11. Container With Most Water

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 [Time Limit Exceeded]

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

```
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) [Accepted]

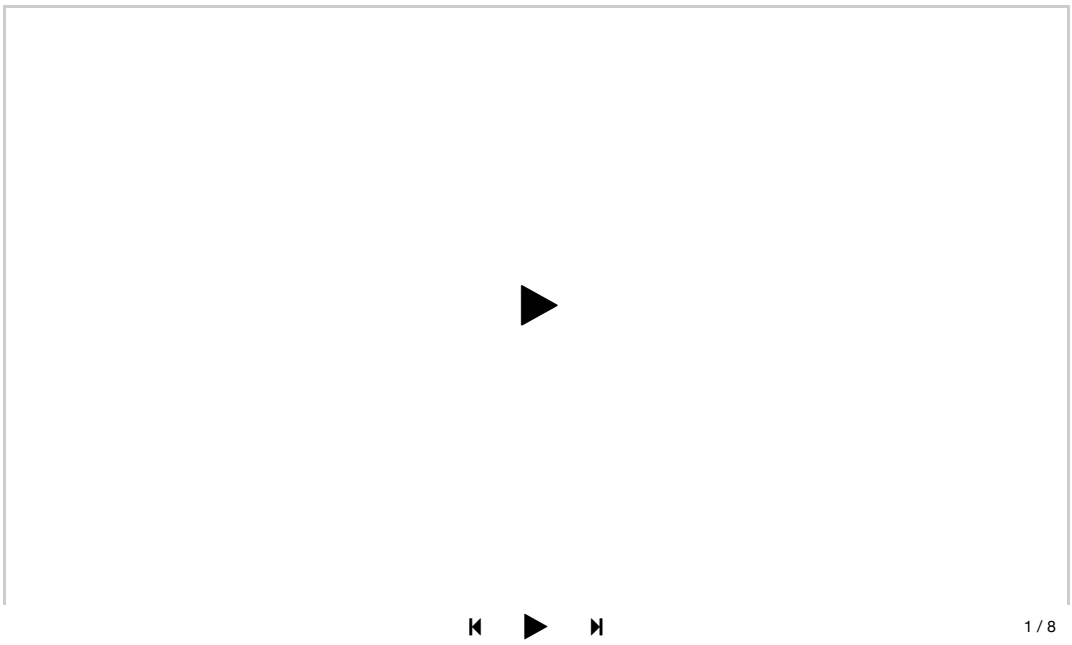
Algorithm

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 (https://discuss.leetcode.com/topic/3462/yet-another-way-to-see-what-happens-in-the-o-n-algorithm) and for the proof click here (https://discuss.leetcode.com/topic/503/anyone-who-has-a-o-n-algorithm/2).

Java Copy

Complexity Analysis

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

Analysis written by: @vinod23 (https://leetcode.com/vinod23)

Join the conversation