42. Trapping rain water:

I/p: height = [0,1,0,2,1,0,1,3,2,1,2,1]

O/p: 6.

Brute force:

For every block (index i), water trapped above it depends on:

* Tallest bar on the left
* Tallest bar on the right.

Water at i:

water = min(maxLeft, maxRight) - height[i]

If result is negative → take 0.

Do this for every index and add.

```
class Solution {
    public int trap(int[] height){
        int n = height.length;
        int totalWater = 0;
      for(int i =0; i<n; i++){
          int leftMax = 0;
          int rightMax = 0;
       for(int j = 0; j<=i; j++){
          leftMax = Math.max(leftMax, height[j]);
       }
       for(int j = i; j<n; j++){
          rightMax = Math.max(rightmax, height[j]);
       }
```

```
totalWater += Math.min(leftMax, rightMax) -
                        height[i];
    }
    return totalWater;
  }
}
```

Time → $O(n^2)$

Optimized: (Two pointers):

* left pointer (start of array)
* right pointer (end of array)
* Track leftMax and rightMax

Always move smaller side.

```
class Solution {
    public int trap(int[] height) {
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0;
        int totalWater = 0;
        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    totalWater += leftMax - height[left];
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
```

```
            rightMax = height[right];
        }else{
            totalWater += rightMax - height[right];
        }
        right--;
    }
}
return totalWater;
    }
}
```

Time → O(n) space → O(1).