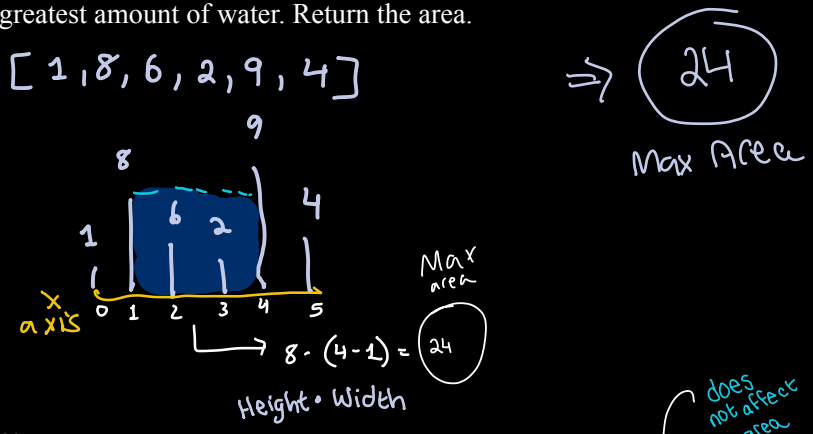
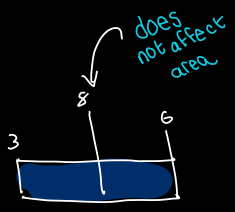
Problem: Given array of + ints (rep heights). Find 2 lines (along with x-axis) which form container that can hold the greatest amount of water. Return the area.





Intro:

- Verify Constraints
  - Line thickness affect area? No
  - Left + right sides count as walls? No
  - Higher line inside container affect area? No
- Create Testcases
  - [7, 1, 2, 3, 9] (7 and 9 are the furthest apart and the longest heights) Area: 7*(4-0) = 28
  - [], Area = 0
  - [7], Area = 0
  - [6, 9, 3, 4, 5, 8] -> 6,8 -> Area: 6*5 = 30 :: 9, 8 -> Area: 8*4 = 32

Brute Force:

- Brainstorming & Pattern Observations

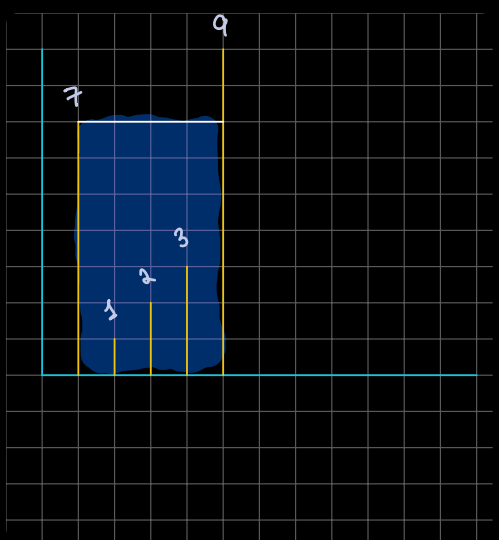
Focus: find the GREATEST area

[7, 1, 2, 3, 9]

area = length * width = l*w

(a, b) - 2 line lengths

min(a, b) * distance(a, b)



2 Pointers + Double For-loop

```
a = 0
b = 1

maxArea =  0

min(7, 1) * (1-0) = 1
0 < 1
maxArea = 1

min(7, 2)*(2-0) = 4
1 < 4
maxArea = 4

min(7, 3)*(3-0) = 9
4 < 9
maxArea = 9

min(7, 9)*(4-0) = 28
9 < 28
maxArea = 28

restart with new indices
a = 1
b = 2

min(1, 2)*(2-1) = 1
28 < 1 -> false

min(1, 3)*(3-1) = 2
28 < 2 -> false

min(1, 9)*(4-1) = 27
28 < 27 -> false

restart with new indices
a = 2
b = 3

min(2, 3)*(3-2) = 2
28 < 2 -> false

min(2, 4)*(4-2) = 4
28 < 4 -> false

…

restart
a = 3
b = 4

…

end
```

- Pseudocode Brute force solution

  Brute force: calculate the max area for every 2 lines

```
maxArea = 0
2 pointer method:
p1 @ first line
p2 @ second line
double for loop and keep moving p2 and
update maxArea if the maxArea is < than
the new area calculated from the 2 lines
```

- Write
- Run through testcases
- Analyze time and space complexity
  Time: O(n^2): 2 pointers & double iterative loop
  Space: O(1)

Optimal:

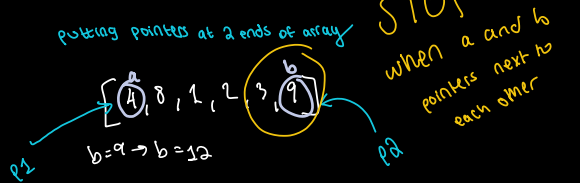
- Brainstorming & Pattern Observations

  No additional caching can happen in first loop or second loop -> no hashmap
  New technique: shifting pointers

area = min(a, b)*distance(a, b)

focus on maximizing — already maxed out — now we focus on maximizing min(a,b)

so we focus on the smaller value and try to increase it => i.e move the smallest-value pointer forward or backward (a) (b)



if larger # gets larger → does not impact min(a,b)

if larger # gets smaller → impacts min(a,b) => SMALLER AREA ✗

- Pseudocode

```
maxArea = 0
a = 0
b = len(arr)-1
while a < b:
    calculate the new area:
min(arr[a],arr[b])*(b-a)
    if new area > maxArea:
        maxArea = new area
    move the pointer at the smallest value
    to the next element
```

- Write code
- Run through testcases
- Analyze time and space complexity

Time: O(n): the pointers are not repeatedly covering the same elements, bc start at max widths and only move according to minimum height inwards, the array is traversed only **one time** with **both pointers**

Space: O(1)