Prefix Sum

1. What is Prefix Sum?

Prefix Sum is a technique used to preprocess an array so that we can answer range sum queries efficiently.

Given an array:

```
arr = [a0, a1, a2, a3, ..., an-1]
```

The prefix sum array `pref` is defined as:

```
pref[0] = arr[0]
pref[i] = arr[0] + arr[1] + ... + arr[i]
```

Why Prefix Sum?

- Reduces repeated calculations
- Converts range sum queries from **O(n)** to **O(1)**
- Widely used in arrays, strings, matrices, and competitive programming

---

2. Basic Example

Original Array

```
arr = [2, 4, 1, 3, 5]
```

Prefix Sum Array

```
pref = [2, 6, 7, 10, 15]
```

Formula for Range Sum (L to R)

```
sum(L, R) = pref[R] - pref[L-1]
```

If `L = 0`, then:

```
sum(0, R) = pref[R]
```

---

3. Dry Run (Step-by-Step)

Array:

```
arr = [3, 1, 4, 2, 5]
```

Construct prefix sum:

i	arr[i]	pref[i]	Explanation
0	3	3	pref[0] = arr[0]
1	1	4	3 + 1
2	4	8	4 + 4
3	2	10	8 + 2
4	5	15	10 + 5

Query Example

Find sum from index `1` to `3`:

```
sum = pref[3] - pref[0]
    = 10 - 3
    = 7
```

Elements: `1 + 4 + 2 = 7`

---

4. Prefix Sum – Java Implementation Explained

Java Code

```
int[] arr = {3, 1, 4, 2, 5};
int n = arr.length;
int[] pref = new int[n];

pref[0] = arr[0];
for (int i = 1; i < n; i++) {
    pref[i] = pref[i - 1] + arr[i];
}
```

Explanation

- `pref[0]` stores the first element
- Each next value adds the current element to the previous prefix sum
- Time Complexity: **O(n)**
- Space Complexity: **O(n)**

Range Query Code

```
int left = 1;
int right = 3;

int sum;
if (left == 0)
    sum = pref[right];
else
    sum = pref[right] - pref[left - 1];
```

---

5. LeetCode Problems Based on Prefix Sum

1️⃣ LeetCode 1480 – Running Sum of 1D Array

Approach

Each element stores sum till that index.

C++ Solution

```
class Solution {
public:
    vector<int> runningSum(vector<int>& nums) {
        for (int i = 1; i < nums.size(); i++)
            nums[i] += nums[i - 1];
        return nums;
    }
};
```

Java Solution

```
class Solution {
    public int[] runningSum(int[] nums) {
        for (int i = 1; i < nums.length; i++)
            nums[i] += nums[i - 1];
        return nums;
    }
}
```

2️⃣ LeetCode 303 – Range Sum Query (Immutable)

C++ Solution

```
class NumArray {
    vector<int> pref;
public:
    NumArray(vector<int>& nums) {
        pref.resize(nums.size());
        pref[0] = nums[0];
        for (int i = 1; i < nums.size(); i++)
```

```
            pref[i] = pref[i - 1] + nums[i];
    }
    int sumRange(int l, int r) {
        if (l == 0) return pref[r];
        return pref[r] - pref[l - 1];
    }
};
```

Java Solution

```
class NumArray {
    int[] pref;
    public NumArray(int[] nums) {
        pref = new int[nums.length];
        pref[0] = nums[0];
        for (int i = 1; i < nums.length; i++)
            pref[i] = pref[i - 1] + nums[i];
    }
    public int sumRange(int l, int r) {
        if (l == 0) return pref[r];
        return pref[r] - pref[l - 1];
    }
}
```

3️⃣ LeetCode 724 – Find Pivot Index

Idea

Left sum = Right sum

C++ Solution

```
class Solution {
public:
    int pivotIndex(vector<int>& nums) {
```

```
        int total = accumulate(nums.begin(), nums.end(), 0);
        int left = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (left == total - left - nums[i])
                return i;
            left += nums[i];
        }
        return -1;
    }
};
```

Java Solution

```
class Solution {
    public int pivotIndex(int[] nums) {
        int total = 0, left = 0;
        for (int x : nums) total += x;
        for (int i = 0; i < nums.length; i++) {
            if (left == total - left - nums[i])
                return i;
            left += nums[i];
        }
        return -1;
    }
}
```

4️⃣ LeetCode 560 – Subarray Sum Equals K

Technique

Prefix Sum + HashMap

C++ Solution

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        unordered_map<int, int> mp;
        mp[0] = 1;
        int sum = 0, count = 0;
        for (int x : nums) {
            sum += x;
            if (mp.count(sum - k))
                count += mp[sum - k];
            mp[sum]++;
        }
        return count;
    }
};
```

Java Solution

```
class Solution {
    public int subarraySum(int[] nums, int k) {
        HashMap<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);
        int sum = 0, count = 0;
        for (int x : nums) {
            sum += x;
            count += map.getOrDefault(sum - k, 0);
            map.put(sum, map.getOrDefault(sum, 0) + 1);
        }
        return count;
    }
}
```

5️⃣ LeetCode 525 – Contiguous Array

Idea

Convert 0 → -1

C++ Solution

```
class Solution {
public:
    int findMaxLength(vector<int>& nums) {
        unordered_map<int, int> mp;
        mp[0] = -1;
        int sum = 0, ans = 0;
        for (int i = 0; i < nums.size(); i++) {
            sum += (nums[i] == 0 ? -1 : 1);
            if (mp.count(sum))
                ans = max(ans, i - mp[sum]);
            else
                mp[sum] = i;
        }
        return ans;
    }
};
```

Java Solution

```
class Solution {
    public int findMaxLength(int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<>();
        map.put(0, -1);
        int sum = 0, ans = 0;
        for (int i = 0; i < nums.length; i++) {
            sum += (nums[i] == 0 ? -1 : 1);
            if (map.containsKey(sum))
                ans = Math.max(ans, i - map.get(sum));
            else
                map.put(sum, i);
```

```
        }
        return ans;
    }
}
```

6️⃣ LeetCode 238 – Product of Array Except Self (Prefix Concept)

C++ Solution

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
        vector<int> ans(n, 1);
        int pref = 1, suff = 1;
        for (int i = 0; i < n; i++) {
            ans[i] *= pref;
            pref *= nums[i];
        }
        for (int i = n - 1; i >= 0; i--) {
            ans[i] *= suff;
            suff *= nums[i];
        }
        return ans;
    }
};
```

Java Solution

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] ans = new int[n];
        Arrays.fill(ans, 1);
```

```
        int pref = 1, suff = 1;
        for (int i = 0; i < n; i++) {
            ans[i] *= pref;
            pref *= nums[i];
        }
        for (int i = n - 1; i >= 0; i--) {
            ans[i] *= suff;
            suff *= nums[i];
        }
        return ans;
    }
}
```