Trapping Rainwater Problem

Session No.: 07
Course Name: Advanced Algorithmic Problem Solving
Course Code: R1UC601B
Instructor Name: Dr. Jitendra
Duration: 50 mins
Date of Conduction of Class: 20 Feb 2026

Recap of Previous Topic

[1-mins]

Bit Manipulation

Concept and Definition for (LO-1)
Optimized Approach
Brian Kernighan's Algorithm

```
vector<int> countBitsOptimized(int n)
{
    vector<int> result;
     for (int i = 0; i <= n; i++) {
       int count = 0, num = i;
       while (num > 0)
       {
         // Removes the last set bit
         num = num & (num - 1);
         count++;
       }
       result.push_back(count);
    }
    return result;
}
```

```
int main()
{
    int n = 5;
    vector<int> res = countBitsOptimized(n);

    for (int i = 0; i <= n; i++) {
        cout << res[i] << " ";
    }
    return 0;
}
```

Time Complexity: O(nlogn)
Each number takes O(number of set bits), which is at most O(logn). Its **Faster**

GSCALE full form and date

10

Concept and Definition for (LO-2)
Optimized Approach

A number n is a power of two **if and only if** it has exactly **one set bit** in its binary representation.

Using **bitwise AND**: $n\&(n-1)==0$

- This expression removes the lowest set bit.
- If the result is 0, then n had only **one** set bit, meaning it was a power of two.

Time Complexity: O(1)
Uses a single **bitwise AND** operation, which is extremely fast.

```
bool isPowerOfTwoOptimized(int n)
{
    return (n > 0) && ((n & (n - 1)) == 0);
}

int main()
{
    int num = 16;
    if (isPowerOfTwoOptimized(num))
        cout << num << " is a power of two." << endl;
    else
        cout << num << " is NOT a power of two." << endl;

    return 0;
}
```

Pre-Class Assessment

[2-mins]

1. Go to wooclap.com
2. Enter the event code in the top banner

Event code
ULCZPY

[1-mins]

Imagine a row of uneven bars forming a histogram, just like buildings in a city skyline. Now, picture a heavy rainfall—some areas between the bars collect water, while others let it flow away.

Your goal is to calculate how much water is trapped between these bars.

Learning Outcomes

[2-mins]

By the end of this session, You will be able to:

Explain the Trapping Rainwater problem statement and its constraints.

Implement different approaches to solve this problem.

Session Outline

- Introduce the Trapping Rainwater problem.
- Explain different approaches.
- Demonstrate optimal strategies to calculate the maximum trapped water efficiently.
- Provide hands-on coding practice to implement the optimized solution.
- Reinforce learning with an interactive quiz and discussion.

Activity-1 (Think – Pair – Share)

[1-mins]

Given an array height[ ], where each element represents the height of a bar, determine how much water can be trapped after the rain.

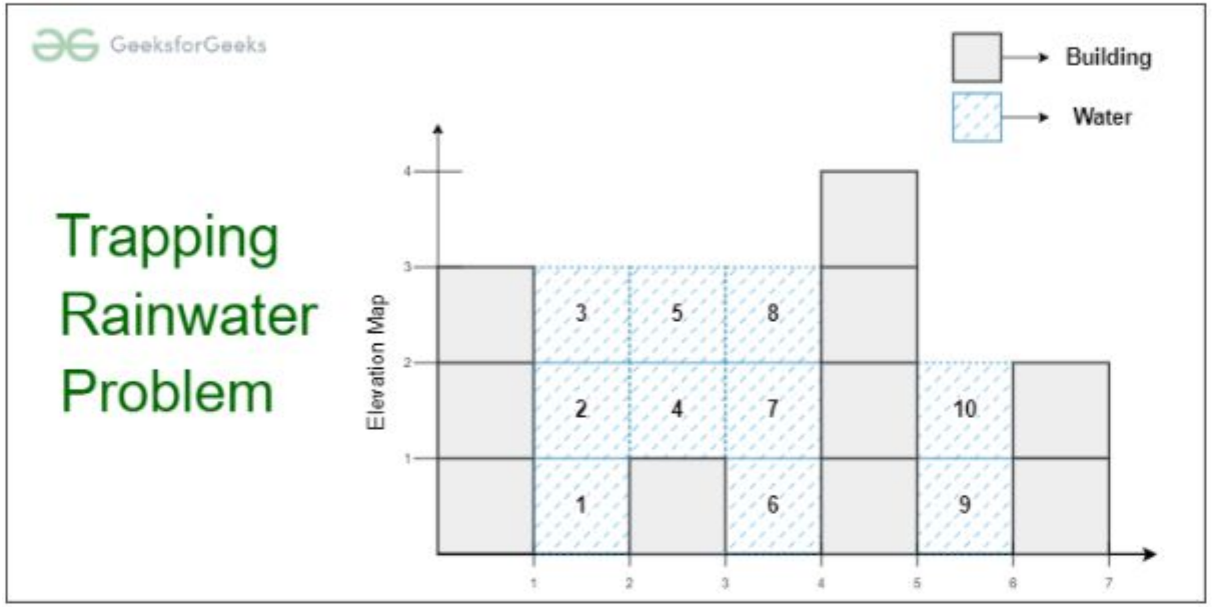
Input:
height [ ]= [4, 2, 0, 3, 2, 5]

Output: 9

Concept and Definition for (LO-1) Trapping Rainwater:

Trapping Rainwater Problem states that given an array of **n** non-negative integers **arr[ ]** representing an elevation map where the width of each bar is **1**, compute **how much water it can trap after rain**.

Examples:
Input: arr[] = [3, 0, 1, 0, 4, 0, 2]
Output: 10
Explanation: The expected rainwater to be trapped is shown in the above image.
Input: arr[] = [3, 0, 2, 0, 4]
Output: 7
Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.
Input: arr[] = [1, 2, 3, 4]
Output: 0
Explanation: We cannot trap water as there is no height bound on both sides
Input: arr[] = [2, 1, 5, 3, 1, 0, 4]
Output: 9
Explanation : We trap 0 + 1 + 0 + 1 + 3 + 4 + 0 = 9 units of water.



Trapping Rainwater Problem

GALGOTIAS UNIVERSITY

Concept and Definition for (LO-1) Trapping Rainwater:

The basic intuition of the problem is as follows:

- An element of the array can store water if there are higher bars on the left and the right.
- The amount of water to be stored in every position can be found by finding the heights of the higher bars on the left and right sides.
- The total amount of water stored is the summation of the water stored in each index.
- No water can be filled if there is no boundary on both sides.

[Naive Approach / Brute Force Approach]

Trapping Water Problem (LO-1)

Find left and right max for each index

- Traverse every array element and find the highest bars on the left and right sides.
- Take the smaller of two heights.
- The difference between the smaller height and the height of the current element is the amount of water that can be stored in this array element.

```
// Function to return the maximum water that can be stored
int maxWater(vector<int>& arr)
{
     int res = 0;
    for (int i = 1; i < arr.size() - 1; i++)
    {
       int left = arr[i];
       for (int j = 0; j < i; j++)
              left = max(left, arr[j]);

       int right = arr[i];
       for (int j = i + 1; j < arr.size(); j++)
              right = max(right, arr[j]);

       res += (min(left, right) - arr[i]);
        }
     return res;
}
```

```
int main()
{
    vector<int> arr = { 4, 3, 1, 0, 6 };
    cout << maxWater(arr);
    return 0;
}
```

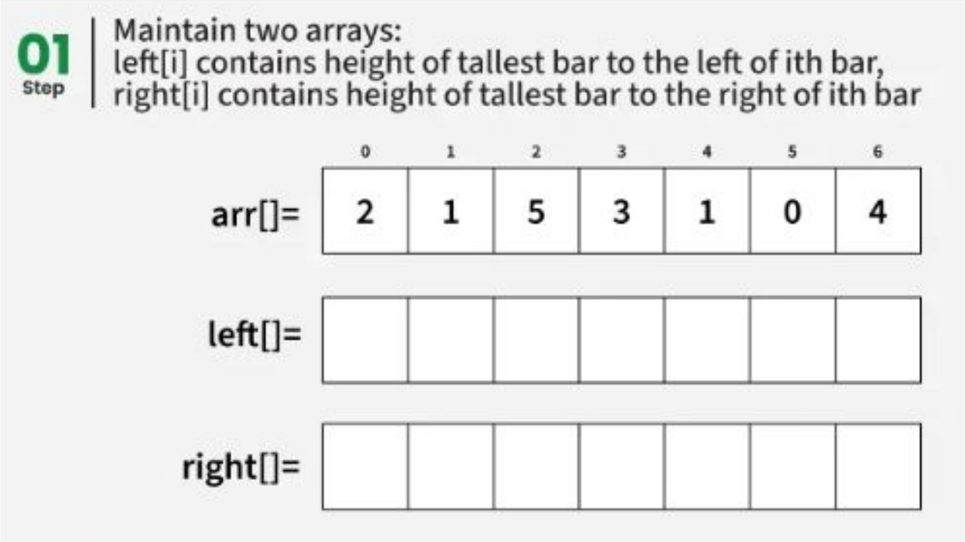
Time Complexity: O(n^2)

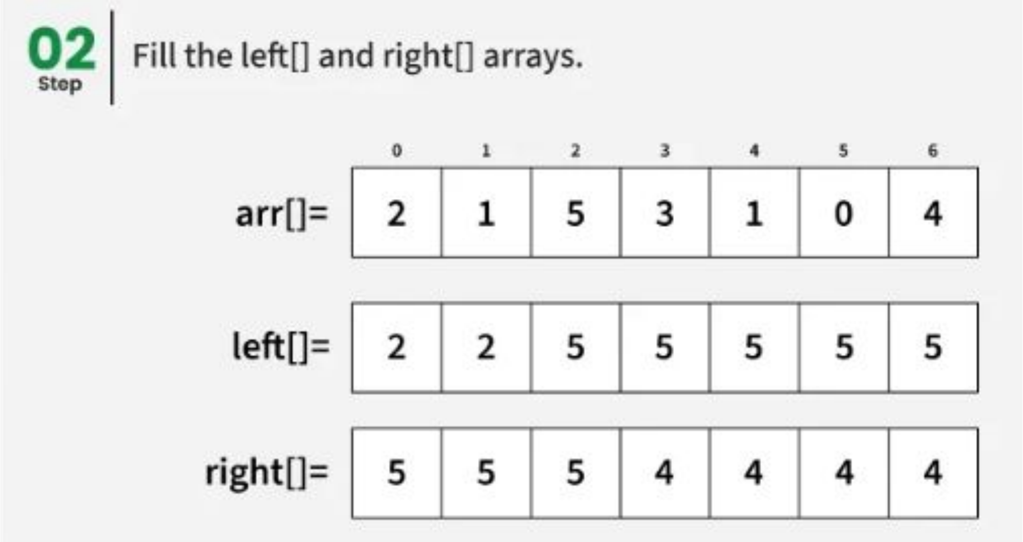
Auxiliary Space: O(1)

Concept and Definition for (LO-2) Better Approach

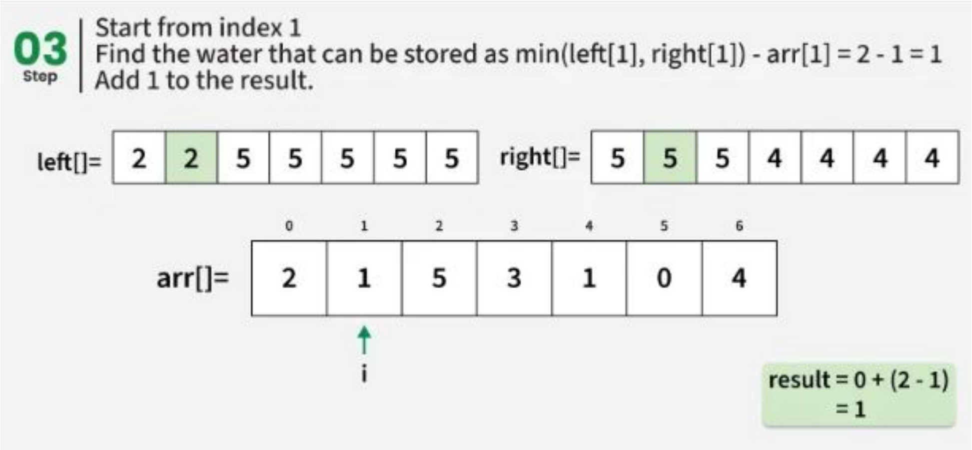
In the previous approach, for every element we needed to calculate the highest element on the left and on the right. So, to reduce the time complexity:

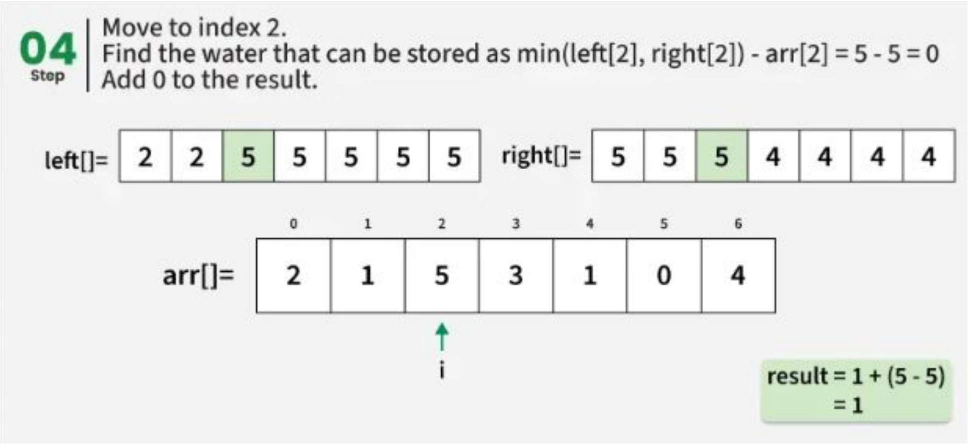
- For every element we first calculate and store the highest bar on the left and on the right (say stored in arrays **left[]** and **right[]**).
- Then iterate the array and use the calculated values to find the amount of water stored in this index, which is the same as (**min(left[i], right[i]) – arr[i]**)

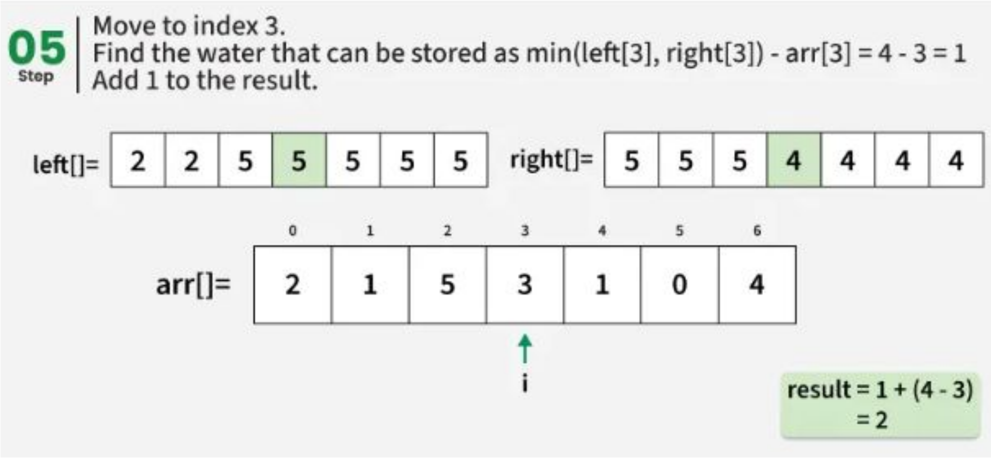


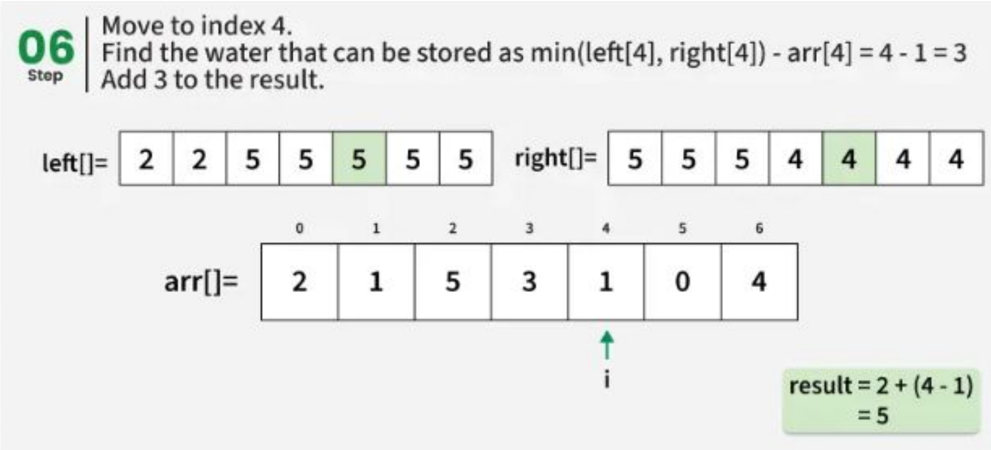


Concept and Definition for (LO-2) Better Approach







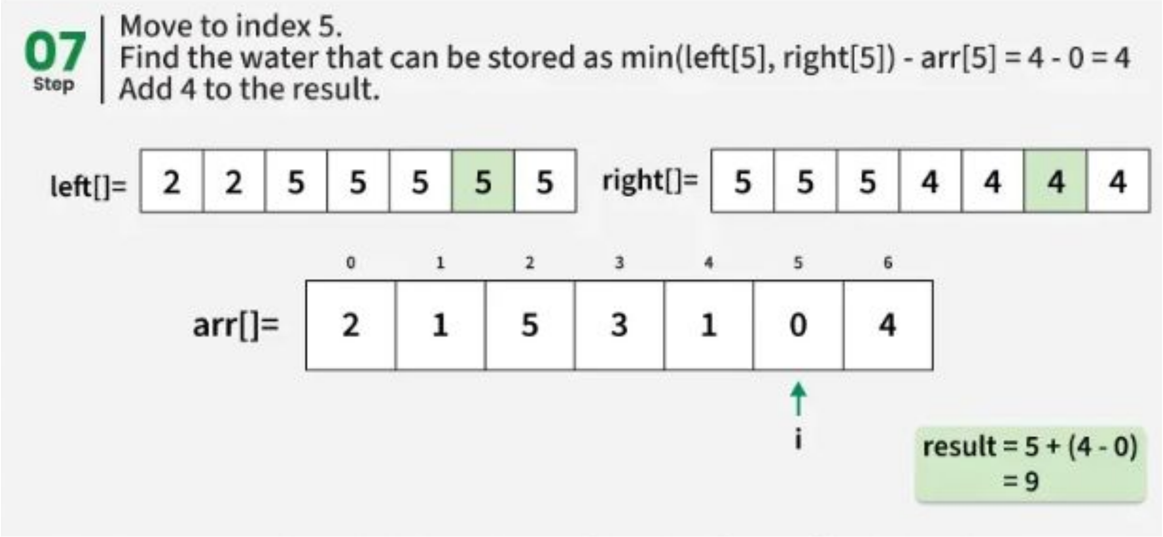


Concept and Definition for (LO-2) Better Approach

```
int maxWater(vector<int>& arr) {
    int n = arr.size();
        vector<int> left(n);
        vector<int> right(n);
    int res = 0;
    left[0] = arr[0];
        for (int i = 1; i < n; i++)
            left[i] = max(left[i - 1], arr[i]);

    right[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--)
            right[i] = max(right[i + 1], arr[i]);

    for (int i = 1; i < n - 1; i++) {
            int minOf2 = min(left[i - 1], right[i + 1]);
            if (minOf2 > arr[i]) {
                res += minOf2 - arr[i];
            }
        }
    return res;
}
```



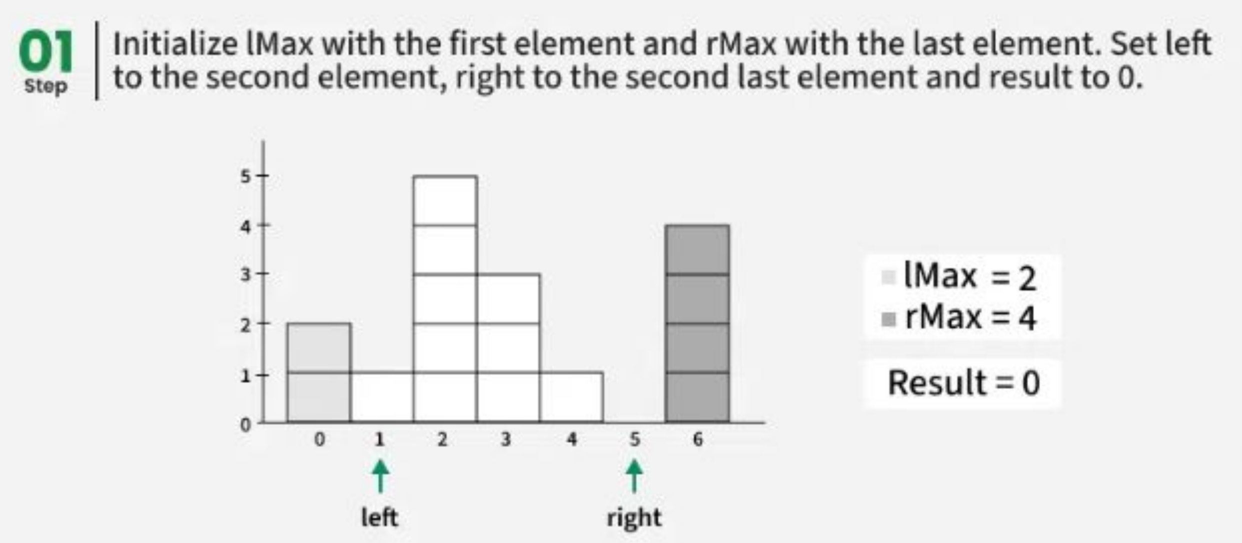
Time Complexity: O(n)

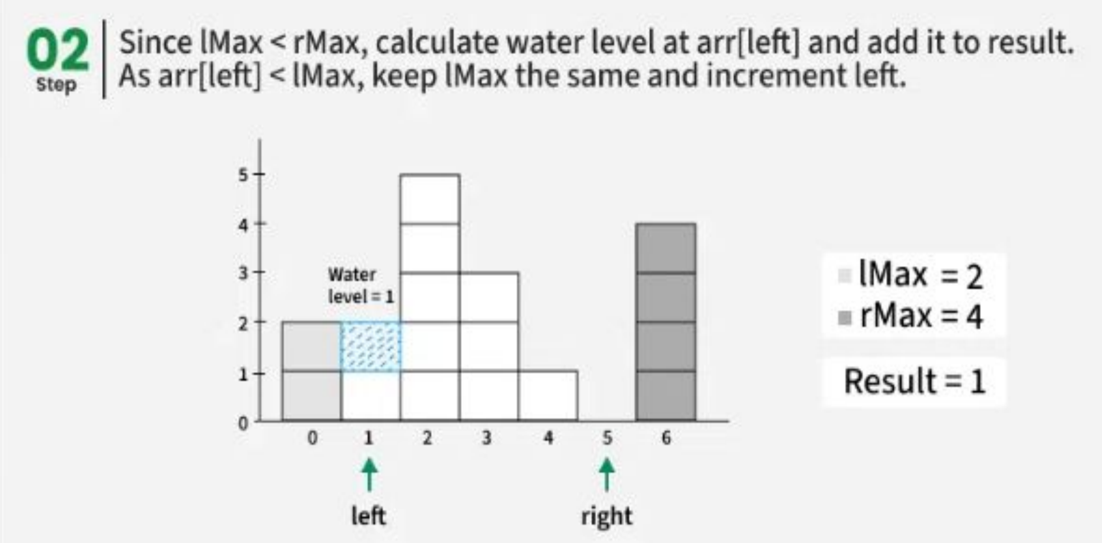
Auxiliary Space: O(n)

Concept and Definition for (LO-2) Optimized Approach

Two Pointer approach is mainly based on the following facts:

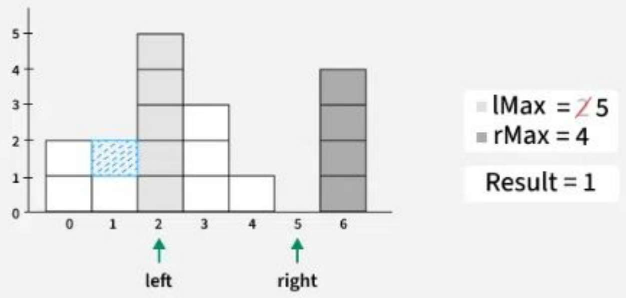
- If we consider a subarray arr[left…right], we can decide the amount of water either for arr[left] or arr[right] if we know the left max (max element in arr[0…left-1]) and right max (max element in arr[right+1…n-1].
- If left max is less than the right max, then we can decide for arr[left]. Else we can decide for arr[right]
- If we decide for arr[left], then the amount of water would be left max – arr[left] and if we decide for arr[right], then the amount of water would be right max – arr[right].



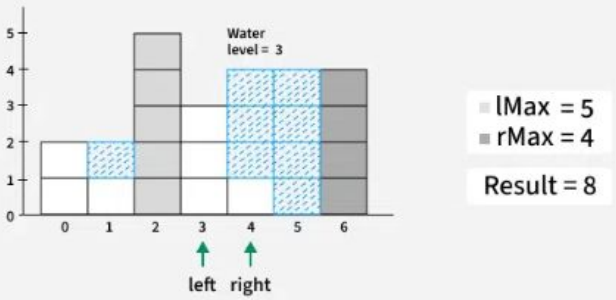


Concept and Definition for (LO-2) Optimized Approach

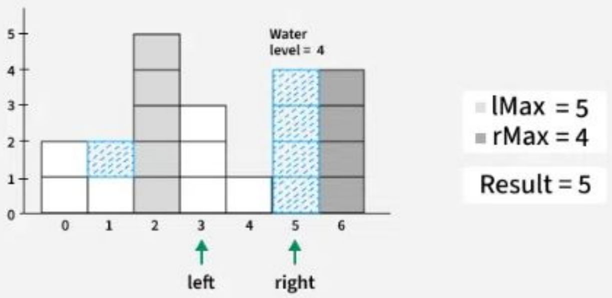
03 Step Again, lMax < rMax, so calculate water level at arr[left] and add it to result.
Now arr[left] > lMax, so update lMax to arr[left] and increment left.



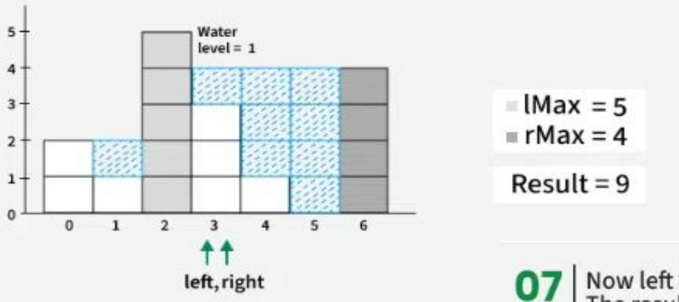
05 Step Again, lMax > rMax, so calculate water level at arr[right] and add it to result.
Since arr[right] < rMax, keep rMax the same and decrement right.



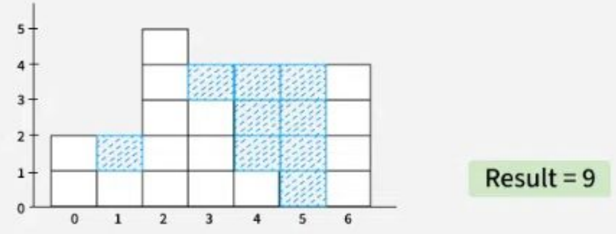
04 Step Since lMax > rMax, calculate water level at arr[right] and add it to result.
As arr[right] < rMax, keep rMax the same and decrement right.



06 Step As lMax > rMax, calculate water level at arr[right] and add it to result.
Since arr[right] < rMax, keep rMax the same and decrement right.



07 Step Now left > right, so the iteration completes.
The result holds the maximum water trapped.



Activity 2 (Pen Paper): Coding Problem

Problem Statement:

- Given an array, find trapping water units with Time Complexity O(n).
- ***Input:*** *arr[] = [3, 0, 2, 0, 4]*
  Output: *7*
  Explanation: *We trap 0 + 3 + 1 + 3 + 0 = 7 units.*

Concept and Definition for (LO-2) Optimized Approach

```
int maxWater(vector<int> &arr) {
    int left = 1;
    int right = arr.size() - 2;

    int lMax = arr[left - 1];
    int rMax = arr[right + 1];

    int res = 0;
    while (left <= right) {
        if (rMax <= lMax) {
            res += max(0, rMax - arr[right]);
            rMax = max(rMax, arr[right]);
            right -= 1;
        }
        else {
            res += max(0, lMax - arr[left]);
            lMax = max(lMax, arr[left]);
            left += 1;
        }
    }
    return res;
}
```

```
int main()
{
    vector<int> arr = {2, 1, 5, 3, 1, 0, 4};    cout
<< maxWater(arr) << endl;
    return 0;
}
```

Time Complexity: O(n), where n is the size of the given array.
Auxiliary Space: O(1)

Assessment: WooFlash Quiz

Share this link

https://app.wooflash.com/moodle/1XHK4J...

Copy

Share this code

1XHK4JQC

Copy

Summary

[1-mins]

GALGOTIAS UNIVERSITY

[Naive Approach / Brute Force Approach]

Trapping Water Problem (LO-1)

Find left and right max for each index

- Traverse every array element and find the highest bars on the left and right sides.
- Take the smaller of two heights.
- The difference between the smaller height and the height of the current element is the amount of water that can be stored in this array element.

GALGOTIAS UNIVERSITY

Concept and Definition for (LO-2) Optimized Approach

```
int maxWater(vector<int> &arr) {
    int left = 1;
    int right = arr.size() - 2;

    int lMax = arr[left - 1];
    int rMax = arr[right + 1];

    int res = 0;
    while (left <= right) {
       if (rMax <= lMax) {
          res += max(0, rMax - arr[right]);
          rMax = max(rMax, arr[right]);
          right -= 1;
       }
       else {
          res += max(0, lMax - arr[left]);
          lMax = max(lMax, arr[left]);
          left += 1;
       }
    }
    return res;
}
```

```
int main()
{
        vector<int> arr = {2, 1, 5, 3, 1, 0, 4};
        cout << maxWater(arr) << endl;
        return 0;
}
```

Time Complexity: O(n), where n is the size of the given array.
Auxiliary Space: O(1)

GSCALE full form and date 19

Learning Outcomes

[2-mins]

Ensure attainment of LO’s in alignment to the learning activities:

Explain the Trapping Rainwater problem statement and its constraints.

Implement different approaches to solve this problem.

Discussion on the post session activities

Key points:

- Understanding the Trapping Rainwater Problem: How to calculate the amount of water trapped between histogram bars.

Different Approaches:

- Brute Force ($O(n^2)$) – Time and O(1) – Space : Check water above each bar by scanning left and right heights.
- Dynamic Programming (O(n)) – Time and O(n) – Space : Pre-compute left max and right max for each index.
- Two-Pointer (O(n)) – Time and O(1) – Space : Efficiently track left and right boundaries to calculate trapped water.

Next Session:

Counting Bits: Count the number of 1s in the binary representation of numbers from 0 to n.

Power of Two: Check if a number is a power of two using bit manipulation.

Review and Reflection from students

Lec -11 Pre Class Lecture Notes DOCX

Lec-11 Pre Class Assessment [SCG] Completion

Lec-11 Activity-2 Upload code here [SCG]
Opened: Sunday, 16 March 2025, 12:00 AM **Due:** Sunday, 23 March 2025, 12:00 AM

Lec-11 Post Class Assessment [SCG] Completion