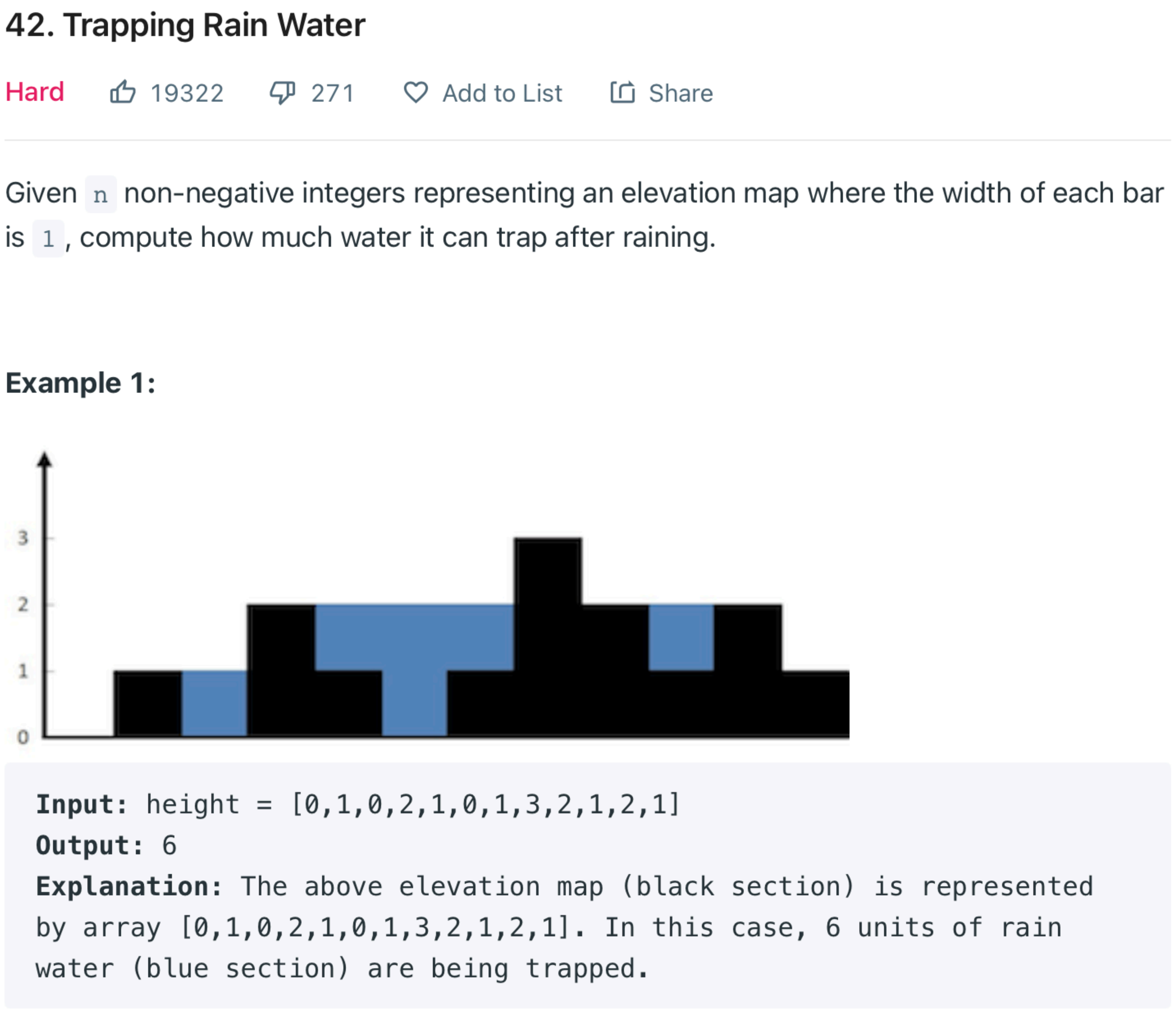
Can you try for Two Pointer Technique : Can be improved on Space : O(1)



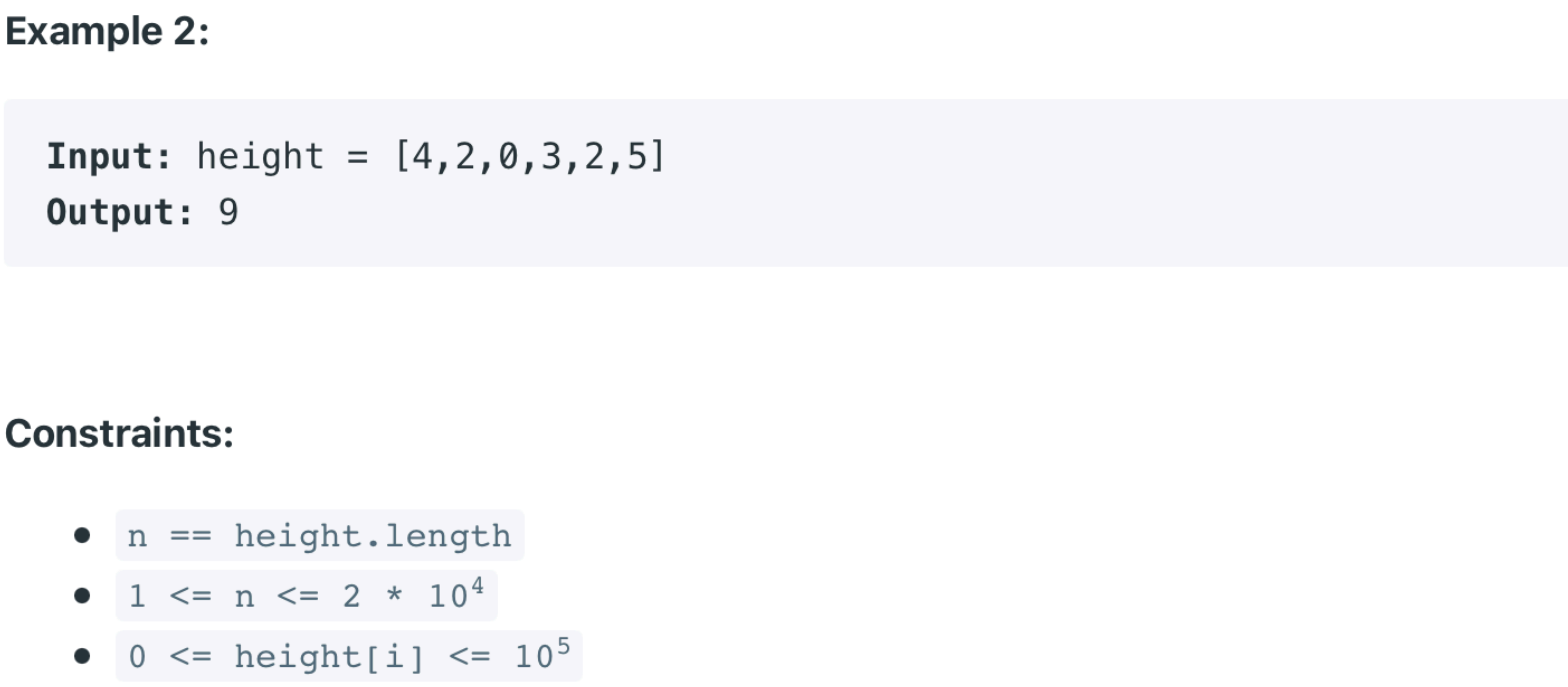
### 42. Trapping Rain Water

Hard 19322 271 Add to List Share

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

**Example 1:**

```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented
by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain
water (blue section) are being trapped.
```



**Example 2:**

```
Input: height = [4,2,0,3,2,5]
Output: 9
```

**Constraints:**

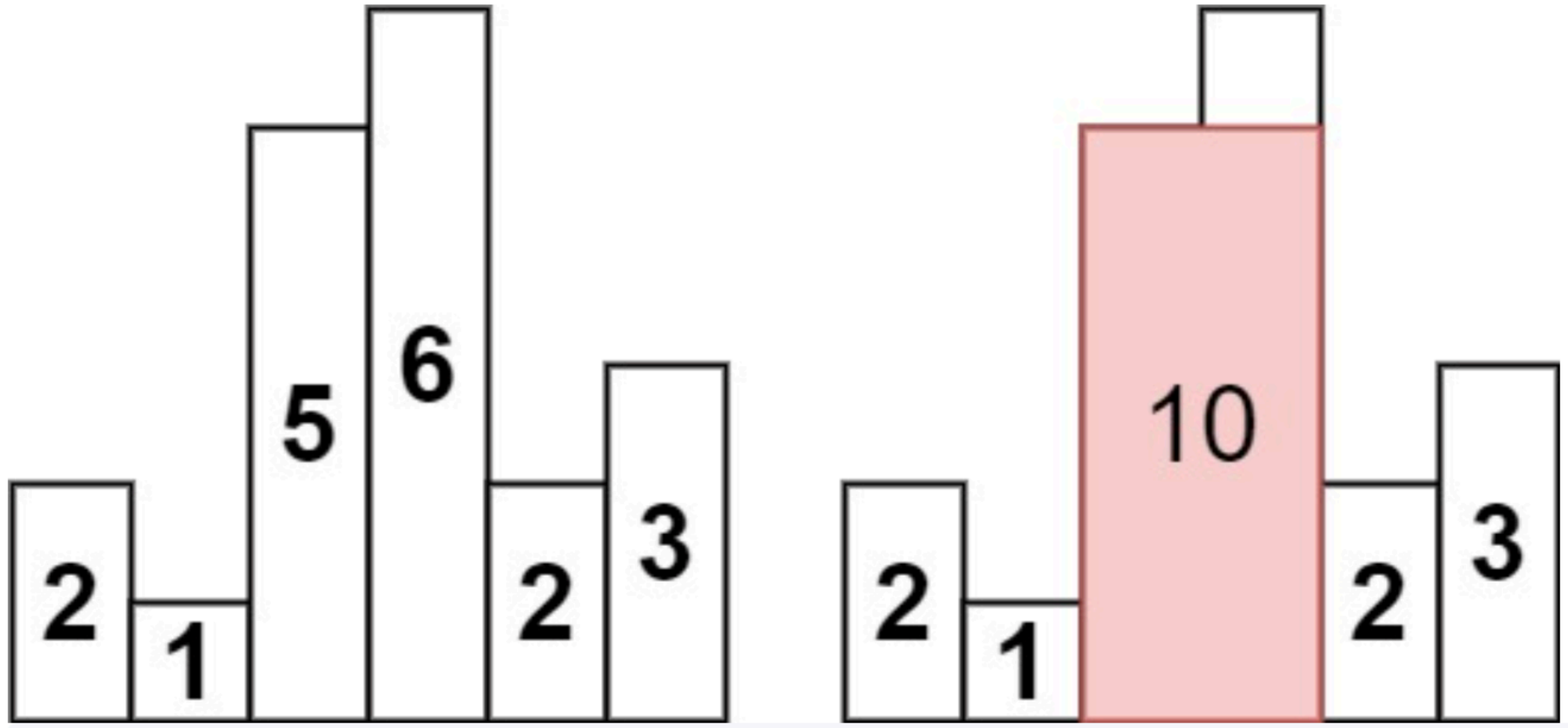
- `n == height.length`
- $1 <= n <= 2 * 10^4$
- $0 <= height[i] <= 10^5$

## 84. Largest Rectangle in Histogram

Hard 👍 10504 👎 151 ♡ Add to List Share

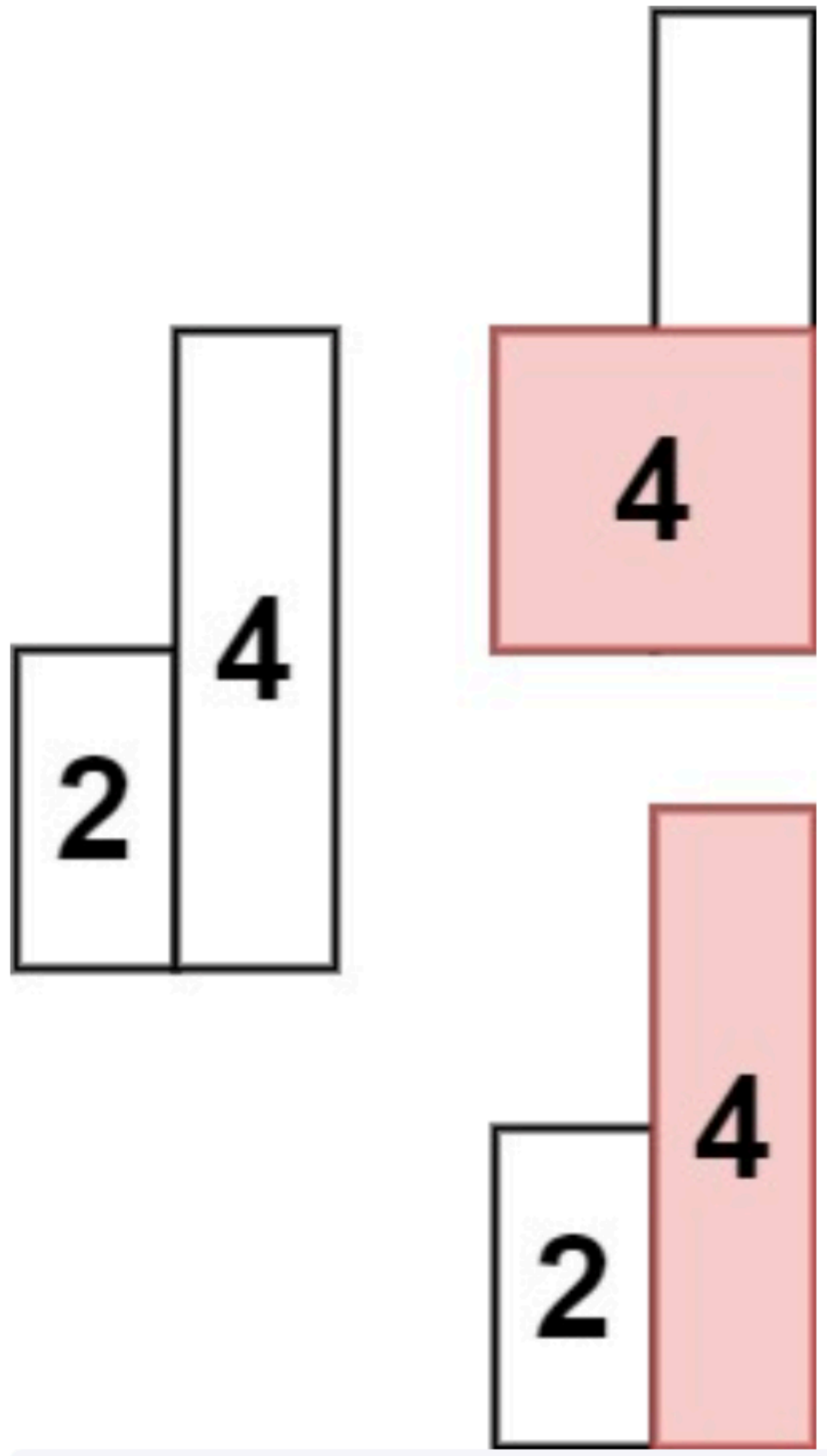
Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram*.

**Example 1:**



```
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area =
10 units.
```

**Example 2:**



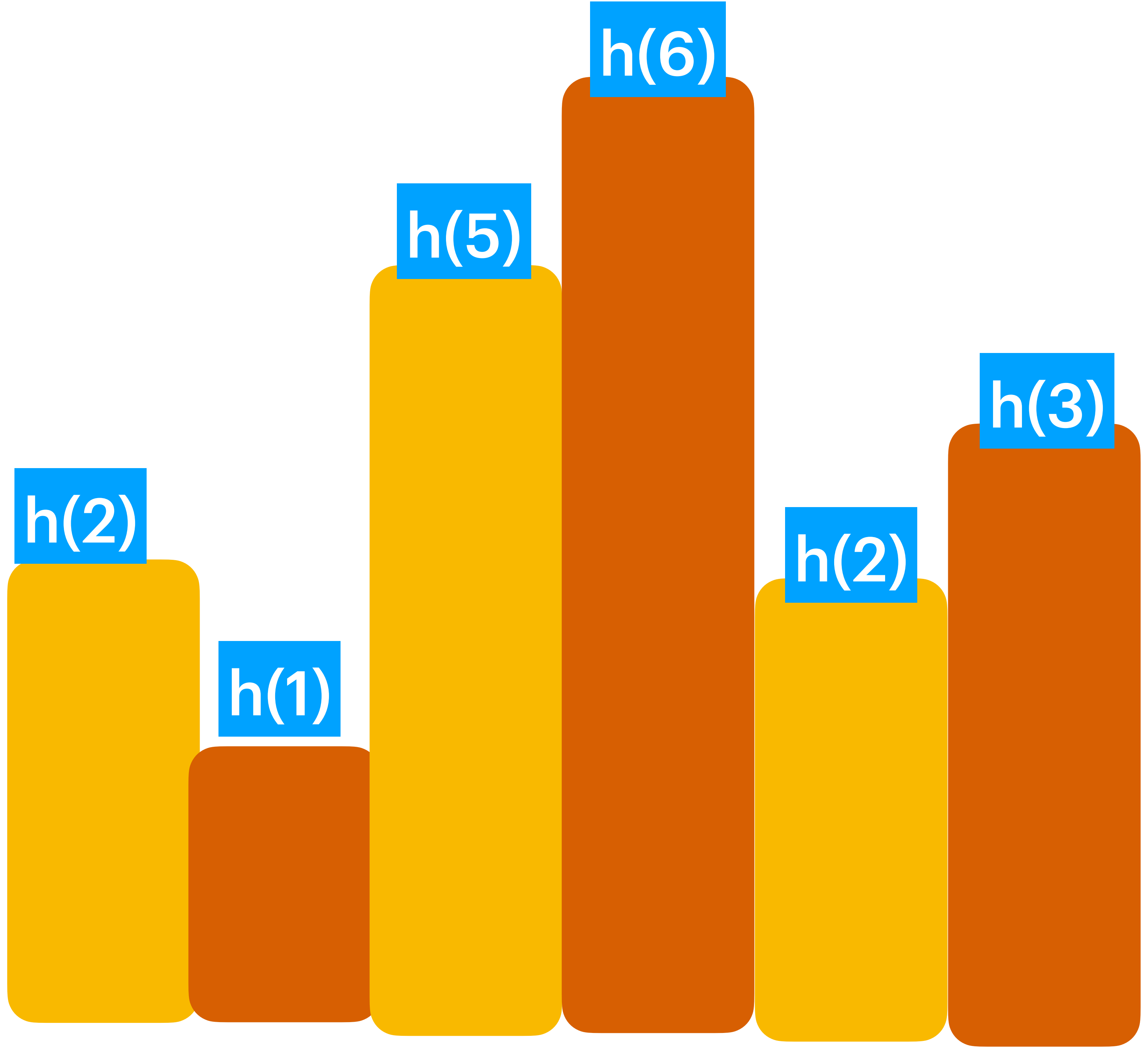
```
Input: heights = [2,4]
Output: 4
```

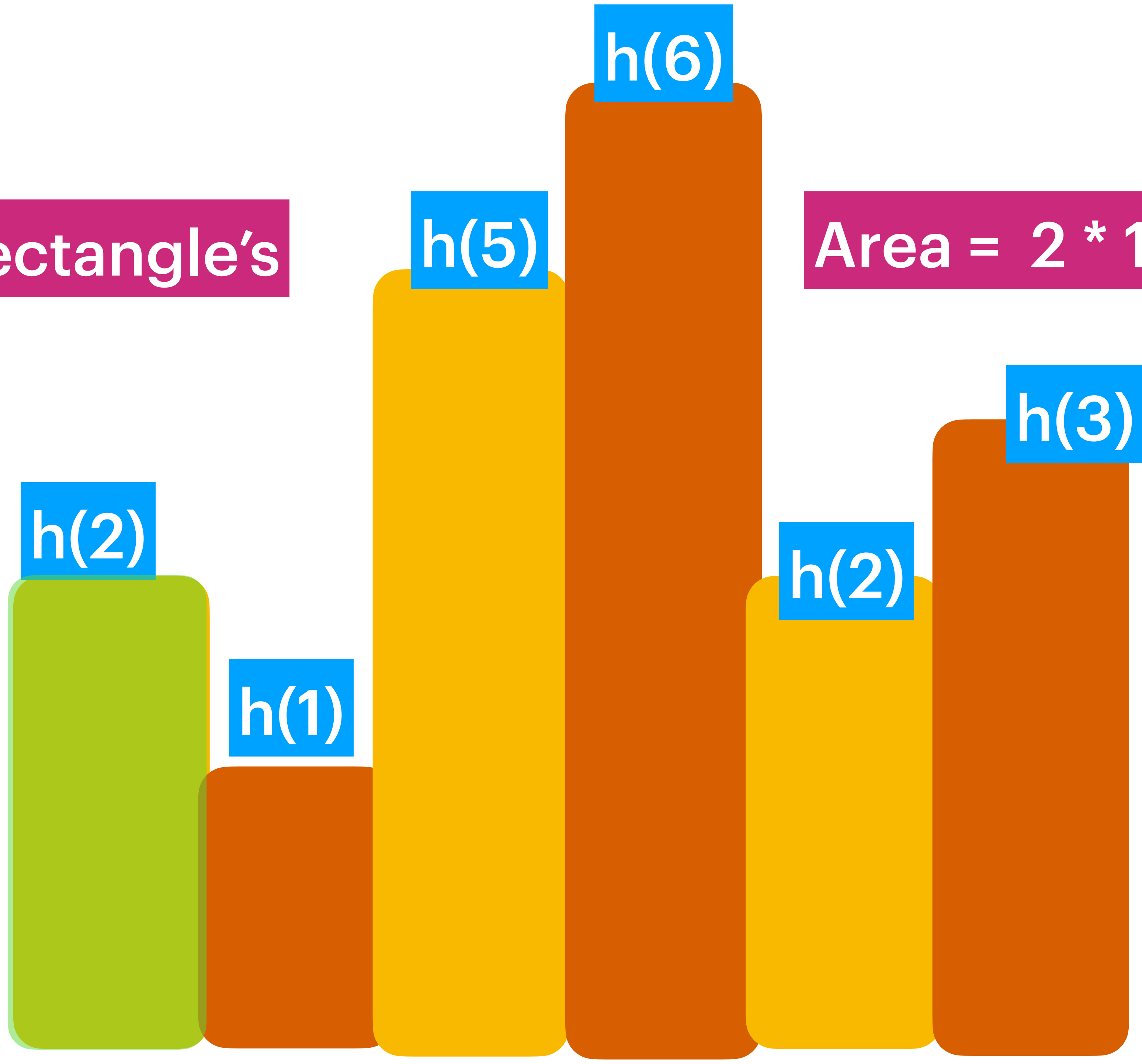
**Constraints:**

- `1 <= heights.length <= 10`$^5$
- `0 <= heights[i] <= 10`$^4$

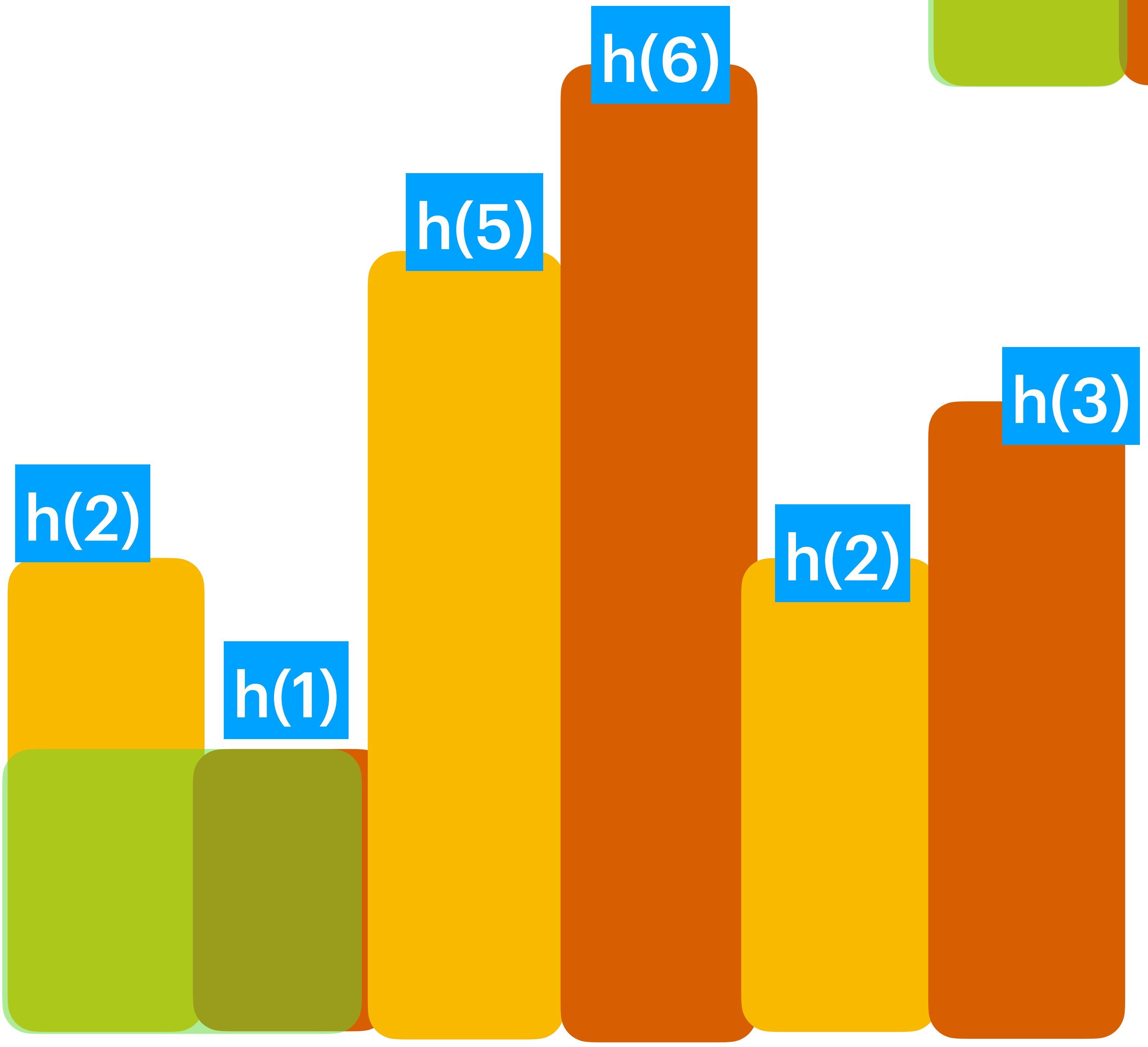
{2,1,5,6,2,3}

Lets Figure Out Possible Rectangle's





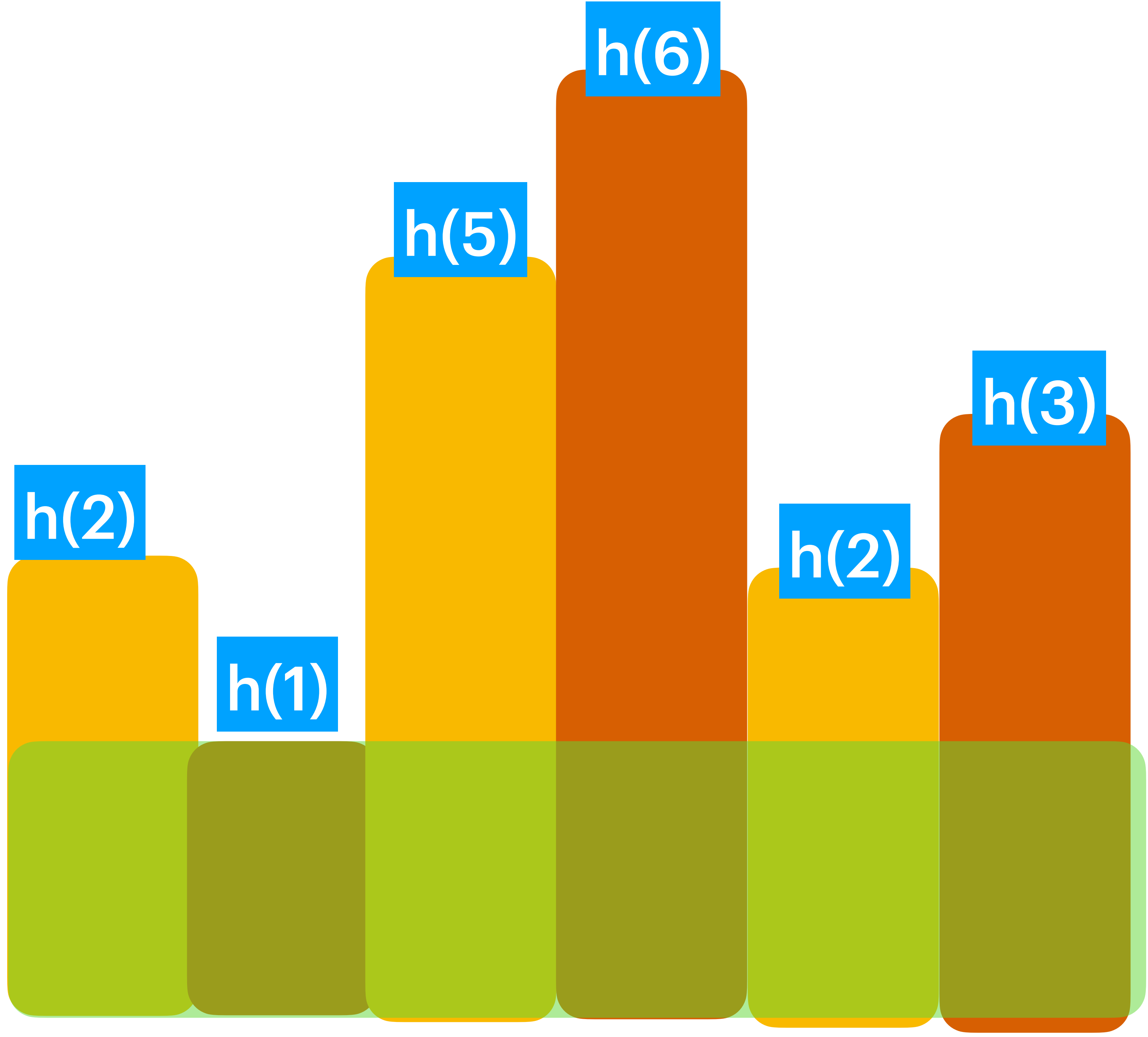
Area = 2 * 1 = 2



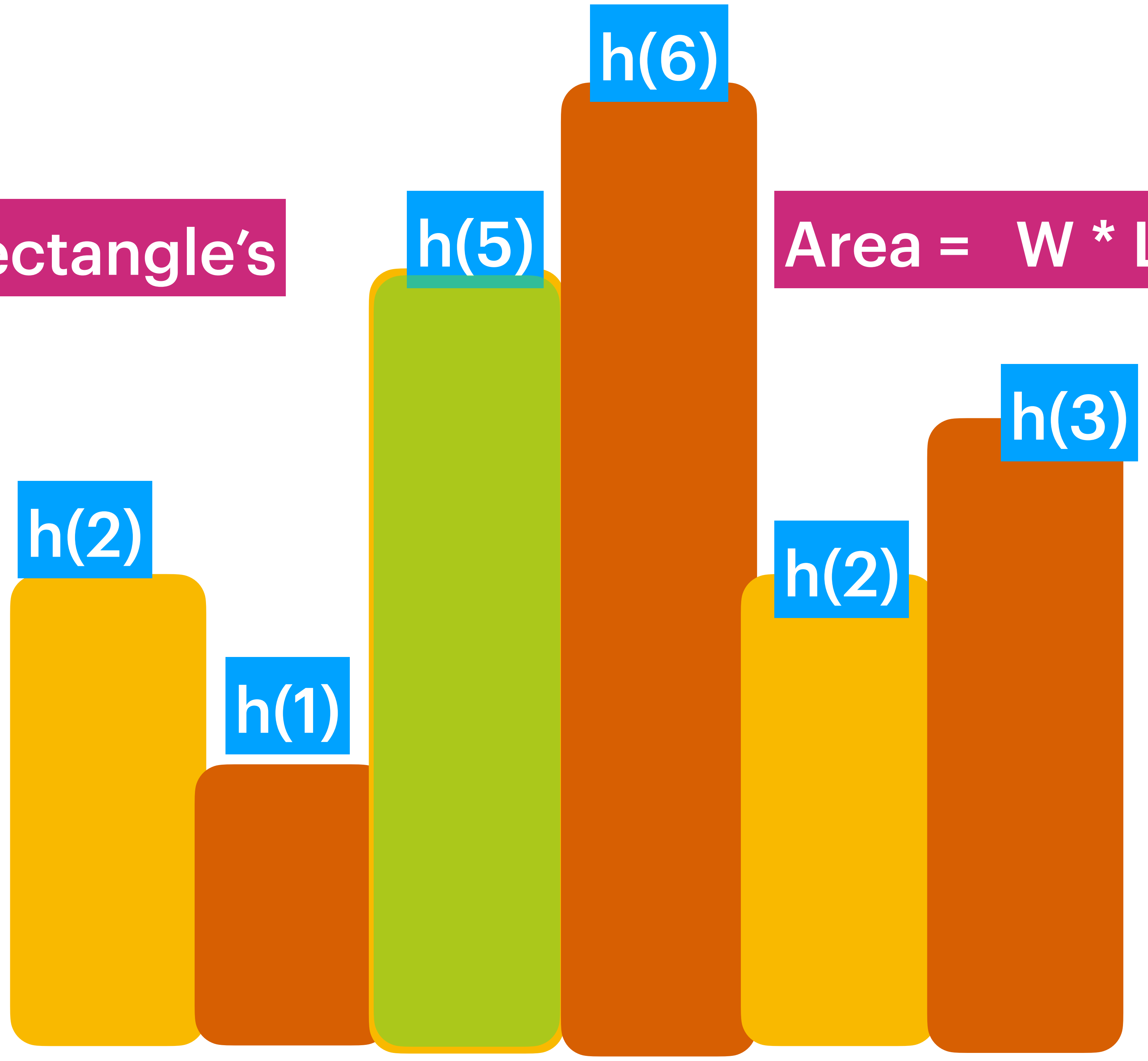
Area = 2 * 1 = 2

{2,1,5,6,2,3}

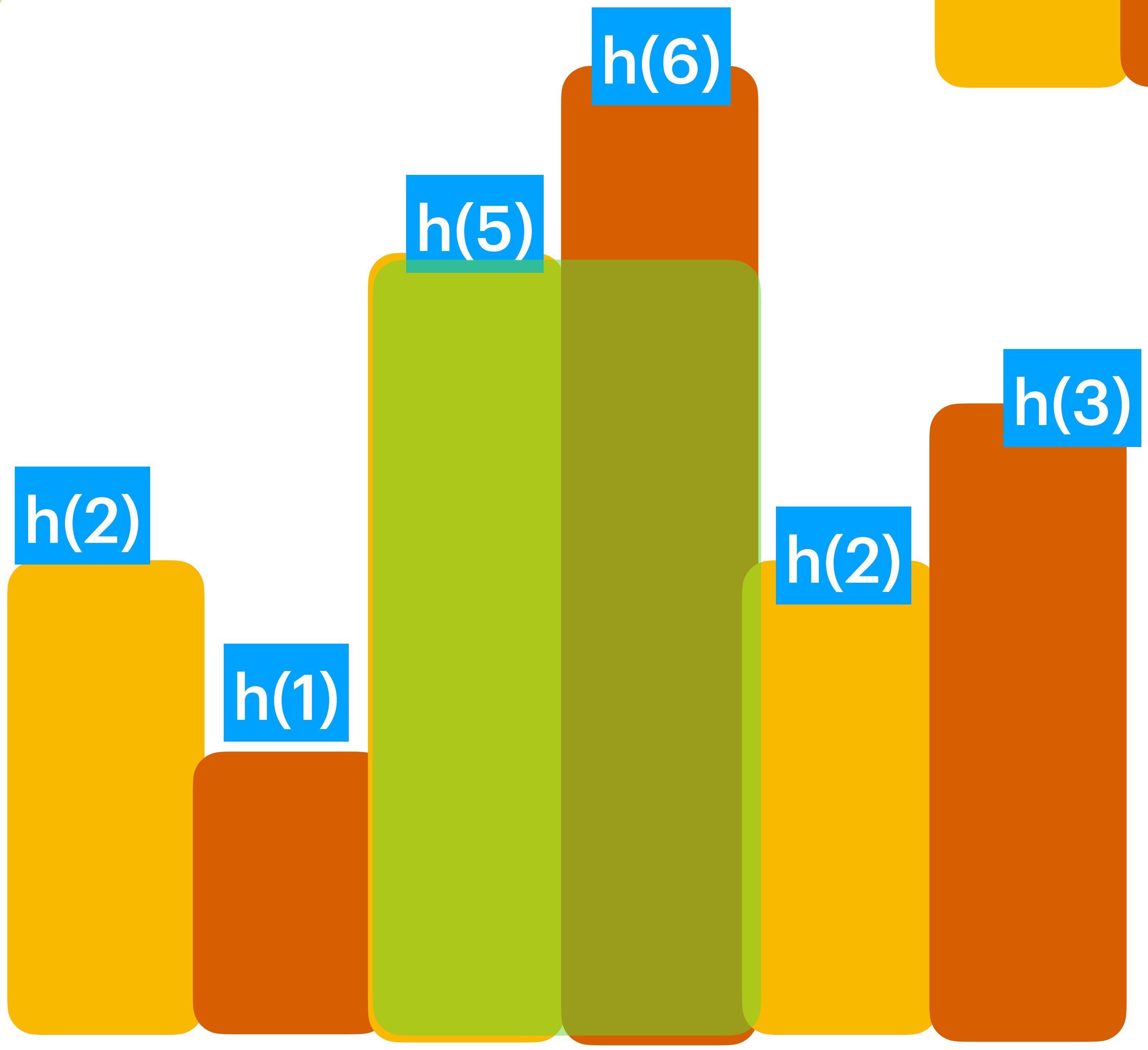
Lets Figure Out Possible Rectangle’s



Area = 6*1 = 6



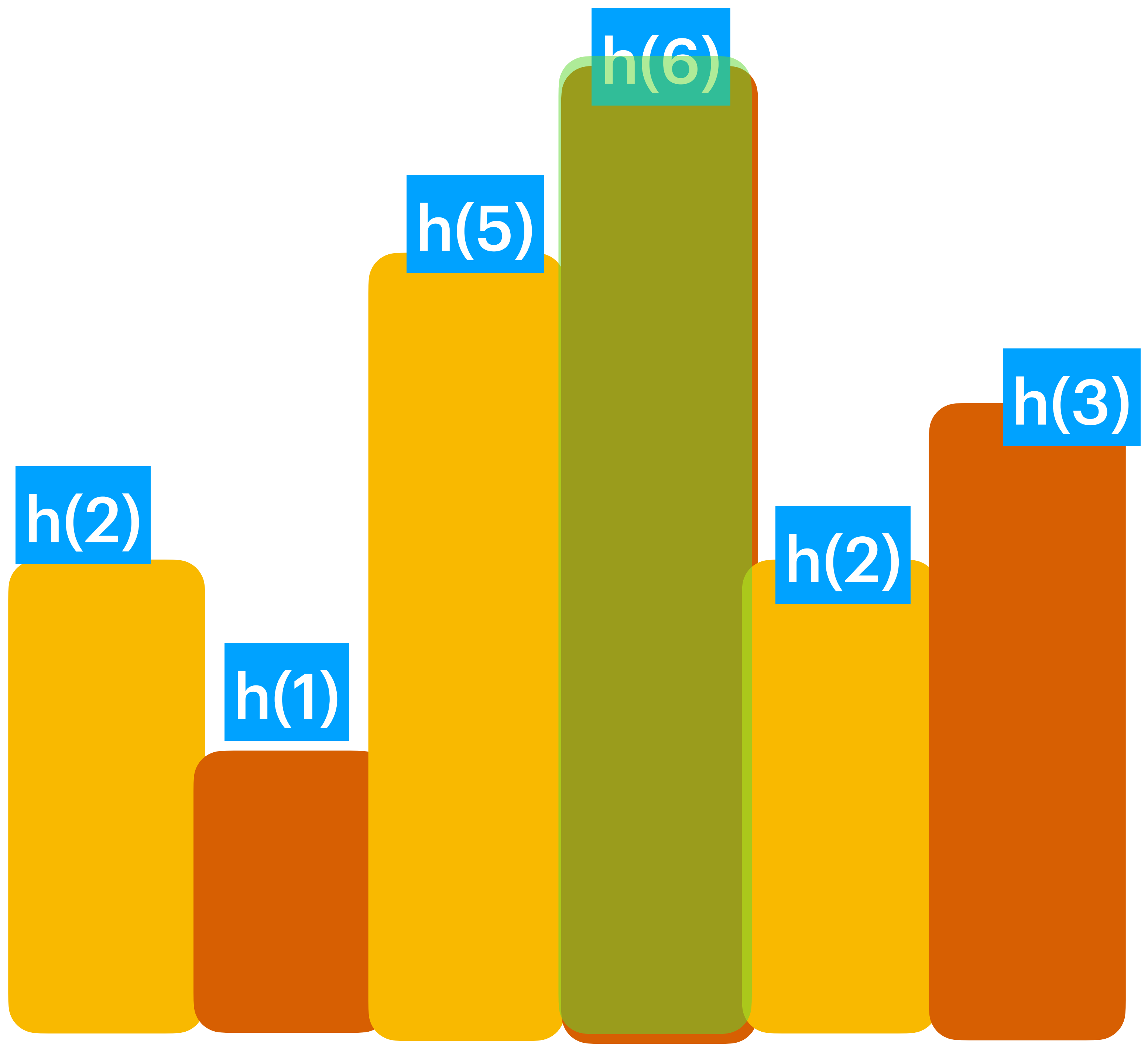
Area = W * L = 1 * 5 = 5



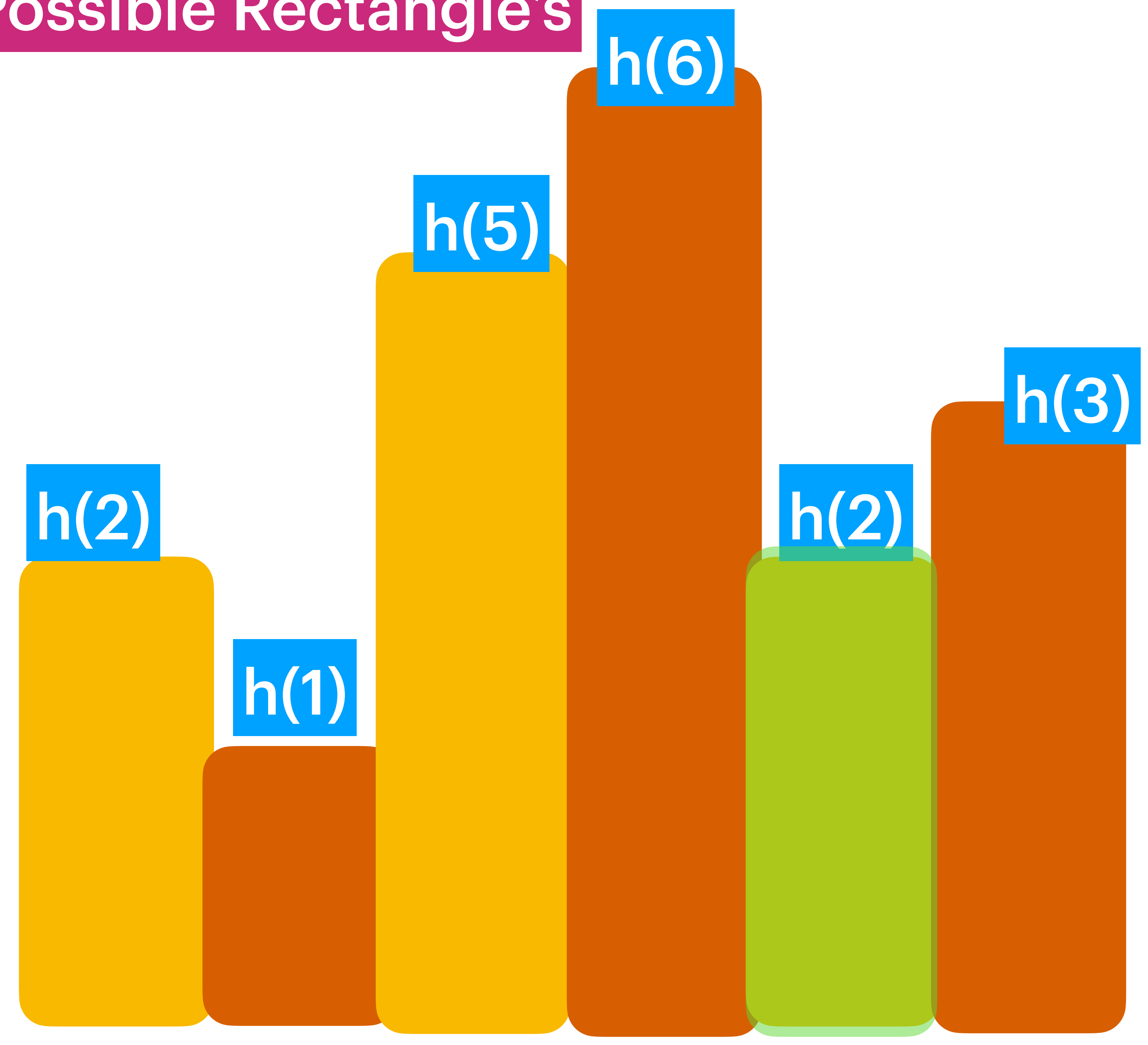
Area = W * L = 2 * 5 = 10

# Lets Figure Out Possible Rectangle’s

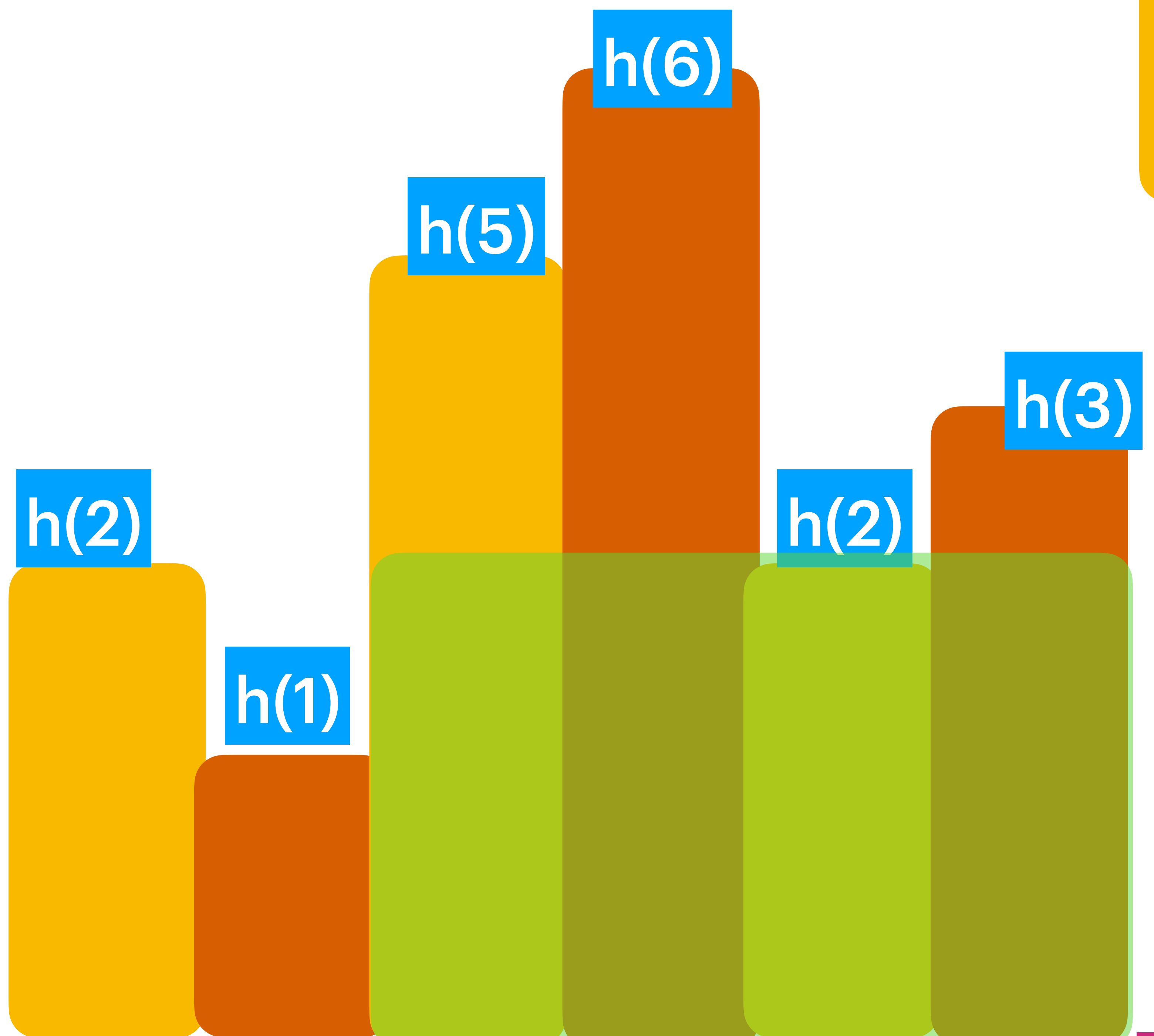
{2,1,5,6,2,3}



Area = W * L = 1 * 6 = 6



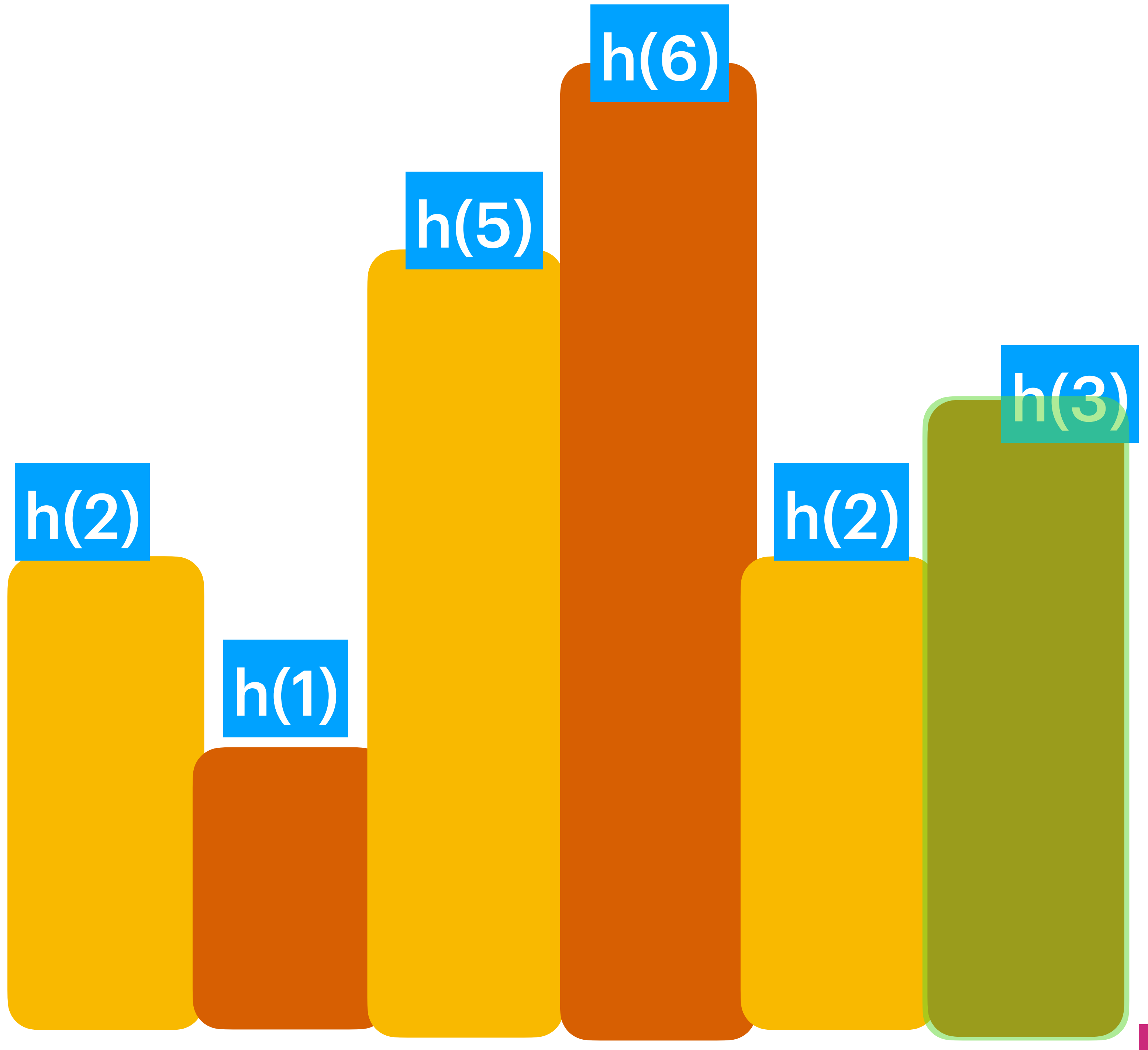
Area = W * L = 1 * 2 = 2



Area = W * L = 4 * 2 = 8

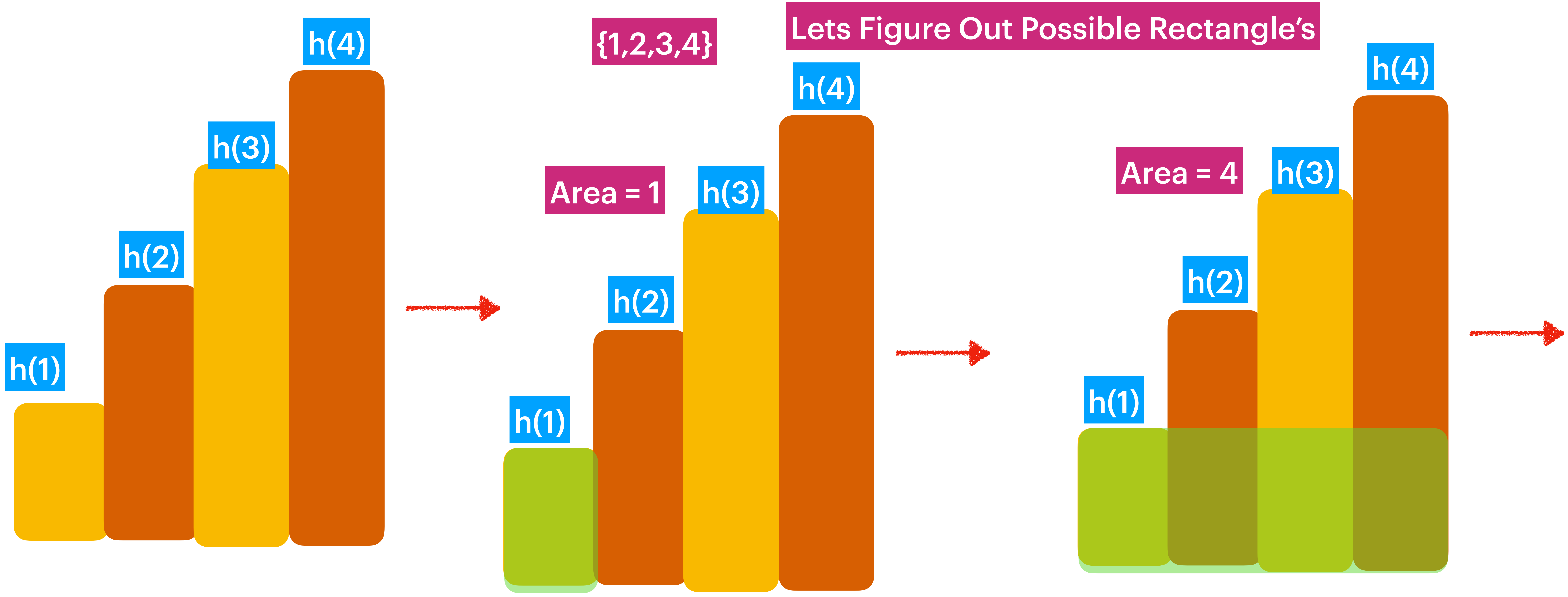
Lets Figure Out Possible Rectangle’s

{2,1,5,6,2,3}



Area = W * L = 1 * 3 = 3

Lets Figure Out Possible Rectangle’s



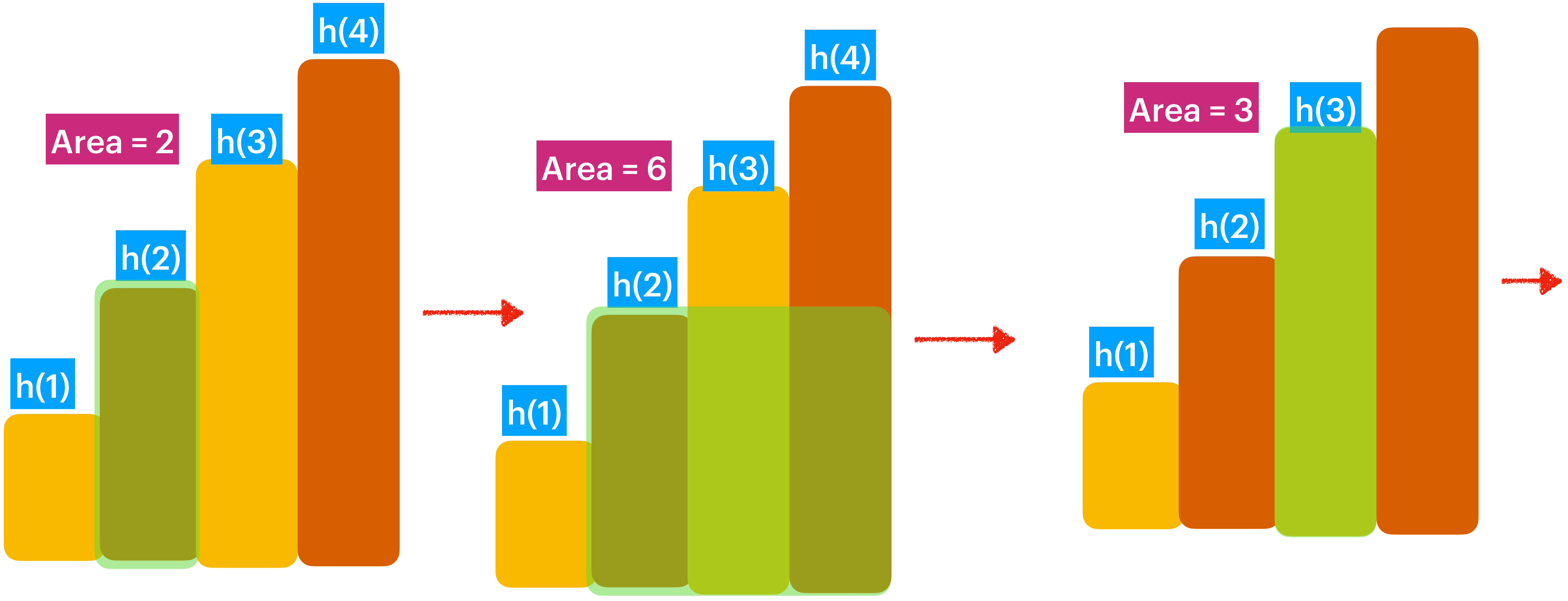
{1,2,3,4}

Area = 1

Area = 4

## Lets Figure Out Possible Rectangle’s

{1,2,3,4}



Area = 2

h(1)

h(2)

h(3)

h(4)

Area = 6

h(1)

h(2)

h(3)

h(4)

Area = 3

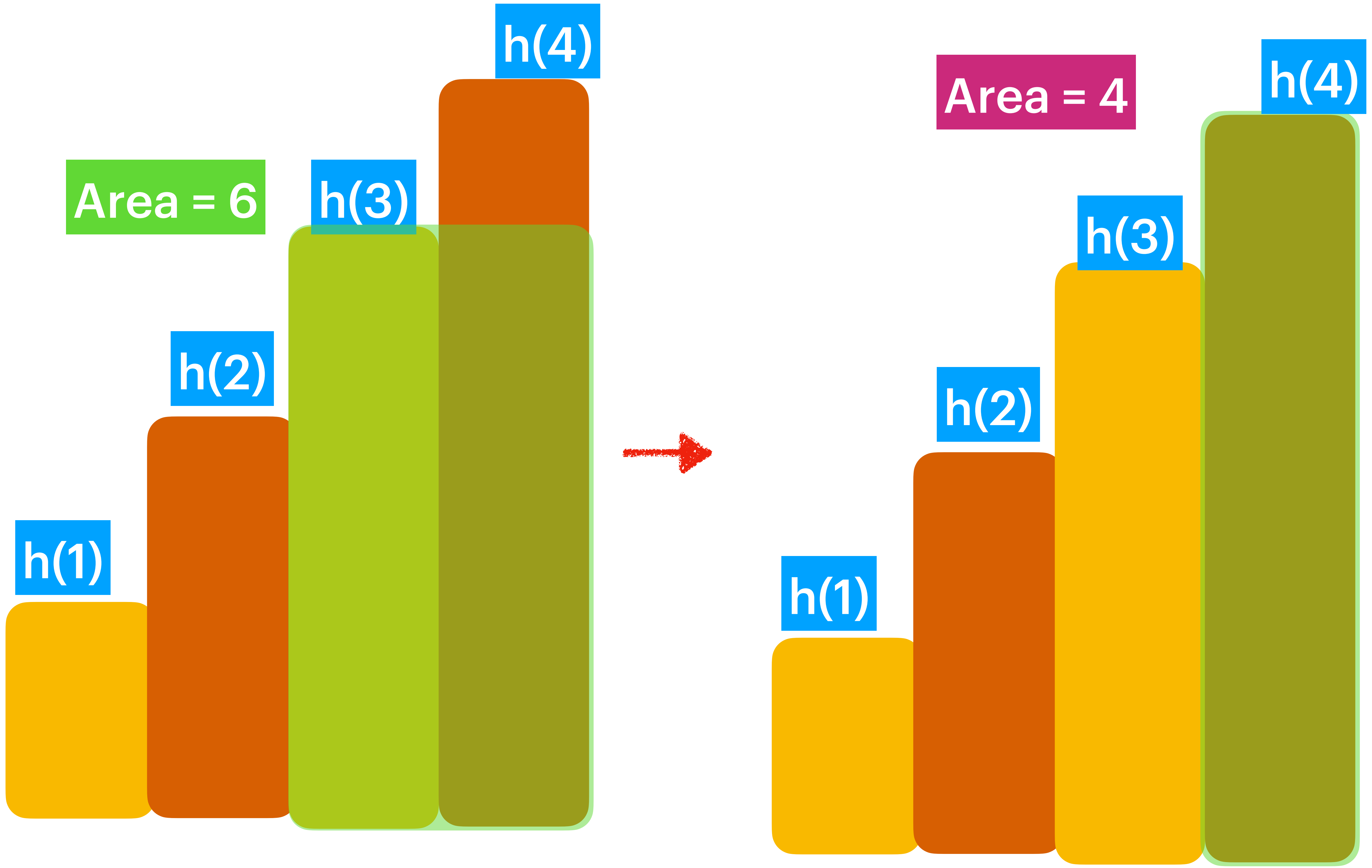
h(1)

h(2)

h(3)

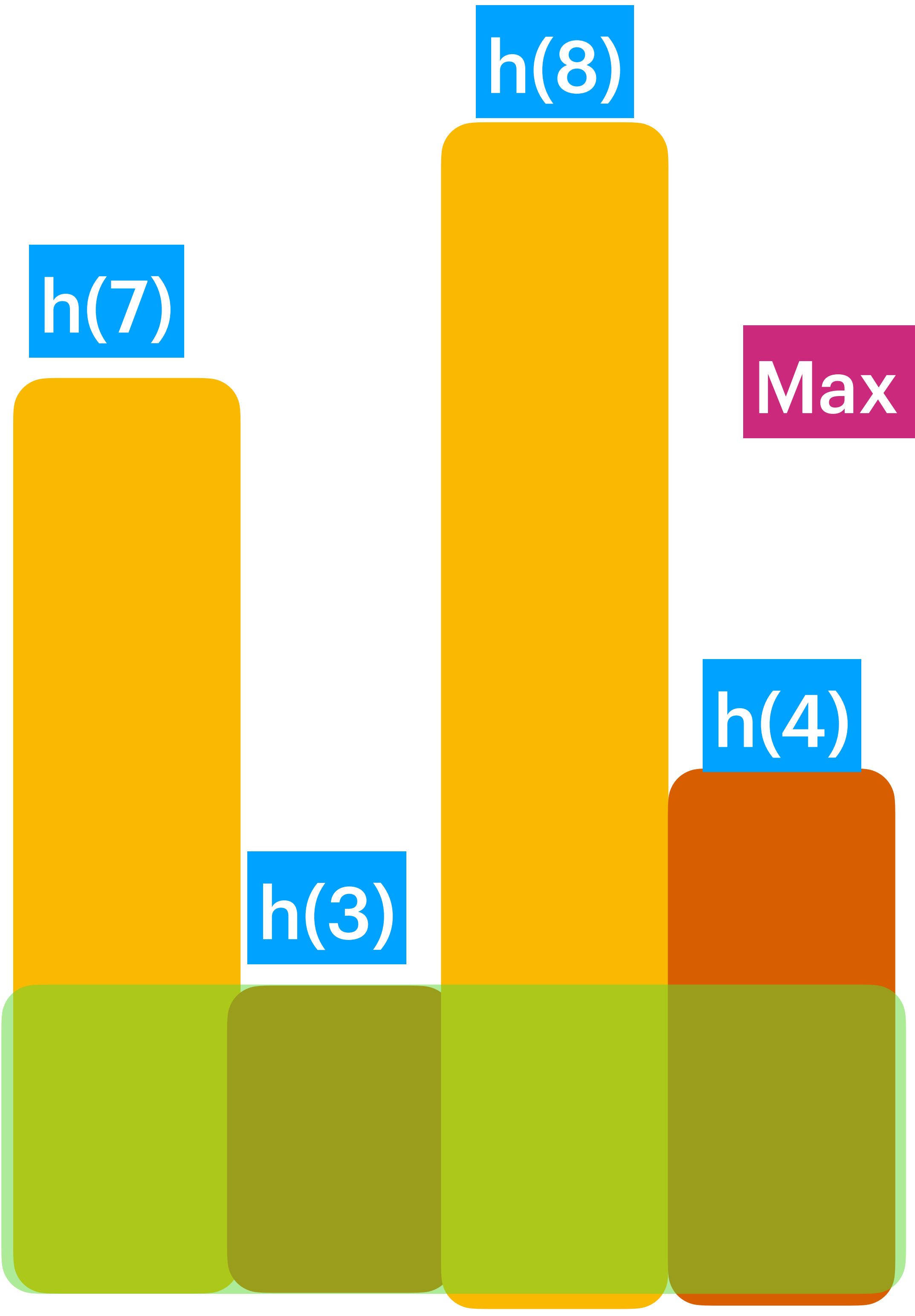
## Lets Figure Out Possible Rectangle’s

{1,2,3,4}

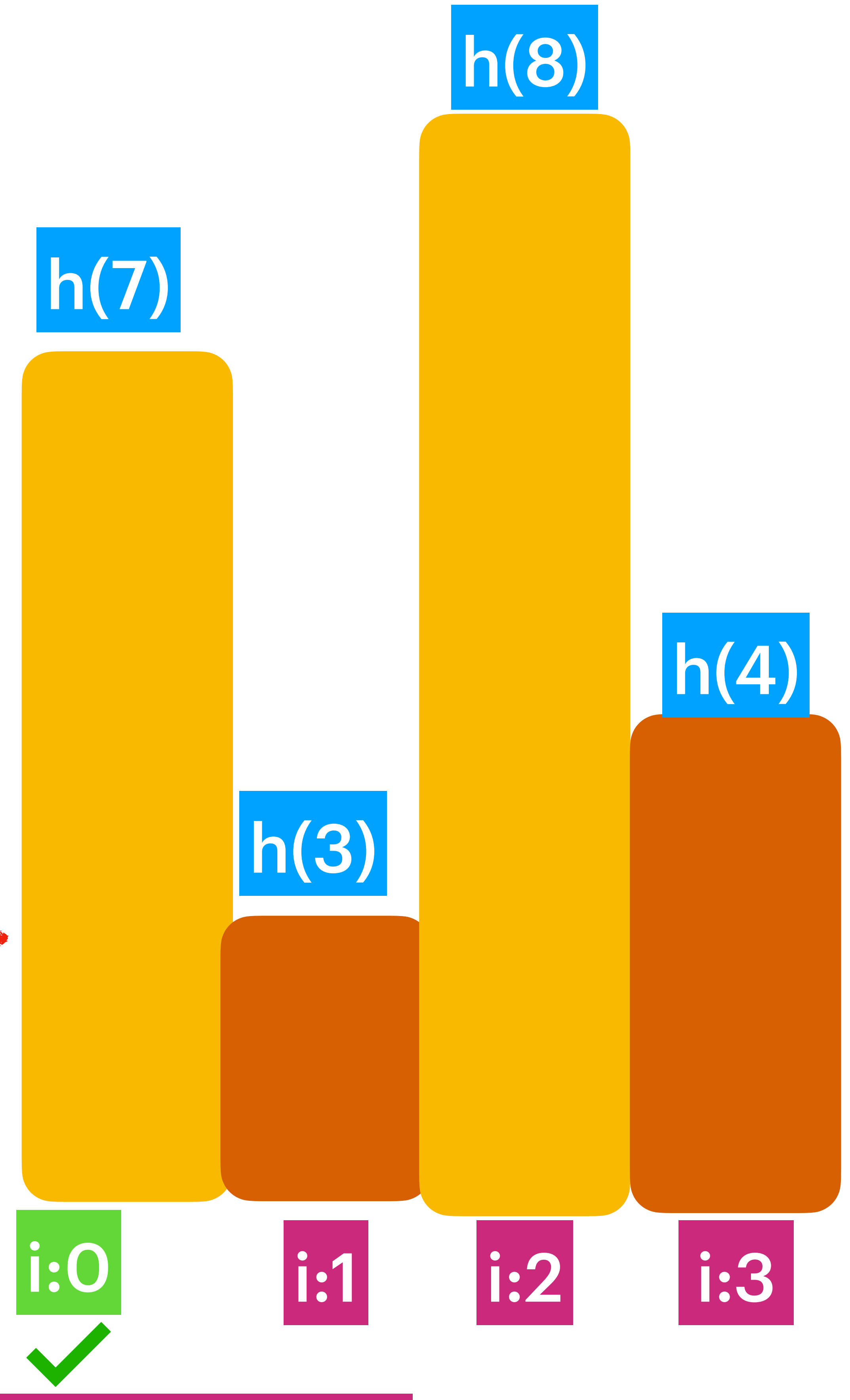


Lets Figure Out Possible Rectangle's

{7,3,8,4}

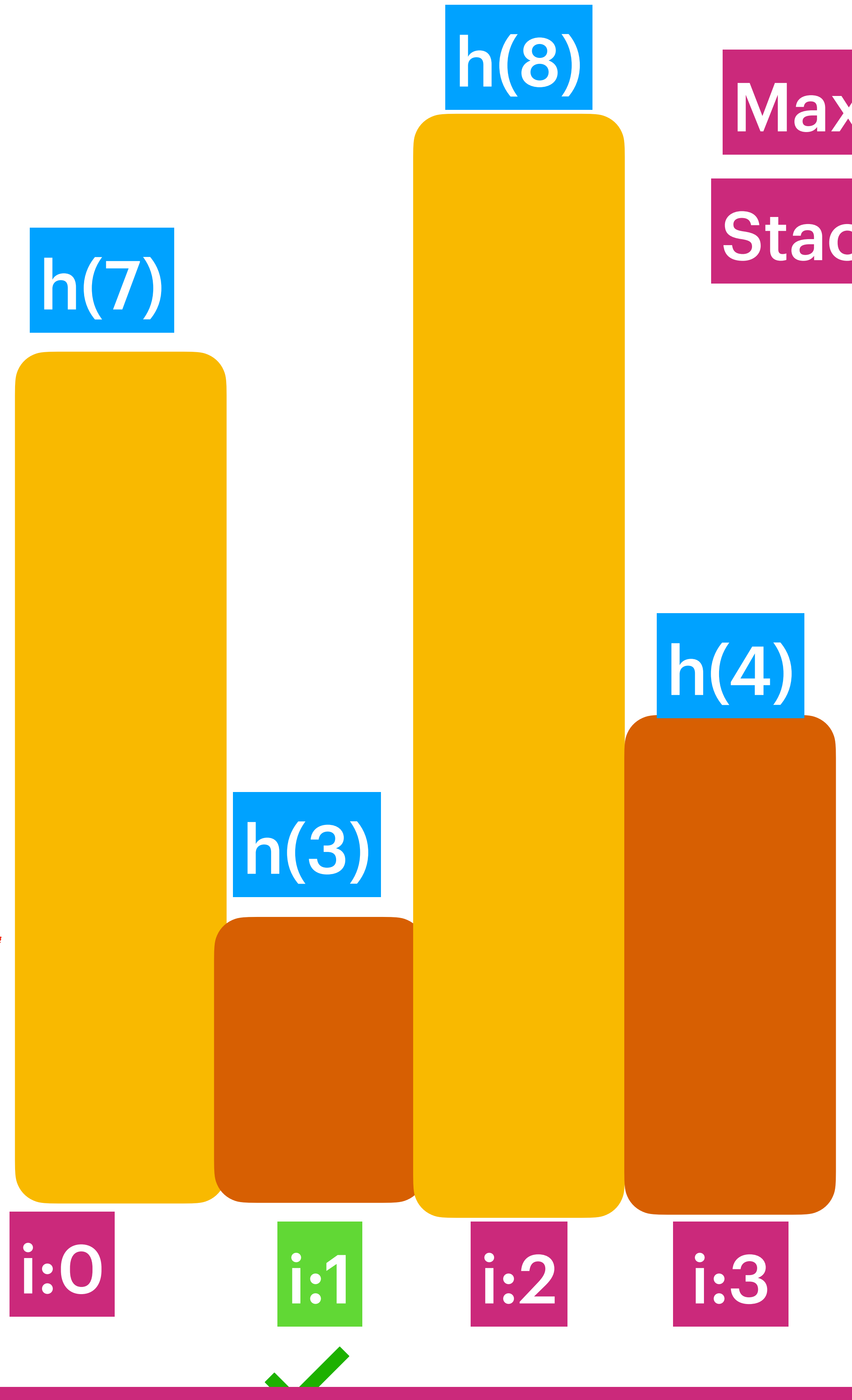


Max Area = 12



As the stack is empty push it.
Stack —> [i:0 — v:7 ]

Max Area = 0



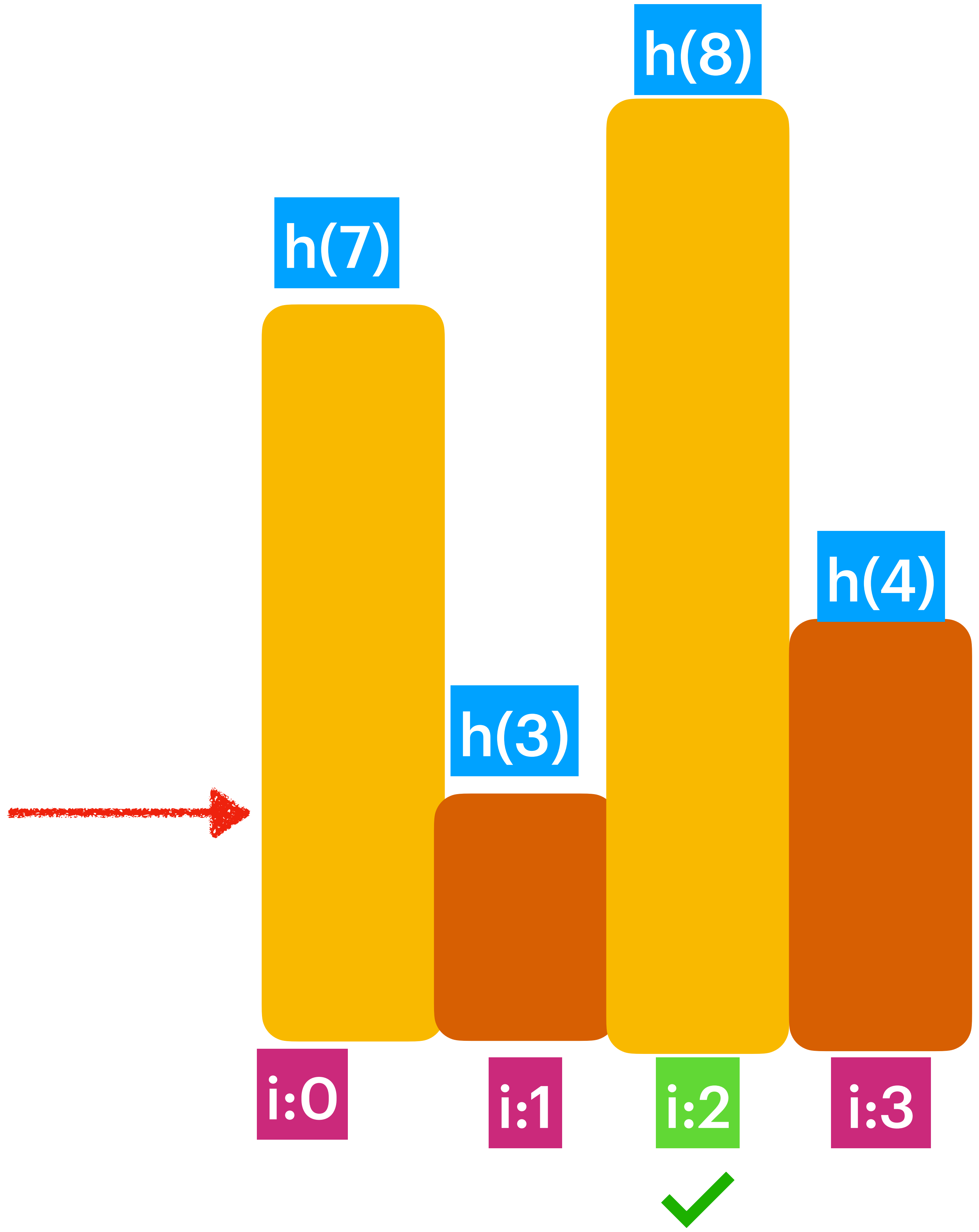
Max Area = 7

Stack —> [i:0 — v:7 ]

Here the currentValue [3] is < stack.peek() [7]
Pop the peek() : i:0 —> v:7 Area -> 1*7 = 7 > currentMax

Update the MaxArea = 7
stack[i:0 — v:3 ]

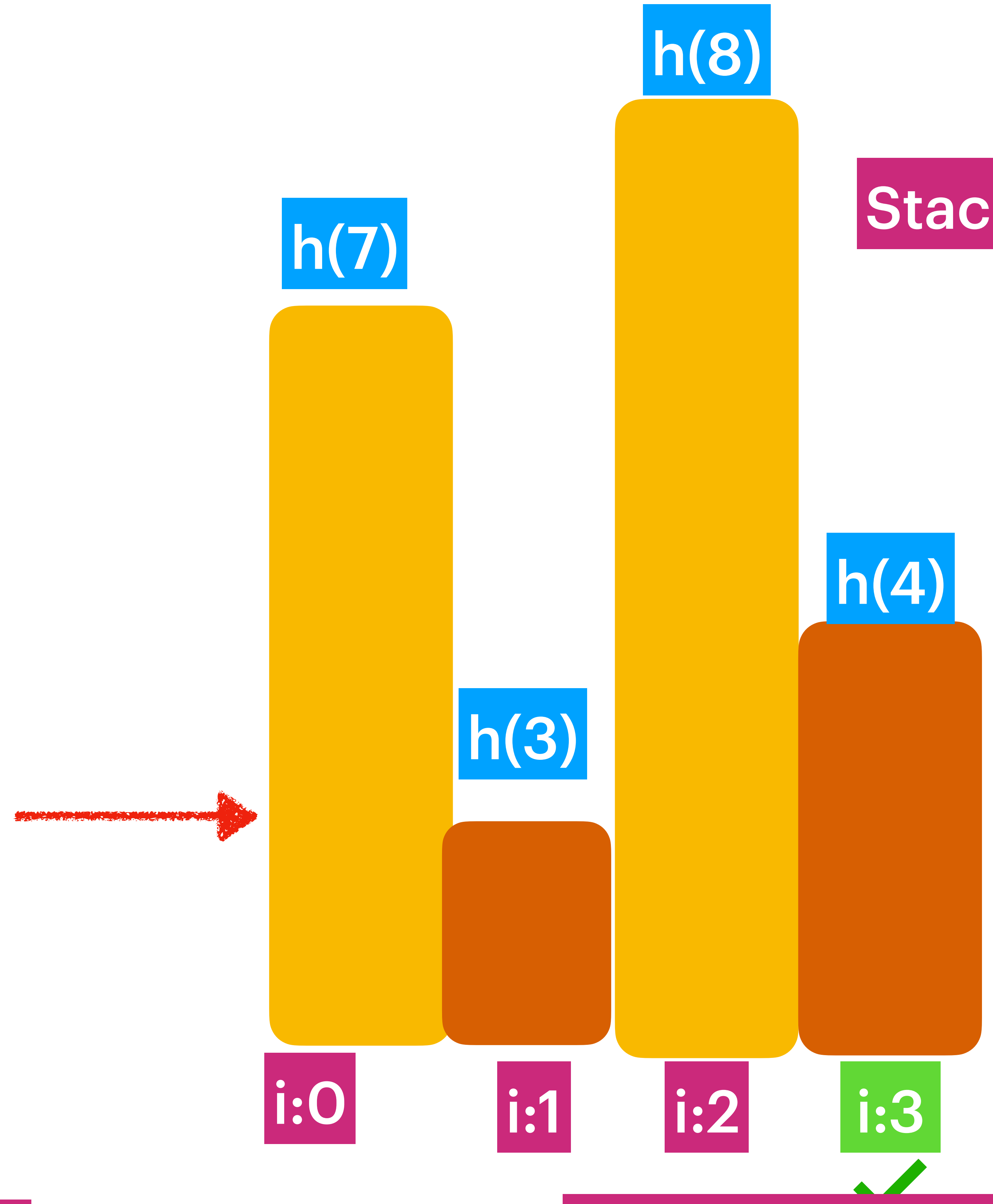
{7,3,8,4}



Here the currentValue [8] is > stack.peek() [3]
So push 8 to the stack —> i : 2 — v:8

Stack —> [(i:0 — v:3) , (i:2 — v:8) ]

Max Area = 7



Stack —> [(i:0 — v:3) , (i:2 — v:8) ]

Max Area = 8

Here the currentValue [4] is < stack.peek() [8]
Pop the peek() : i:2 —> v:8 Area -> 1*8 = 8

8 > currentMax[7]
Update the MaxArea = 8
stack will have [i:0 — v:3 ]
Now the stack.peek() [3] < currentValue [4] so
Push 4 to the stack.
[ (i:0 — v:3) , (i: 2, v:4) ]

h(7)
h(3)
h(8)
h(4)
i:0
i:1
i:2
i:3
h(7)
h(8)
h(3)
h(4)
i:0
i:1
i:2
i:3
h(7)
h(3)
h(4)
i:0
i:1
i:2
i:3

Max Area = 8

Stack —-> [ (i:0 — v:3) , (i: 2, v:4) ]

Stack.pop() =
(i:2, v:4)

Area = 2 * 4 = 8 no update on Max

Stack.pop() =
(i:0 , v:3)

Area = 4 * 3 = 12 > currentMax

MaxArea = 12

Stack isEmpty()
—> return MaxArea

Time Complexity : O(n)
Space Complexity : O(n)