# Container With Most Water

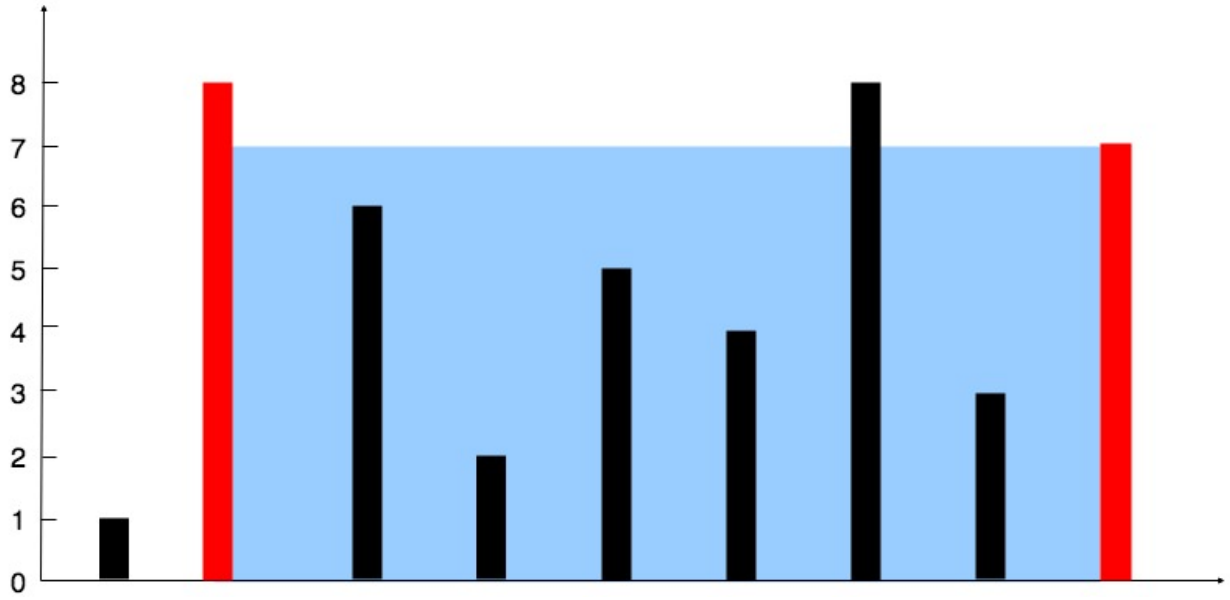
| | |
|---|---|
| Difficulty | Medium |
| Category | Two Pointers |
| Question | https://leetcode.com/problems/container-with-most-water/ |
| Solution | https://youtu.be/UuiTKBwPgAo |
| Status | Done |

## Question

> You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `ith` 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

**Example 1:**




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

**Example 2:**

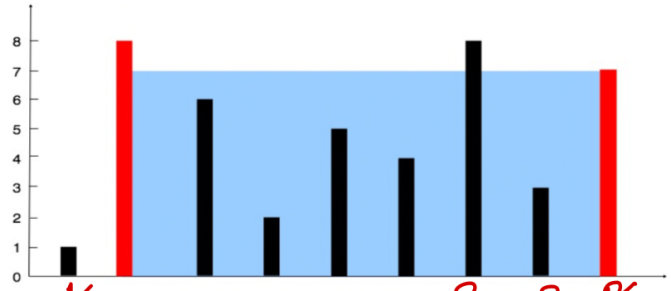
```
Input: height = [1,1]
Output: 1
```

# Idea

Shrinking window, left/right initially at endpoints, shift the pointer with min height;

# ⑧ Container with Most water.

**Example 1:**



**Input:** height = [1,8,6,2,5,4,8,3,7]
**Output:** 49
**Explanation:** 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 2:**

**Input:** height = [1,1]
**Output:** 1

Max Area

8
49

area = (r-l) * min(h[l], h[r])

= (8-0) * 1 = 8

## Solution

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        res = 0  # Initialize the result (maximum area) to 0
        l, r = 0, len(height) - 1  # Initialize two pointers, 'l' and 'r', to the two ends of the array

        while l < r:  # Loop until 'l' is less than 'r'
            # Calculate the area between the two lines (height[l] and height[r])
            area = (r - l) * min(height[l], height[r])

            # Update the result by taking the maximum of the current result and the calculated area
            res = max(res, area)

            if height[l] < height[r]:
                l += 1  # Move the left pointer to the right if the left line is shorter
            else:
                r -= 1  # Move the right pointer to the left if the right line is shorter

        return res  # Return the maximum area found
```

## Explanation

> **Time Complexity:**
> The time complexity of this code is O(n), where 'n' is the length of the `height` list. This is because we use a two-pointer approach that moves towards each other, and each step of the loop either increments the left pointer or decrements the right pointer.
>
> **Space Complexity:**
> The space complexity is O(1) because we only use a fixed amount of extra space to store the variables `res`, `l`, `r`, and `area`, regardless of the size of the input list.