Container With Most Water - Problem Explanation

■ Problem Understanding

You're given an array height where each element represents the height of a vertical line on the x-axis. You need to find two lines that, together with the x-axis, form a container that can store the maximum amount of water.

Formula:
`Area = (j - i) × min(height[i], height[j])`

■ Example

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

Explanation:
For i=1 (height=8) and j=8 (height=7), width = 7, height = 7
Area = 7 × 7 = 49 (maximum possible area)
```

■ Approaches

■ Brute Force (O(n²))

Try all possible pairs and calculate the area for each.

C++:
```
int maxArea(vector<int>& height) {
    int n = height.size();
    int maxArea = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int area = (j - i) * min(height[i], height[j]);
            maxArea = max(maxArea, area);
        }
    }
    return maxArea;
}
```
Java:
```
public int maxArea(int[] height) {
    int n = height.length;
    int maxArea = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int area = (j - i) * Math.min(height[i], height[j]);
            maxArea = Math.max(maxArea, area);
        }
    }
    return maxArea;
}
```
Python:
```
def maxArea(height):
    n = len(height)
```

```
    max_area = 0
    for i in range(n):
        for j in range(i + 1, n):
            area = (j - i) * min(height[i], height[j])
            max_area = max(max_area, area)
    return max_area
```

■ Optimized — Two Pointer Approach (O(n))

Use two pointers: one at the start, one at the end. Move the smaller height pointer inward, since moving the taller one cannot increase area.

C++:

```
int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        int width = right - left;
        int h = min(height[left], height[right]);
        maxArea = max(maxArea, width * h);
        if (height[left] < height[right])
            left++;
        else
            right--;
    }
    return maxArea;
}
```

Java:

```
public int maxArea(int[] height) {
    int left = 0, right = height.length - 1;
    int maxArea = 0;
    while (left < right) {
        int width = right - left;
        int h = Math.min(height[left], height[right]);
        maxArea = Math.max(maxArea, width * h);
        if (height[left] < height[right])
            left++;
        else
            right--;
    }
    return maxArea;
}
```

Python:

```
def maxArea(height):
    left, right = 0, len(height) - 1
    max_area = 0
    while left < right:
        width = right - left
        h = min(height[left], height[right])
        max_area = max(max_area, width * h)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1
    return max_area
```

■ Summary

Approach	Time	Space	Description
Brute Force	$O(n^2)$	O(1)	Check all pairs
Two Pointer	O(n)	O(1)	Optimal approach