Prefix Sum

A prefix sum is when you create a new array where each element is the sum of all previous elements including the current one in the original array. Let me show you with a simple example:

Original array: [3, 1, 4, 1, 5] Prefix sum: [3, 4, 8, 9, 14]

Here's how it's calculated:

- First element (3): Just take the first number
- Second element (4): 3 + 1
- Third element (8): 3 + 1 + 4
- Fourth element (9): 3 + 1 + 4 + 1
- Fifth element (14): 3 + 1 + 4 + 1 + 5

Example 1 range Sum C++

```
#include <iostream>
#include <vector>
using namespace std;

int rangeSum(vector<int>& nums, int L, int R) {
   int sum = 0;

   // Brute force: Sum elements from L to R
   for (int i = L; i <= R; i++) {
      sum += nums[i];
   }

   return sum;
}

int main() {
   vector<int> nums = {1, 2, 3, 4, 5}; // Example array
   int L = 1, R = 3; // Example range

   cout << "Sum of elements in range [" << L << ", " << R << "] is: " << rangeSum(nums, L, R) << endl;
   return 0;
}
```

Range Sum Java

```
class RangeSum {
   public static int calculateRangeSum(int[] nums, int start, int end) {
      int sum = 0;

      // Iterate over the specified range
      for (int i = start; i <= end; i++) {
         sum += nums[i];
      }
      return sum;
   }
   public static void main(String[] args) {
      // Example array
      int[] nums = {1, 2, 3, 4, 5};

      // Specify the range [start, end] (inclusive)
      int start = 1; // Index 1 (value 2)
```

```
        int end = 3;   // Index 3 (value 4)

        // Compute the range sum
        int rangeSum = calculateRangeSum(nums, start, end);

        // Output the result
        System.out.println("Range Sum from index " + start + " to " + end + " is: " + rangeSum);
    }


}
```

Range Sum c++ using Prefix Sum

```
#include <iostream>
#include <vector>
using namespace std;

 int  solve(vector<int> nums,int L,int R){
       int n = nums.size();
       vector<int> prefix(nums.size());
       prefix[0]=nums[0];
       for (int i = 1; i < n; i++) {
              prefix[i] = prefix[i-1] + nums[i];
        }

  for (int i = 0; i < n; i++) {
     cout <<prefix[i]<<" ";
  }
     cout <<"\n";
    // Get the sum of elements in the range [L, R]
        if (L==0)
           return prefix[R];
        return prefix[R] - prefix[L-1];

}

int main() {
    vector<int> nums = {1, 2, 3, 4, 5}; // Example array
    int L = 1, R = 3; // Example range

    cout << "Sum of elements in range [" << L << ", " << R << "] is: " << solve(nums,L, R) << endl;

    return 0;
}
```

Range Sum using Prefix Sum Java

```
class Main {
    private static int[] prefix;

    // Method to precompute prefix sum and get range sum
    public static int rangeSum(int[] nums, int L, int R) {
        int n = nums.length;
        prefix = new int[n]; // Same size as the input array
        prefix[0] = nums[0]; // Initialize the first element
```

```
        // Compute the prefix sum array
        for (int i = 1; i < n; i++) {
            prefix[i] = prefix[i - 1] + nums[i];
        }

        // Get the sum of elements in the range [L, R]
        if (L == 0)
            return prefix[R]; // If starting index is 0, return prefix[R] directly
        return prefix[R] - prefix[L - 1];
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 5}; // Example array
        int L = 1, R = 3; // Example range

        System.out.println("Sum of elements in range [" + L + ", " + R + "] is: " + rangeSum(nums, L, R));
    }
}
```

1480. Running Sum of 1d Array

C++

Space O(n)

Time O(n)

```
class Solution {
public:
    vector<int> runningSum(vector<int>& nums) {
    vector<int>v;
    int sum=0;
    for(int i=0;i<nums.size();i++){
            sum+=nums[i];
            v.push_back(sum);
    }
    return v;
    }
};
```

Space O(1)

Time O(n)

```
#include <iostream>
#include <vector>
using namespace std;

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

int main() {
    vector<int> nums = {1, 2, 3, 4};
    vector<int> result = runningSum(nums);

    cout << "Running Sum: ";
```

```
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

Java
Space O(n)
Time O(n)

```
class Solution {
    public int[] runningSum(int[] nums) {
    int[] ans = new int[nums.length];
    ans[0] = nums[0];
    for (int i = 1; i < nums.length; i++)
        ans[i] = ans[i-1] + nums[i];
    return ans;
    }
}
```

Space O(1)
Time O(n)

```
public class RunningSum1DArray {
    public static int[] runningSum(int[] nums) {
        for (int i = 1; i < nums.length; i++) {
            nums[i] += nums[i - 1];
        }
        return nums;
    }
```

303. Range Sum Query - Immutable
Space O(n)
Time O(n)
Time of range Query O(1)

```
class NumArray {
public:
    vector<int> s;

    NumArray(vector<int>& nums) {
        int n = nums.size();
        s.resize(n);
        s[0]=nums[0];
        for (int i = 1; i < n; ++i) s[i] = s[i-1] + nums[i];
    }

    int sumRange(int left, int right) {
        if (left>0)return s[right] - s[left-1];
        return s[right];
    }
};
```

or

```
class NumArray {
private:
   vector<int> prefix;

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

   int sumRange(int left, int right) {
      return prefix[right + 1] - prefix[left];
   }
};
```

Java

```
class NumArray {
   private int[] prefixSum;
   public NumArray(int[] nums) {
     int n = nums.length;
     prefixSum = new int[n + 1]; // Extra space for easier calculations
      for (int i = 0; i < n; i++) {
         prefixSum[i + 1] = prefixSum[i] + nums[i];
      }
   }

   public int sumRange(int left, int right) {
       return prefixSum[right + 1] - prefixSum[left];
   }
}
```

724. Find Pivot Index

Brute Force

```
#include <iostream>
#include <vector>
using namespace std;

// Function to find the pivot index
int pivotIndex(vector<int>& nums) {
   int n = nums.size();

   // Iterate through each index
   for (int i = 0; i < n; i++) {
      int leftSum = 0, rightSum = 0;

      // Calculate left sum
      for (int j = 0; j < i; j++) {
         leftSum += nums[j];
      }

      // Calculate right sum
```

```
        for (int j = i + 1; j < n; j++) {
            rightSum += nums[j];
        }

        // Check if left sum equals right sum
        if (leftSum == rightSum) {
            return i;
        }
    }

    return -1; // Return -1 if no pivot index is found
}

int main() {
    // Input the array
    vector<int> nums = {1, 7, 3, 6, 5, 6}; // Example input

    // Find and print the pivot index
    int result = pivotIndex(nums);
    cout << "Pivot Index: " << result << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {
    // Function to find the pivot index
    public static int pivotIndex(int[] nums) {
        int n = nums.length;

        // Iterate through each index
        for (int i = 0; i < n; i++) {
            int leftSum = 0, rightSum = 0;

            // Calculate left sum
            for (int j = 0; j < i; j++) {
                leftSum += nums[j];
            }

            // Calculate right sum
            for (int j = i + 1; j < n; j++) {
                rightSum += nums[j];
            }

            // Check if left sum equals right sum
            if (leftSum == rightSum) {
                return i;
            }
        }

        return -1; // Return -1 if no pivot index is found
    }
```

```
    public static void main(String[] args) {
        // Input the array
        int[] nums = {1, 7, 3, 6, 5, 6}; // Example input

        // Find and print the pivot index
        int result = pivotIndex(nums);
        System.out.println("Pivot Index: " + result);
    }
}
```

Prefix Sum

```
public int pivotIndex(int[] nums) {
        int totalSum = 0, leftSum = 0;

        // Calculate the total sum of the array
        for (int num : nums) {
            totalSum += num;
        }

        // Traverse the array to find the pivot index
        for (int i = 0; i < nums.length; i++) {
            if (leftSum == totalSum - leftSum - nums[i]) {
                return i;
            }
            leftSum += nums[i];
        }

        return -1; // If no pivot index is found

    }
```

Input: nums = [1,7,3,6,5,6]
total = 28
left_total = 0

[1,7,3,6,5,6]
 i

i is current number(= pivot number)
total = 28
left_total = 0
right_total = 27 (= 28 - 0 - 1)

[1,7,3,6,5,6]
   i

i is current number(= pivot number)
total = 28
left_total = 1
right_total = 20 (= 28 - 1 - 7)

right_total = left_total? → No
Add pivot number to left total

[1,7,3,6,5,6]
  i

i is current number(= pivot number)
total = 28
left_total = 8
right_total = 17 (= 28 - 8 - 3)

right_total = left_total? → No
Add pivot number to left total

[1,7,3,6,5,6]
    i

i is current number(= pivot number)
total = 28
left_total = 11
right_total = 11 (= 28 - 11 - 6)

right_total = left_total? → Yes

right_total = left_total? → No
Add pivot number to left total

1588. Sum of All Odd Length Subarrays

Brute force code in c++

```
int sumOddLengthSubarrays(vector<int>& arr) {
   int n = arr.size();
   int totalSum = 0;

   for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j += 2) {
         for (int k = i; k <= j; k++) {
            totalSum += arr[k];
         }
      }
   }
   return totalSum;
}
```

Prefix Sum in c++

```
class Solution {
public:
   int sumOddLengthSubarrays(vector<int>& arr) {
   int n = arr.size();
   vector<int> prefixSum(n + 1, 0);
   int totalSum = 0;

   // Compute prefix sums
   for (int i = 0; i < n; i++) {
```

```
        prefixSum[i + 1] = prefixSum[i] + arr[i];
    }

    // Calculate sum of all odd-length subarrays
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j += 2) {
            totalSum += prefixSum[j + 1] - prefixSum[i];
        }
    }

    return totalSum;
    }
```

Java Brute force

```
public int sumOddLengthSubarrays(int[] arr) {
    int n = arr.length;
    int totalSum = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j += 2) {
            for (int k = i; k <= j; k++) {
                totalSum += arr[k];
            }
        }
    }
    return totalSum;
}
```

Java Prefix Sum

```
class Solution {
    public int sumOddLengthSubarrays(int[] arr) {
        int n = arr.length;
        int[] prefix = new int[n + 1];

        // Build the prefix sum array
        for (int i = 0; i < n; i++) {
            prefix[i + 1] = prefix[i] + arr[i];
        }

        int totalSum = 0;

        // Calculate the sum of all odd-length subarrays
        for (int i = 0; i < n; i++) {
            for (int len = 1; i + len <= n; len += 2) {
                int j = i + len - 1;
                totalSum += prefix[j + 1] - prefix[i];
            }
        }

        return totalSum;
    }
}
```

121. Best Time to Buy and Sell Stock

C++(TLE)

```
class Solution {
public:
   int maxProfit(vector<int>& prices) {
     int n = prices.size();
   int maxProfit = 0;
   for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
         maxProfit = max(maxProfit, prices[j] - prices[i]);
      }
   }
   return maxProfit;
   }
};
```

Java(TLE)

```
class Solution {
   public int maxProfit(int[] prices) {
     int n = prices.length;
      int maxProfit = 0;
      for (int i = 0; i < n; i++) {
         for (int j = i + 1; j < n; j++) {
            maxProfit = Math.max(maxProfit, prices[j] - prices[i]);
      }
   }
   return maxProfit;
   }
}
```

Prefix Solution C++

```
class Solution {
   public int maxProfit(int[] prices) {
     int n = prices.length;
   int minPrice = Integer.MAX_VALUE;
   int maxProfit = 0;

   for (int price : prices) {
      minPrice = Math.min(minPrice, price);
      maxProfit = Math.max(maxProfit, price - minPrice);
   }

   return maxProfit;
   }
}
```

Prefix Sum(Java)

```
public int maxProfit(int[] prices) {
   int n = prices.length;
   int minPrice = Integer.MAX_VALUE;
   int maxProfit = 0;

   for (int price : prices) {
      minPrice = Math.min(minPrice, price);
      maxProfit = Math.max(maxProfit, price - minPrice);
   }
```

```
    return maxProfit;
}
```

560. Subarray Sum Equals K

C++ brute force

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
    int count = 0;
    int n = nums.size();
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += nums[j];
            if (sum == k) {
                count++;
            }
        }
    }
    return count;
    }
};
```

Java

```
class Solution {
    public int subarraySum(int[] nums, int k) {
    int count = 0;
    int n = nums.length;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += nums[j];
            if (sum == k) {
                count++;
            }
        }
    }
    return count;
    }
}
```