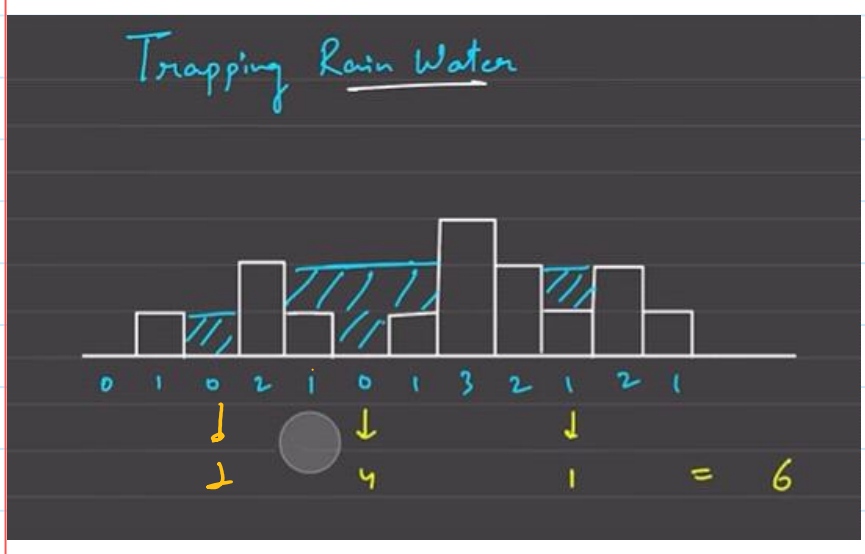
Tapping Rain water

19 February 2025 00:54



1 + 4 + 1 = 6
(Contain the water)

Example 1:

Input: height= [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Method - 1.

```
import java.util.*;
class TUF {
   static int trap(int[] arr) {
      int n = arr.length;
      int waterTrapped = 0;
      for (int i = 0; i < n; i++) {
         int j = i;
         int leftMax = 0, rightMax = 0;
         while (j >= 0) {
            leftMax = Math.max(leftMax, arr[j]);
            j--;
         }
         j = i;
         while (j < n) {
            rightMax = Math.max(rightMax, arr[j]);
            j++;
         }
         waterTrapped += Math.min(leftMax, rightMax) - arr[i];
      }
      return waterTrapped;
   }

   public static void main(String args[]) {
      int arr[] = {0,1,0,2,1,0,1,3,2,1,2,1};
      System.out.println("The duplicate element is " + trap(arr));
   }
}
```

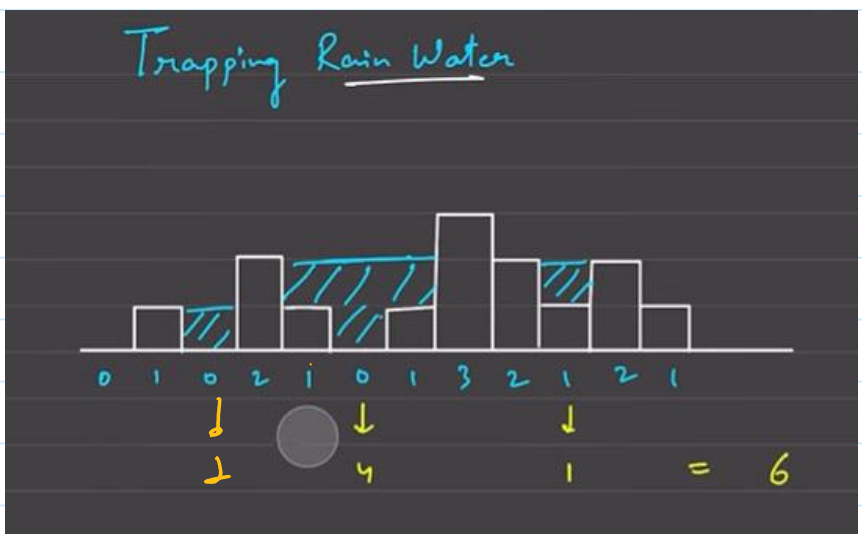
Output: The water that can be trapped is 6

Time Complexity: O(N*N) as for each index we are calculating leftMax and rightMax so it is a nested loop.

Space Complexity: O(1).

From <https://takeuforward.org/data-structure/trapping-rainwater/>

What happen in this problem



At any idx point

(left max) ← ↑ → (Right max)

Then Use the formula

total += min(leftmax, Right max) - height

(Add only value greater than zero

Where is possibility to get +ve)

At value is 3 " = (leftmax = 2 = Rightmax)
and height = 3

then,

Val = 2 - 3 = (-1) Not Added

But in Soln. idxing started from (j = i) i,e
(leftmax = Rightmax = height = 3)
Then
val = min(3,3) − 3 = 0
it's fine.
(understand the reason to use j = i)

Solution 2:Better solution

- Approach: Take 2 array prefix and suffix array
- precompute the leftMax and rightMax for each index beforehand
- use the formula min(prefix[I], suffix[i])-arr[i] to compute water trapped at each index.

Function

```
static int trap(int[] arr) {
    int n = arr.length;
    int prefix[] = new int[n];
    int suffix[] = new int[n];
    prefix[0] = arr[0];
    for (int i = 1; i < n; i++) {
        prefix[i] = Math.max(prefix[i - 1], arr[i]);
    }
    suffix[n - 1] = arr[n - 1];
    for (int i = n - 2; i >= 0; i--) {
        suffix[i] = Math.max(suffix[i + 1], arr[i]);
    }
    int waterTrapped = 0;
    for (int i = 0; i < n; i++) {
        waterTrapped += Math.min(prefix[i], suffix[i]) - arr[i];
    }
    return waterTrapped;
}
```

```
public static void main(String args[]) {
    int arr[] = {0,1,0,2,1,0,1,3,2,1,2,1};
    System.out.println("The duplicate element is " + trap(arr));
}
```

Output: The water that can be trapped is 6

Time Complexity: O(3*N) as we are traversing through the array only once. And O(2*N) for computing prefix and suffix array.

Space Complexity: O(N)+O(N) for prefix and suffix arrays.

…………………………………………………………………………

Solution 3: Optimal Solution(Two pointer approach)

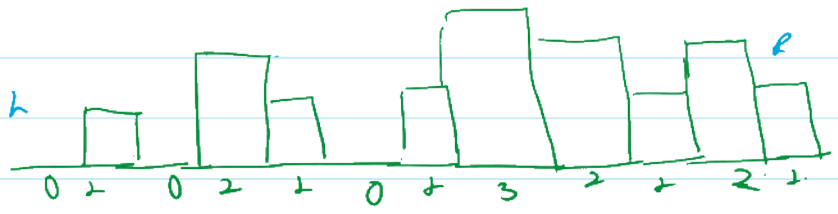
Function

```
static int trap(int[] height) {
    int n = height.length;
    int left = 0, right = n - 1;
    int res = 0;
    int maxLeft = 0, maxRight = 0;
    while (left <= right) {
        if (height[left] <= height[right]) {
            if (height[left] >= maxLeft) {
                maxLeft = height[left];
            } else {
                res += maxLeft - height[left];
            }
            left++;
        } else {
            if (height[right] >= maxRight) {
                maxRight = height[right];
            } else {
                res += maxRight - height[right];
            }
            right--;
        }
    }
    return res;
}
```

Output: The water that can be trapped is 6

Time Complexity: O(N) because we are using 2 pointer approach.

Space Complexity: O(1) because we are not using anything extra.



→ If You think like that ki greater वॉल को उठा लो the problem is that there is no water left.

Greate thing is that take smaller and .. Update the …leftmax and right max and ..move with the pointer …