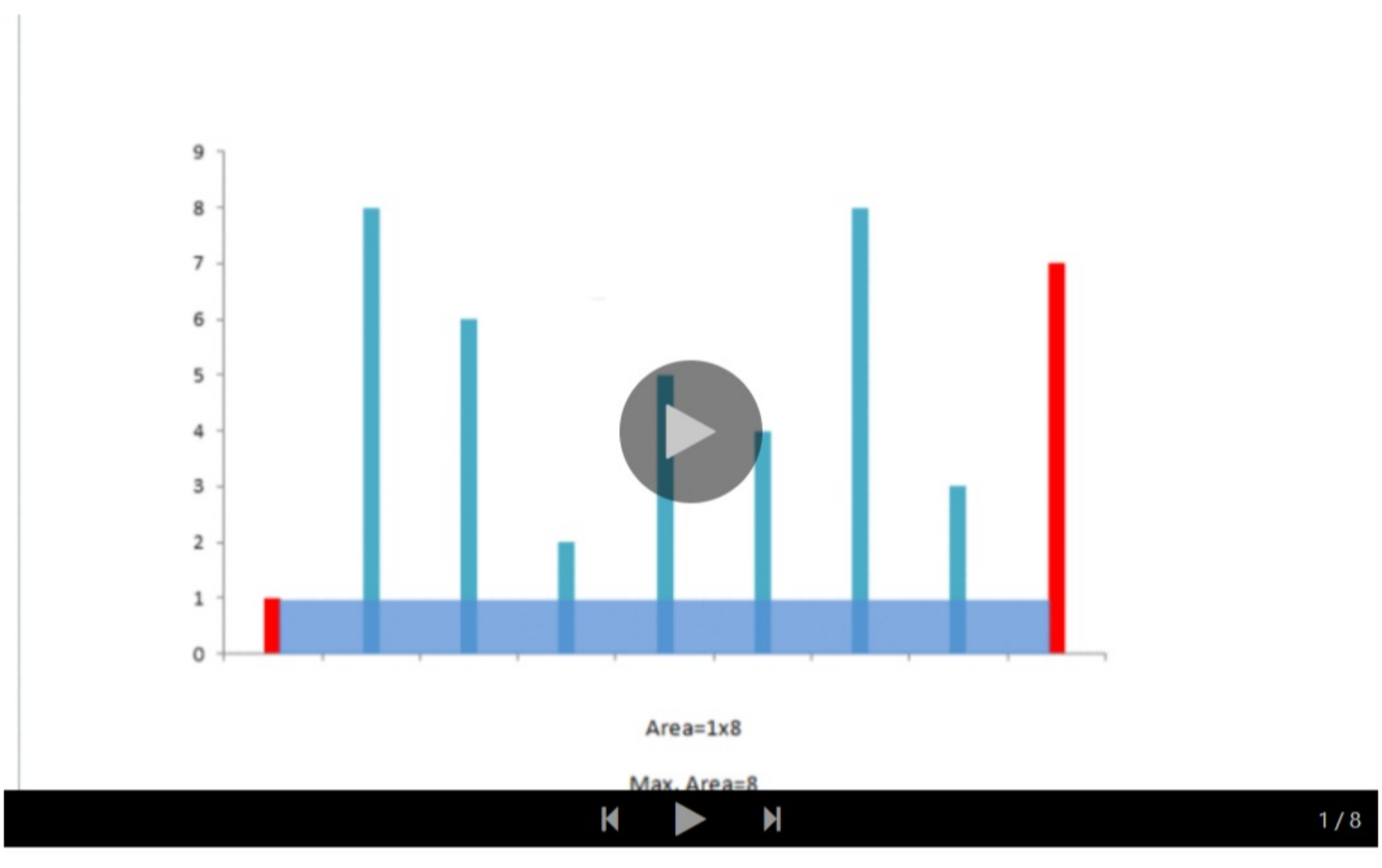
Algorithm

The intuition behind this approach is that the area formed between the lines will always be limited by the height of the shorter line. Further, the farther the lines, the more will be the area obtained.

We take two pointers, one at the beginning and one at the end of the array constituting the length of the lines. Futher, we maintain a variable maxarea to store the maximum area obtained till now. At every step, we find out the area formed between them, update maxarea and move the pointer pointing to the shorter line towards the other end by one step.

The algorithm can be better understood by looking at the example below:

```
1 8 6 2 5 4 8 3 7
```



How this approach works?

Initially we consider the area constituting the exterior most lines. Now, to maximize the area, we need to consider the area between the lines of larger lengths. If we try to move the pointer at the longer line inwards, we won't gain any increase in area, since it is limited by the shorter line. But moving the shorter line's pointer could turn out to be beneficial, as per the same argument, despite the reduction in the width. This is done since a relatively longer line obtained by moving the shorter line's pointer might overcome the reduction in area caused by the width reduction.

For further clarification click here and for the proof click here.

Java | Copy

```java
public class Solution {
    public int maxArea(int[] height) {
        int maxarea = 0, l = 0, r = height.length - 1;
        while (l < r) {
            maxarea = Math.max(maxarea, Math.min(height[l], height[r]) * (r - l));
            if (height[l] < height[r])
                l++;
            else
                r--;
        }
        return maxarea;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. Single pass.
- Space complexity : $O(1)$. Constant space is used.

Rate this article: ★ ★ ★ ★ ★

Previous Next

Comments: 176 Sort By

Type comment here... (Markdown is supported)

Preview Post

zylatis ★243 November 23, 2018 9:54 AM

I found a lot of the discussion and proof about this quite opaque, but one thing helped it finally clicked for me (which is sort of proof by contradiction i guess)

You have two heights H_left and H_right, and H_right < H_left, then we know we have two choices, we want to move one of them. If we move the larger one, we *cannot* increase the height for the simple

Read More

239 Share Reply

SHOW 12 REPLIES

vipkk67 ★89 August 12, 2018 12:21 PM Report

```
int maxArea(vector<int> &height) {
    int m = INT32_MIN;
    for (int i = 0, j = height.size() - 1; i < j;) {
        m = max(m, (j - i) * min(height[i], height[j]));
```

Read More

51 Share Reply

davidhuangdw ★78 March 8, 2019 6:33 PM

A simple proof:

1. case hi <= hj:

- we can prove that j is the best choice(within the range from i to j) for i

Read More

40 Share Reply

SHOW 3 REPLIES

jbi3 ★24 November 30, 2018 4:48 AM Report

it doesn't make sense when left height and right height are the same and do a right--. for example: [1,2,100,1,1,1,1,1,2].

24 Share Reply

SHOW 3 REPLIES

XiangkunYe ★69 September 4, 2018 8:14 AM

python3 solution:

class Solution:
def maxArea(self, height):
"""

Read More

34 Share Reply

smarsu ★79 March 11, 2018 12:30 PM Report

if the array is 1 1 9 1 or 1 9 1 1, would the algorithm be right?

27 Share Reply

SHOW 1 REPLY

phantom1911 ★13 September 18, 2018 10:20 PM

What happens when height[l] == height[r]? How to decide what to do in this case l++ or r-- ?

13 Share Reply

SHOW 6 REPLIES

IndigoBeast ★71 September 28, 2018 2:33 PM

I made a formal proof of the algorithm. For those who are skeptical about it, please take a look. Maybe my explanation helps.

10 Share Reply

SHOW 4 REPLIES

songlei1994 ★6 January 3, 2019 7:27 PM

We use **S[i][j]** to record the area between a[i] and a[j].
At the begin,if a[0]<= a[n],we consider the max area of the subarray a[1:n],
Why can we discard the conditon of **S[0][i] ,i<= n-1**,that because

S[0][i] = i * min(a[0],a[i]) <= i * a[0] < n * a[0] = S[0][n]

Read More

6 Share Reply

SHOW 1 REPLY

humy ★34 January 22, 2019 5:15 AM Report

The two pointer approach:
52 ms, faster than 99.60% of Python3

```
class Solution:
    def maxArea(self, height):
```

Read More

4 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 5 6 ... 17 18 ›