Prefix Sum

Contents

[I. Introduction 2](#_Toc199776422)

[Understanding the Problem with Examples 2](#_Toc199776423)

[Naive Approach 2](#_Toc199776424)

[Time and Space Complexity 3](#_Toc199776425)

[II. Prefix Sum 4](#_Toc199776426)

[1. Real-World Analogy: Cricket Overs 4](#_Toc199776427)

[2. Prefix Sum Concept 4](#_Toc199776428)

[3. Variations 4](#_Toc199776429)

[Time and Space Complexity 6](#_Toc199776430)

[III. Equilibrium Index 7](#_Toc199776431)

[1. Brute Force Solution 7](#_Toc199776432)

[Time and Space Complexity 7](#_Toc199776433)

[2. Use Prefix Sum 8](#_Toc199776434)

[Edge Case Handling 8](#_Toc199776435)

[Time and Space Complexity 8](#_Toc199776436)

[IV. Sum of Even Elements in a Range 9](#_Toc199776437)

[Use Prefix Sum (Ignoring Odd Numbers) 9](#_Toc199776438)

[Time and Space Complexity 10](#_Toc199776439)

# Introduction

* You're given:
  + An array A of n elements.
  + q queries.
    - Each query gives you two indices l and r.
  + You must calculate the sum of elements from index l to r (inclusive).

### Understanding the Problem with Examples

* Let’s say the array is:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Array: | -3 | 6 | 2 | 7 | 5 | 2 | 8 | -9 | 3 | 1 |

* Example Queries:
  + Query (4, 8) → 5 + 2 + 8 + (-9) + 3 = 9
  + Query (6, 9) → 8 + (-9) + 3 + 1 = 3
  + Query (0, 4) → -3 + 6 + 2 + 7 + 5 = 17
  + Query (7, 7) → -9

### Naive Approach

* For each query, iterate from l to r.
  + Keep a sum variable to accumulate values.
  + Output or store the result.

vector<int> prefixSumNaive(const vector<int>& vec,

const vector<pair<size\_t, size\_t>> queries) {

vector<int> answer;

for (const auto& [l, r] : queries) {

int sum = 0;

for (auto i = l; i <= r; i++)

sum += vec[i];

answer.push\_back(sum);

}

return answer;

}

/\*

For query [4, 8]: 9

For query [6, 9]: 3

For query [0, 4]: 17

For query [7, 7]: -9

\*/

### Time and Space Complexity

#### Time Complexity Analysis

* Time per Query:
  + Each query may take up to O(n) time (worst case, l = 0, r = n-1).
* Total Time for q Queries:
* Example:
  + If and , then
  + Total operations:
* If we remember, in we can perform operations, then total time take for the above is:

#### Space Complexity

* No extra space is used for calculating the sum → space.
* But we may store answers in an output array of size q.
* Should this output array be counted?
  + We don’t count output space in space complexity by convention, unless it is used for algorithmic processing.
  + Reason: It is required by the problem, not created to aid the algorithm.

# Prefix Sum

## Real-World Analogy: Cricket Overs

* You’re given **cumulative scores after each over**. Example:

| **Over Number** | **Cumulative Score** |
| --- | --- |
| 1 | 2 |
| 2 | 8 |
| 3 | 14 |
| 4 | 29 |
| … | … |
| 10 | 97 |

* Runs in Over 10 = Score[10] - Score[9] = 97 - 88 = 9
* Runs in Over 6–10 = Score[10] - Score[5] = 97 - 31 = 66
* This is the core intuition behind **prefix sums**!

## Prefix Sum Concept

* A Prefix Sum array stores the cumulative sum from the start (index 0) up to every index i.
* Example: Given array: [2, 5, 7, 3, 2]
  + Prefix Sum: [2, 7, 14, 17, 19]
  + prefix[i] = A[0] + A[1] + ... + A[i]
  + Sum of A[l to r] = prefix[r] - prefix[l - 1]
  + If l == 0, then: prefix[r] is the answer.

## Variations

* Separate Array (Standard Prefix Sum)
  + Store prefix sums in a new array to keep the original array unchanged.
* In-Place Prefix Sum
  + Build prefix sum in the same array to save space.
  + Modify the original array directly.
* Retrieve the Original Array (In-Place Restore)
  + After modifying the array with prefix sums, restore it to its original form.
  + Use backward iteration and subtract to recover the original values.

|  |
| --- |
| // 01 Separate Array  vector<int> prefixSumSaperatePSArray(const vector<int>& vec,  const vector<pair<size\_t, size\_t>> queries)  {  size\_t len = vec.size();  // First create prefix sum.  vector<int> ps(len); // Allocate memory  ps[0] = vec[0];  for (size\_t i = 1; i < len; i++) {  ps[i] = vec[i] + ps[i - 1];  }  vector<int> answer;  for (const auto& [l, r] : queries) {  if (l == 0)  answer.push\_back(ps[r]);  else  // sum[4, 8] = sum[0, 8] - sum[0, 3]  // sum[4, 8] = ps[8] - ps[3] = ps[r] - ps[l -1]  answer.push\_back(ps[r] - ps[l - 1]);  }  return answer;  }  // 02 In-place  vector<int> prefixSumInplace(vector<int>& vec,  const vector<pair<size\_t, size\_t>> queries)  {  size\_t len = vec.size();  for (size\_t i = 1; i < len; i++) {  vec[i] = vec[i] + vec[i - 1];  }  vector<int> answer;  for (const auto& [l, r] : queries) {  if (l == 0)  answer.push\_back(vec[r]);  else  answer.push\_back(vec[r] - vec[l - 1]);  }  return answer;  }  // 03 Retrieve the original array (In place)  vector<int> prefixSumInplaceRestore(vector<int>& vec,  const vector<pair<size\_t, size\_t>> queries)  {  size\_t len = vec.size();  // In-place Prefix Sum Calculation  for (size\_t i = 1; i < len; i++) {  vec[i] = vec[i] + vec[i - 1];  }    // Answer the queries  vector<int> answer;  for (const auto& [l, r] : queries) {  if (l == 0)  answer.push\_back(vec[r]);  else  answer.push\_back(vec[r] - vec[l - 1]);  }  // Restore the Original Array  for (size\_t i = len -1 ; i >= 1 ; i--) {  vec[i] = vec[i] - vec[i - 1];  }  return answer;  } |

### Time and Space Complexity

| **Variation** | **Time Complexity** | **Space Complexity** | **Modifies Input** | **Restores Input** |
| --- | --- | --- | --- | --- |
| Separate Array | O(n) | O(n) | No | Not needed |
| In-Place Prefix Sum | O(n) | O(1) | Yes | No |
| In-Place + Restore | O(n) | O(1) | Yes | Yes |

# Equilibrium Index

* An equilibrium index in an array is an index i such that:
  + Sum of elements to the left of i = Sum of elements to the right of i

A[0] + A[1] + ... + A[i-1] == A[i+1] + ... + A[n-1]

* + Special Cases:
    - If i == 0: Left sum is 0
    - If i == n-1: Right sum is 0
* Example:

Array: [3, 3, 5, 3, 3]

Index: 0 1 2 3 4

* Index 2:
* Left sum = 3+3 = 6
* Right sum = 3+3 = 6
* So, index 2 is an equilibrium index.

## Brute Force Solution

int countEquilibriumIndexNaive(

const vector<int>& vec) {

int count = 0;

auto len = vec.size();

for (size\_t i = 0;i < len;i++) {

int right\_sum = 0;

for (size\_t j = 0;j < i;j++)

{

right\_sum += vec[j];

}

int left\_sum = 0;

for (size\_t j = i + 1;j < len;j++)

{

left\_sum += vec[j];

}

if (right\_sum == left\_sum)

count++;

}

return count;

}

* Keep a count variable.
* Check for every index
  + Calculate left sum.
  + Calculate right sum.
  + Check if left sum == right sum
    - Increment count

### Time and Space Complexity

* Outer loop → O(n)
* Inner loops → Each does up to O(n) work

## Use Prefix Sum

* Step 1: Build Prefix Sum Array
* Step 2: For each index i, compute:
  + Left Sum = Sum[0, i-1] => PS[i-1]
  + Right Sum = SUM[i+1, N-1] =>PS[N-1] – PS[i]
  + Check if left sum == right sum
    - Increment the count;

### Edge Case Handling

| **Index** | **Condition** | **Result** |
| --- | --- | --- |
| i = 0 | Left sum = 0 (no elements before index 0) | Handle with if (i==0) |
| i = n-1 | Right sum = 0 (no elements after last index) | PS[N-1] – PS[i] =0 |

|  |
| --- |
| int countEquilibriumIndex(const vector<int>& vec)  {  auto len = vec.size();  vector<int> ps(len);  ps[0] = vec[0];  for (size\_t i = 1; i < len; i++)  ps[i] = vec[i] + ps[i - 1];  int count = 0;  for (size\_t i = 0; i < len; i++)  {  auto sum\_left = i == 0 ? 0 : ps[i - 1];  auto sum\_right = ps[len - 1] - ps[i];  if (sum\_left == sum\_right)  count++;  }  return count;  } |

### Time and Space Complexity

# Sum of Even Elements in a Range

* You are given: An array A of size n and q queries.
* Each query has two indices L and R.
  + Your task: For each query, compute the sum of even numbers in the index range [L, R].
* Example:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Array: | 2 | 3 | 2 | 4 | 2 | 9 | 8 | 7 | 7 | 6 |

* Sample Queries:
  + Query1-[0, 5] -Even elements: 2, 2, 4, 2 → sum = 10
  + Query2-[5, 5] - No even elements → sum = 0
  + Query3-[3, 7] - Even: 4, 2, 8 → sum = 14

### Use Prefix Sum (Ignoring Odd Numbers)

* You’ll build a prefix sum array, but:
  + Treat odd numbers as 0
  + Add only even numbers

|  |
| --- |
| vector<int> SumOfEvenElements(const vector<int>& vec,  const vector<pair<size\_t, size\_t>> queries)  {  auto len = vec.size();  vector<int> ps(len);  ps[0] = vec[0] % 2 == 0 ? vec[0] : 0;  for (size\_t i = 1; i < len; i++) {  ps[i] = ps[i - 1] + (vec[i] % 2 == 0 ? vec[i] : 0);  }  vector<int> answer;  for (const auto& [l, r] : queries) {  if (l == 0)  answer.push\_back(ps[r]);  else  answer.push\_back(ps[r] - ps[l - 1]);  }  return answer;  } |

### Time and Space Complexity

* Prefix sum building
* Answering all queries
* Total
* Prefix array space