1. Problem

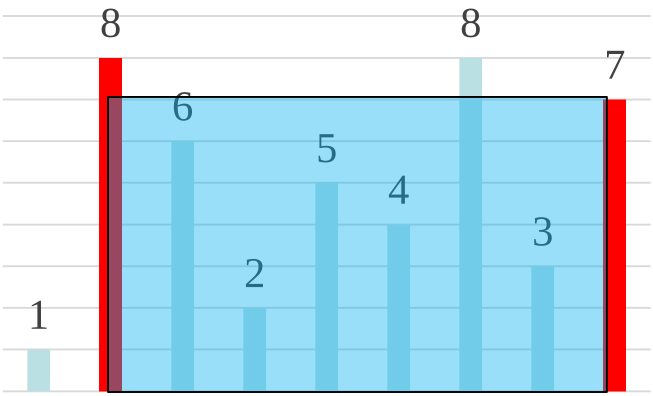
- Container with Most Water

Input: A non-negative integers array.

Output: An integer which corresponds to the maximum area of water that can be contained.

Example: Input: A = [1, 8, 6, 2, 5, 4, 8, 3, 7]

Output: Area of blue section: 49



	Time complexity	Space complexity
Naive brute force	$O(n^2)$	$O(1)$
Two pointer	$O(n)$	$O(1)$

1) Naive approach (Brute Force)

Pseudocode:

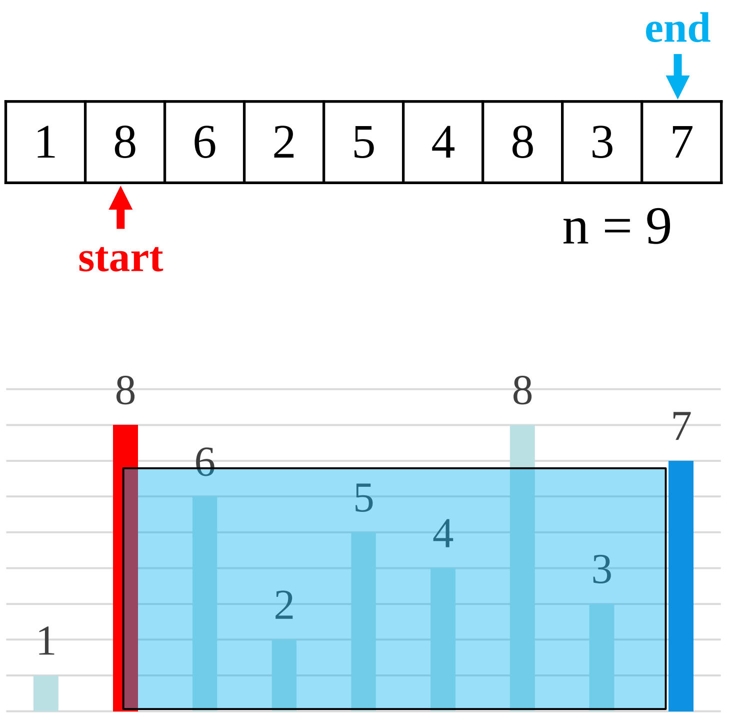
maxArea (A[0..(n - 1)])

Input: A non-negative integers array: A

Output: The max area that can be contained.

```
area_max ← 0
for start ← 0 to n - 2 do
    for end ← start + 1 to n - 1 do
        height_curr ← min(A[start], A[end])
        area_curr ← height_curr * (end – start)
        area_max ← max(area_max, area_curr)
return area_max
```

Visualization:



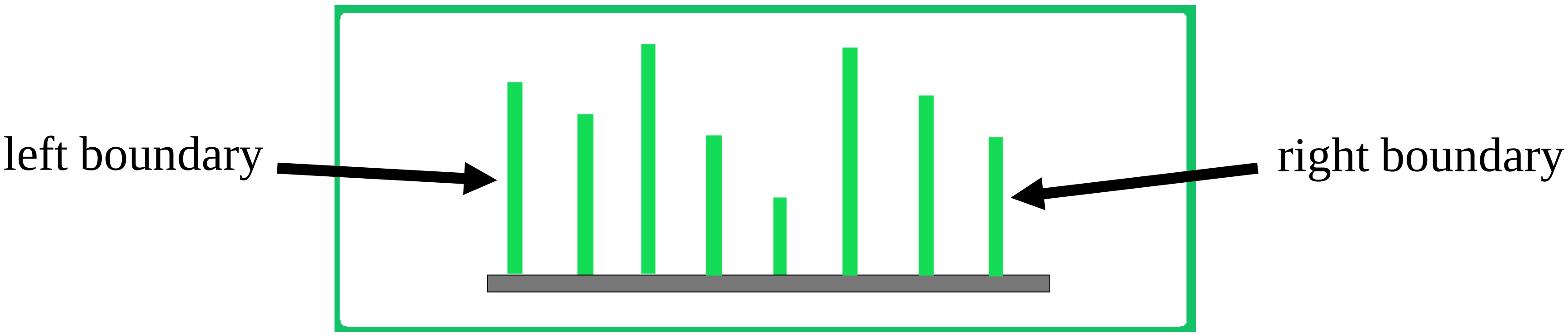
Area_curr = 7 * (8 - 1) = 49

Time complexity: $O(n^2)$

Space complexity: $O(1)$

2) Two pointers

Intuition:



$$\text{area} = \underbrace{\min(\text{height\_left}, \text{height\_right})}_{\text{limit 1: height}} * \underbrace{(\text{index\_right} - \text{index\_left})}_{\text{limit 2: width}}$$

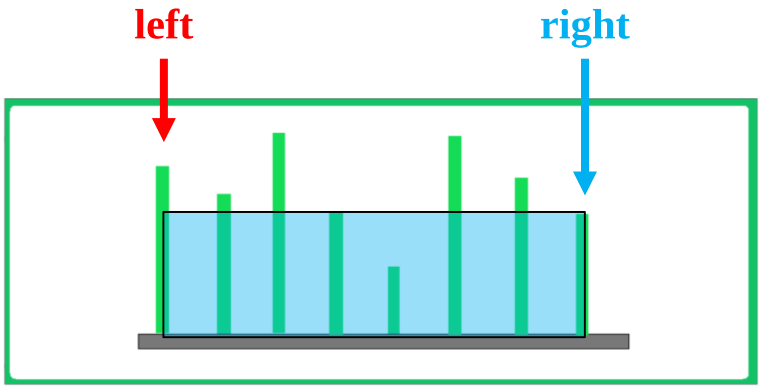
Get started with **maximum width**: take 2 pointers at the **left and right boundary**

https://www.geeksforgeeks.org/container-with-most-water/

2) Two pointers

Pseudocode:

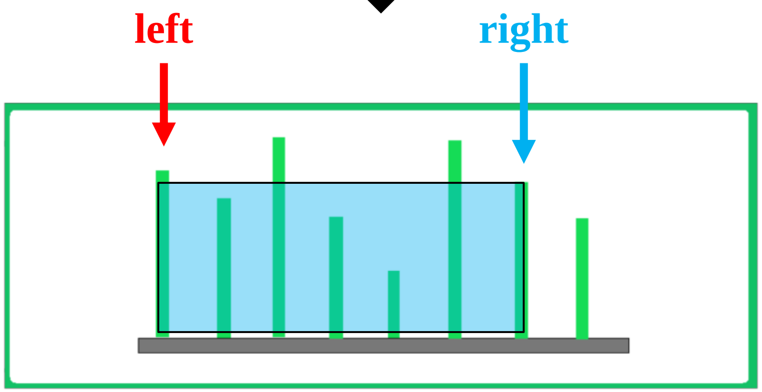
```
maxArea (A[0..(n - 1)])
Input: A non-negative integers array: A
Output: The max area that can be contained.
area_max ← 0; left ← 0; right ← n - 1
while left < right do
    height_curr ← min(A[left], A[right])
    area_curr ← height_curr * (right – left)
    area_max ← max(area_max, area_curr)
    if A[left] <= A[right] do
        left = left + 1
    else
        right = right - 1
return area_max
```



initially

A[left] > A[right]

right index 1 step towards left



1st move

Time complexity: O(n) Space complexity: O(1)

	Time complexity	Space complexity
Naive brute force	O(n^2)	O(1)
Two pointer	O(n)	O(1)

- Trapping Rainwater

Input: A non-negative integers array.

Output: The maximum area the given structure can trap.

Example: Input:

3	0	2	0	4

Output: Area of blue section: 7