Container with most water

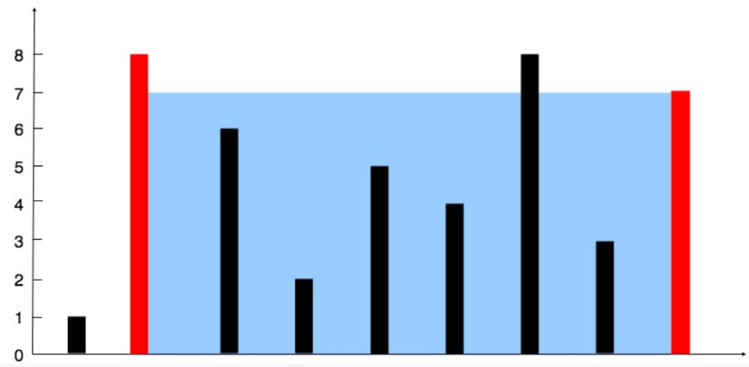
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

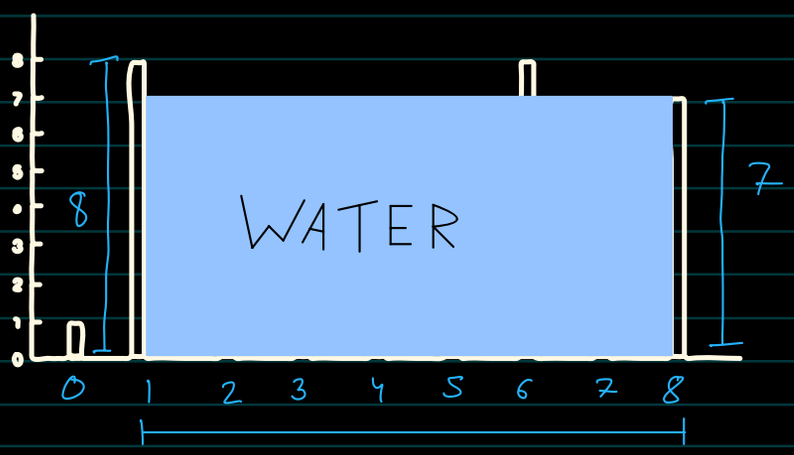
Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:

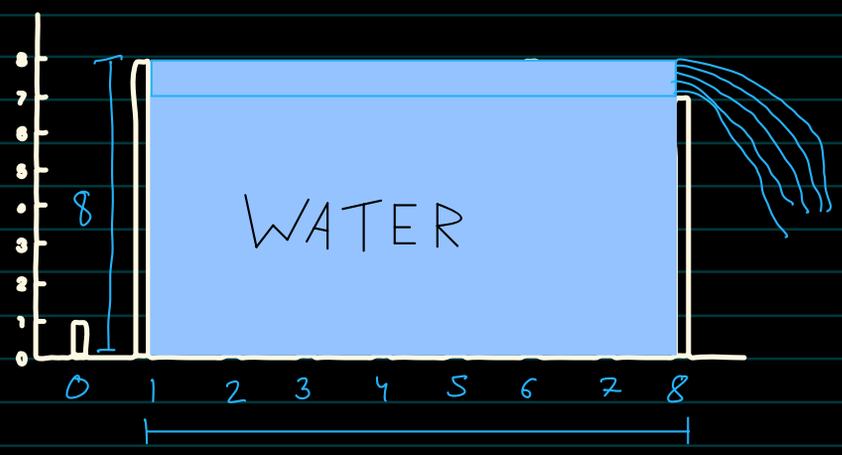



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the
```



So we have to find area of the rectangle

→ So Rectangle who have max area will be our answer but we have to note that height we are choosing must be minimum because if we choose max height then water will spill from another side.



So we can conclude that our Height will be

= min (Height H_1, Height H_2)

and our length will be the indexes of selected Height

i.e (Index of H_2 - Index of H_1) where Index H_2 > Index H_1

Brute force (TLE)

Given Array :- H [1, 8, 6, 2, 5, 4, 8, 3, 7]

i ↑ j ↑

Max area = 0

1st loop (i : 0 ⟶ H.size())

2nd loop (j : i+1 ⟶ H.size())

// Get min Height

int minHeight = min (H[i], H[j])

// Get length b/w i and j

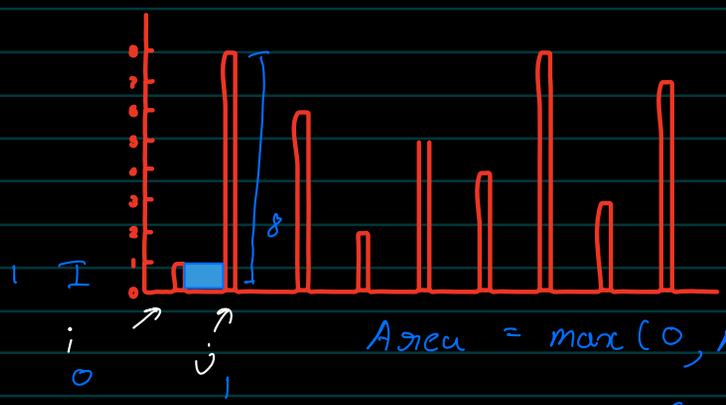
int len = j - i (j > i)

maxArea = max (maxArea, minHeight * len)

return maxArea

So In this approach we are getting area of each Rectangle present

At j = 1



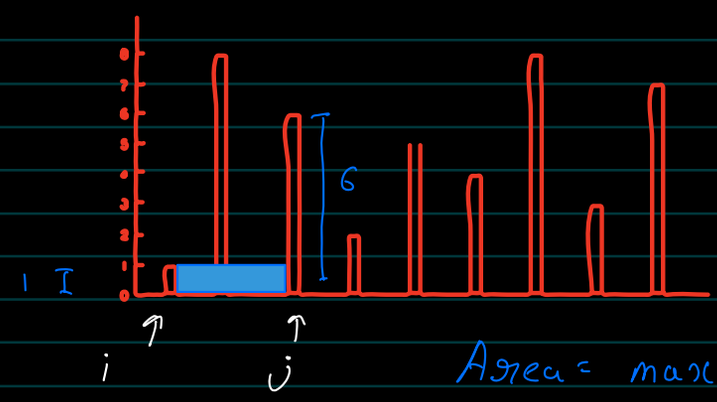
Area = max(0, min(1,8) * 1-0)

= max(0, 1*1)

= max(0, 1)

Area = 1

At j = 2

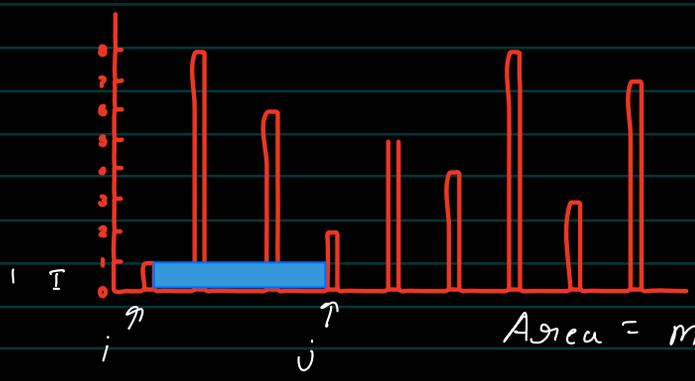


Area = max(0, min(1,6) * 2-0)

= max(0, 2)

Area = 2

At j = 3

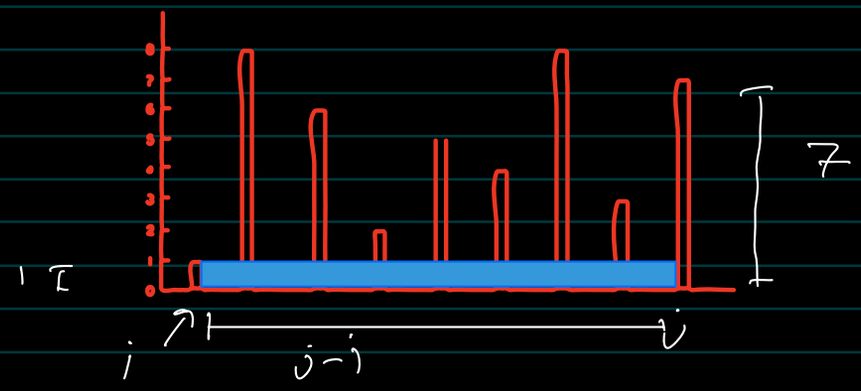


Area = max(2, (1, 2) * 3-0)

= max(2, 3)

= 3

Similarly till j = H.size()

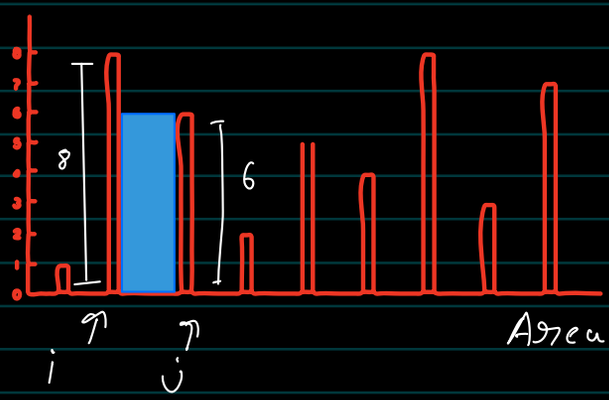


Area = max(7, min(1,7)*8-0)

= max(7, 8)

= 8

Now i++

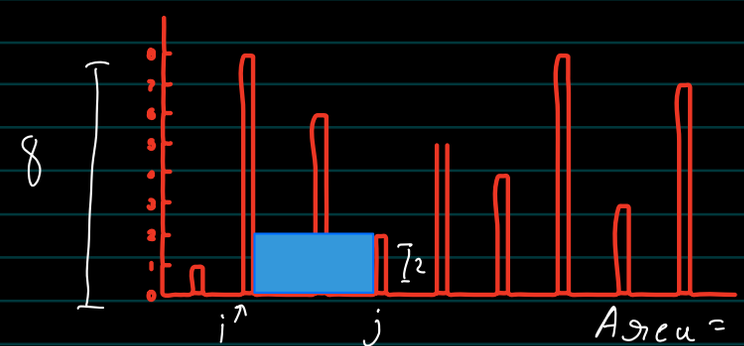


Area = max(8, min(8,6)*j-i)

= max(8, 6*2-1)

= 8

j++

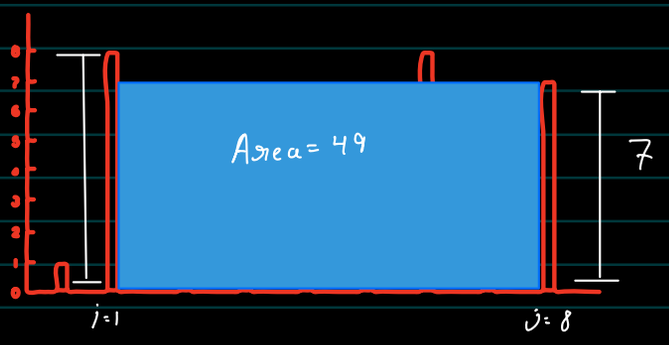


Area = max(8, min(8,2)*3-1)

= (8, 2*2)

= 8

At last we will have our ans



Area = max (Area, min (8, 7) * 8-1)

= max (Area, 7*7)

= 49

T.C - O(n^2) (Nested loop)

S.C - O (1)

Optimal solution

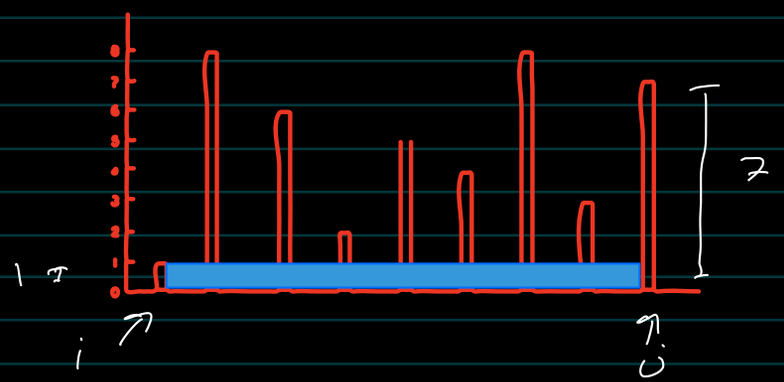
In this approach we will be using inward sliding window using 2 pointer

H [1, 8, 6, 2, 5, 4, 8, 3, 7]

i → 1

j → 7 (j = M.size()-1)

At first we will take current Height as our Area

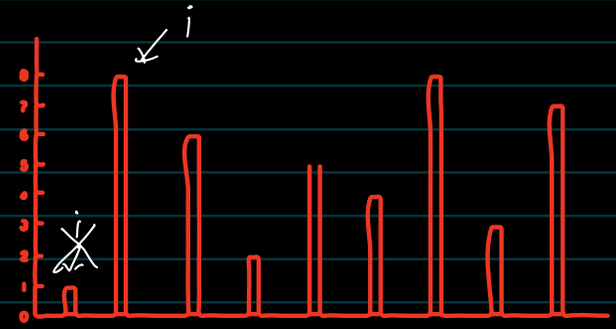


Area = max(0, min(1, 7) * 8 - 1)

= 8

Now we have to find next height which must be greater than current minHeight until i<j

i.e move i till we get H[i] ≤ minHeight && i<j



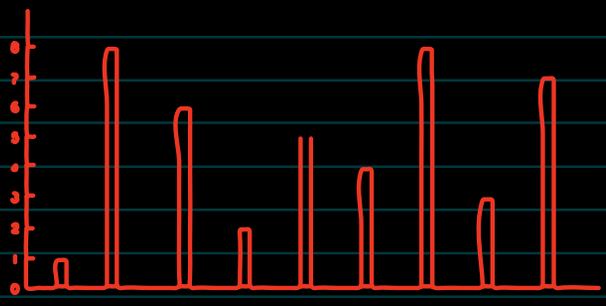
minHeight = 1, H[i] = 8

i = 1

H[i] ≤ minHeight && i<j (True)

increase i++ → for 8 False

Similarly for j because we will be selecting minHeight



minHeight = 1 j = 8

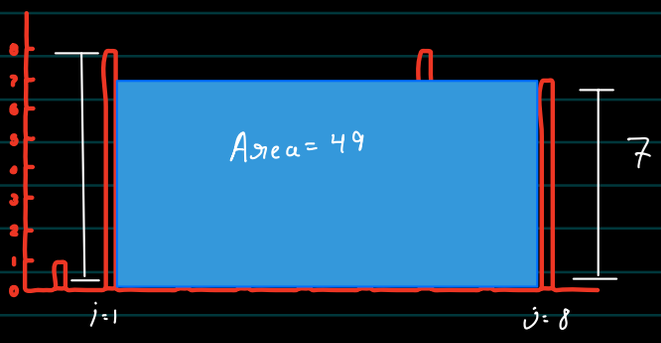
j = 8

H[j] = 7

H[j] ≤ minHeight && i<j

8 ≤ 7 False (No change)

Now Find area for new heights



Area = max (8 , min(8,7) * 8-1)

= max (8 , 7*7)

= 49

again

minHeight = 7

H[i] = 8 , i = 1

H[i] ≤ minHeight && i<j

8 ≤ 7 False

Same for j

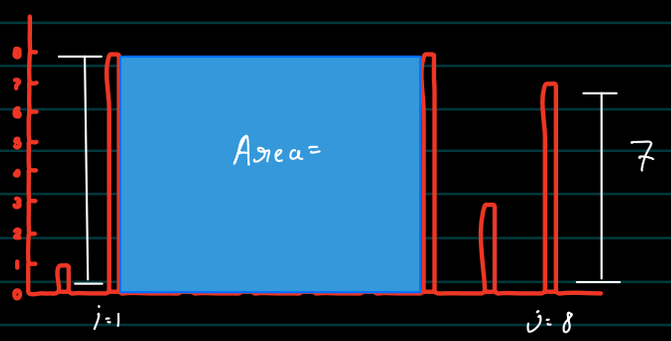
minHeight = 7

H[j] = ~~7~~ 8 ; i = ~~8~~ 7 6

H[j] ≤ minHeight && i < j (True)

j-- 7 ≤ 8 False

Find for new Heights



Area = max (49 , min(8,8) * 6-1)

= max (49, 8*5)

= 49

Again for i and j and this we don't have any larger the current one so return ans.

Brute force code

```cpp
class Solution {
public:
    int maxArea(vector<int>& height) {
        int maxvalue=0;
        for(int i=0;i<height.size();i++){
            for(int j=i+1;j<height.size();j++){
                // cout<<"max value: "<<maxvalue<<endl;
                int len=j-i;
                int minHeight=min(height[i],height[j]);
                // cout<<"new value: "<<len<<" x "<<minHeight<<" = "<<len*minHeight<<end
                maxvalue=max(maxvalue,len*minHeight);
            }
        }
        return maxvalue;

    }
};
```

Optimal code

```cpp
class Solution {
public:
    int maxArea(vector<int>& height) {
    int water = 0;
    int i = 0, j = height.size() - 1;
    while (i < j) {
        int h = min(height[i], height[j]);
        water = max(water, (j - i) * h);
        while (height[i] <= h && i < j) i++;
        while (height[j] <= h && i < j) j--;
    }
    return water;
}
};
```