1. **Prefix Sum**

Prefix sum is the technique where you precompute & store the cumulative sum of the sequence of elements that allows fast sum calculation of any continuous range.

Let's say we have a sequence of elements A as mentioned below-

A = {a0, a1, a2, a3, a4, a5}

so Prefix Sum P will be calculated as

P=  {p0, p1, p2, p3, p4, p5}

where-

p0 = a0

p1 = a1 + a0

p2 = a0 + a1 + a2

p3 = a0 + a1 + a2 + a3

p4 = a0 + a1 + a2 + a3 + a4

p5 = a0 + a1 + a2 + a3 + a4 + a5

Q) Say we need to sum get sum of all elements from indices   
 [2 to 5] => [a2 + a3 + a4 + a5] or [p5 - p1]

[1 to 4] => [a1 + a2 + a3 + a4]   or [p4 - p0]  
 [0 to 4] => [a0 + a1 + a2 + a3 + a4] or [p4]

# Revision Notes: Prefix Sum and Query Optimization

## Introduction to Prefix Sum

### Basics of Prefix Sum

A prefix sum array is an array that contains the cumulative sum of elements of the original array. The prefix sum array for an array `A` is defined as `pf[i]`, which stores the sum of elements from the start up to index `i` of the array `A`.

### Example

Given an array `A`:

A = [2, 5, -1, 7, 1]

The prefix sum array `pf` would be calculated as follows:

pf[0] = A[0] pf[i] = pf[i-1] + A[i] for i > 0

So, the prefix sum array `pf` will be:

pf = [2, 7, 6, 13, 14]

## Calculating the Prefix Sum Array

### Step-by-Step Construction

1. Initialize the prefix sum array `pf` of the same length as array `A`.

2. Assign the first element:

pf[0] = A[0]

3. For each subsequent element, sum it with the previous cumulative sum:

pf[i] = pf[i-1] + A[i]

### Code Implementation

#### Brute Force Approach

```cpp

int pf[N];

for (int i = 0; i < N; i++) {

int sum = 0;

for (int j = 0; j <= i; j++) {

sum += A[j];

}

pf[i] = sum;

}

**Optimized Approach**

int pf[N];

pf[0] = A[0];

for (int i = 1; i < N; i++) {

pf[i] = pf[i-1] + A[i];

}

This optimized approach reduces redundant calculations and computes the prefix sum array in O(N) time.

**Query Optimization using Prefix Sum**

**Problem Description**

Given an array of integers and multiple queries asking for the sum of elements in a specified range [L, R], efficiently calculate the sum for each query using the prefix sum array.

**Example**

Consider the original array A:

A = [1, 2, 3, 4, 5]

Prefix sum array pf:

pf = [1, 3, 6, 10, 15]

To find the sum of elements from index 1 to 3 (A[1] to A[3]):

sum = pf[3] - pf[0] = 10 - 1 = 9

**Query Result Calculation**

Use the prefix sum array to calculate the sum of elements in constant time O(1) for any range query [L, R]:

1. If L is 0, then the sum from 0 to R is:
2. sum = pf[R]
3. Otherwise, the sum from L to R is:
4. sum = pf[R] - pf[L-1]

**Time and Space Complexity**

**Brute Force Query**

For each query, the time complexity would be O(R - L + 1), which can be very inefficient for large arrays and multiple queries.

**Optimized Query with Prefix Sum**

* Computing the prefix sum array: O(N)
* Answering each query: O(1)
* Total time complexity for Q queries: O(N + Q)

**Space Complexity**

* The space complexity of storing the prefix sum array is O(N).

**Special Cases and Variations**

**Even and Odd Indexed Prefix Sums**

* Create separate prefix sum arrays for even and odd indexed elements to handle specific types of queries.

**Example**

For an array A:

A = [2, 3, 1, 6, 4, 5]

Even indexed prefix sum array PSe:

PSe = [2, 2, 3, 3, 7, 7]

Odd indexed prefix sum array PSo:

PSo = [0, 3, 3, 9, 9, 14]

This allows efficiently computing the sum for ranges that involve only even or odd indexed elements.

**Special Index**

Count the number of "special indexes" where removing the element at that index makes the sum of even indexed elements equal to the sum of odd indexed elements in the resultant array.

int countSpecialIndexes(int A[], int n) {

int pfe[n], pfo[n];

// Build prefix arrays for even and odd indexed elements

for (int i = 0; i < n; i++) {

if (i == 0) {

pfe[0] = (i % 2 == 0) ? A[i] : 0;

pfo[0] = (i % 2 == 1) ? A[i] : 0;

} else {

pfe[i] = pfe[i-1] + ((i % 2 == 0) ? A[i] : 0);

pfo[i] = pfo[i-1] + ((i % 2 == 1) ? A[i] : 0);

}

}

int count = 0;

for (int i = 0; i < n; i++) {

int evenSum = (i == 0) ? 0 : pfe[i-1] + (pfo[n-1] - pfo[i]);

int oddSum = (i == 0) ? 0 : pfo[i-1] + (pfe[n-1] - pfe[i]);

if (evenSum == oddSum) count++;

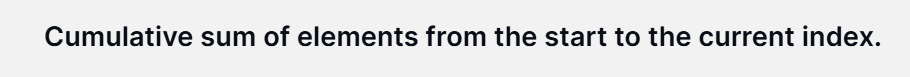
}

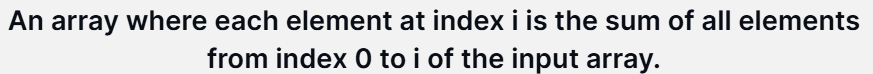
return count;

}

These steps ensure optimized and efficient handling of multiple range queries using the prefix sum approach.

These notes should provide a comprehensive overview and revision of the key concepts of prefix sums and query optimizations discussed in the class .





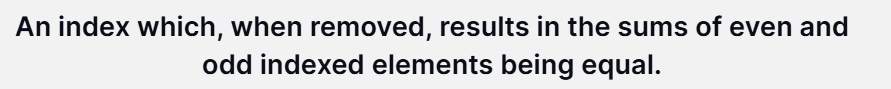




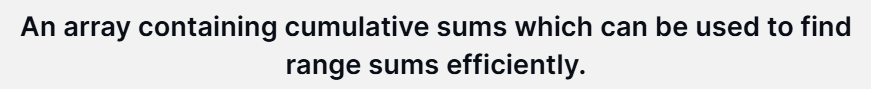






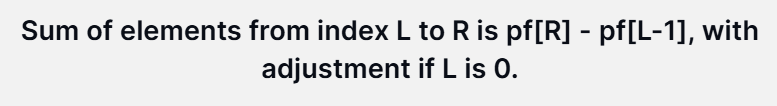




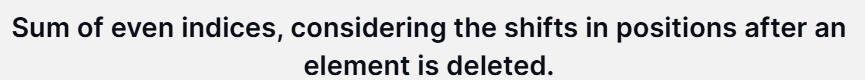




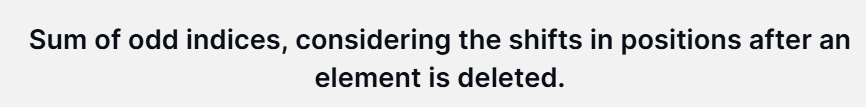




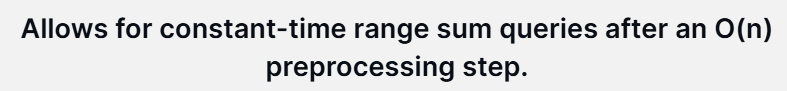












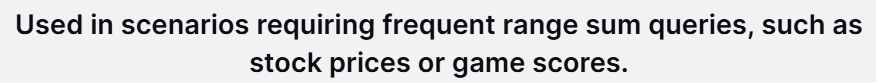




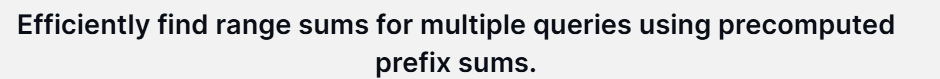




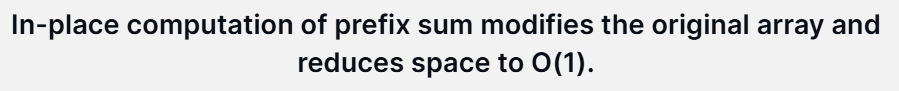




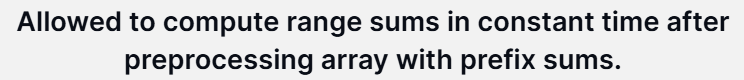




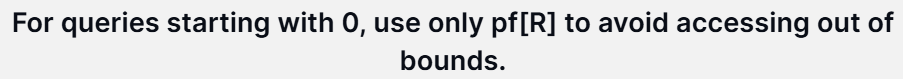




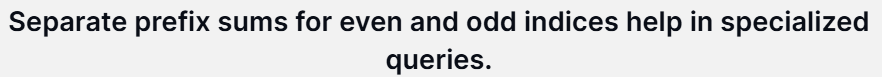




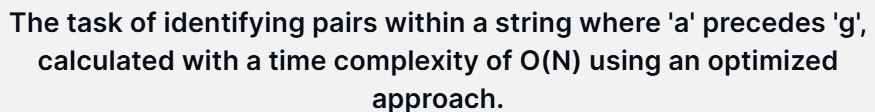




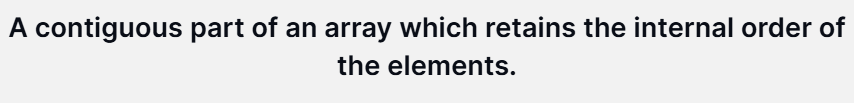




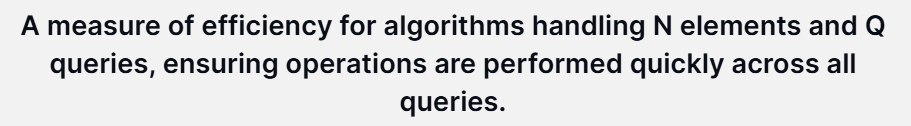




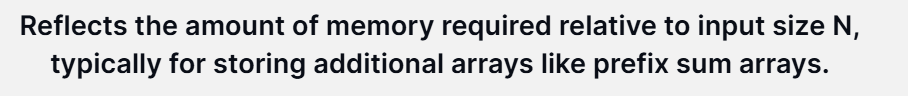




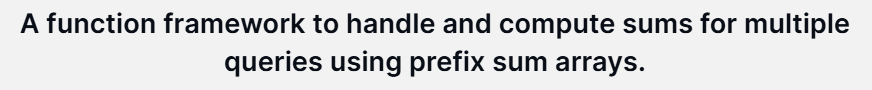




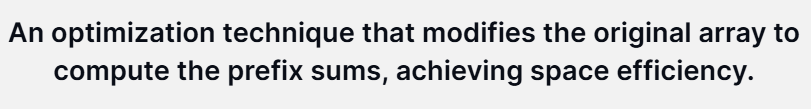




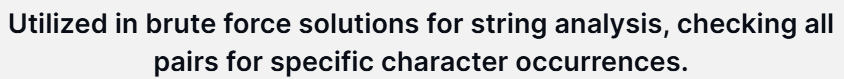












**Comprehensive Revision Notes on Prefix Sums and Special Indices**

**Introduction to Prefix Arrays**

A **prefix array** is a cumulative sum array which facilitates efficient range query answers. It allows for quick computation of sums over subarrays without iterating over the elements multiple times. This concept can be applied to both even and odd indexed elements separately, as well as the entire array.

**Creating a Prefix Array**

To create a prefix array for a given array A, the following process is used:

1. **Start with the first element**: The first element of the prefix array Pf[0] is the first element of the array A[0].
2. **Iterate through the array**: For each subsequent element, add it to the sum of all previous elements calculated so far.
   * For instance, Pf[i] = Pf[i-1] + A[i], which efficiently computes the cumulative sum upto index i.

**Example of a Prefix Array**

Given an array A = [2, 5, -1, 7, 1], its prefix array PF would be created as follows:

* PF[0] = 2
* PF[1] = 2 + 5 = 7
* PF[2] = 2 + 5 - 1 = 6
* PF[3] = 2 + 5 - 1 + 7 = 13
* PF[4] = 2 + 5 - 1 + 7 + 1 = 14

This means PF[i] represents the sum of all elements from 0 to i【4:17†transcript.txt】.

**Prefix Arrays for Even and Odd Indexed Elements**

For some applications, only the sum of even or odd indexed elements is relevant.

**Even Indexed Prefix Sum Array**

To compute the prefix sum only for even indexed elements, treat all odd indexed elements as 0:

* If i is even, PSe[i] = PSe[i-1] + A[i]
* If i is odd, PSe[i] = PSe[i-1]

For example, given A = [2, 4, 3, 1, 5], the even indexed prefix sum array would be: [2, 2, 5, 5, 10]【4:18†typed.md】.

**Odd Indexed Prefix Sum Array**

Similarly, for odd indexed elements:

* If i is odd, PSo[i] = PSo[i-1] + A[i]
* If i is even, PSo[i] = PSo[i-1]

This mechanism ensures that you are only summing the relevant indices【4:18†typed.md】.

**Special Indices in an Array**

Special indices are those which, when removed, make the sum of all even indexed elements equal to the sum of all odd indexed elements. To determine special indices:

1. **Create Prefix Sum Arrays**: Construct prefix arrays for both even and odd indexed elements of the original array A.
2. **Compute Differences**: For each index i, compute the sum of odd and even indexed elements after hypothetical removal of i.
3. **Special Condition**: An index i is special if the modified sums of even and odd indexed elements are equal【4:11†typed.md】.

**Example Calculation**

For an array A[] = [4, 3, 2, 7, 6, -2]:

* If we remove indices 0 or 2, the remaining subarrays satisfy the condition where the sum of even indexed elements equals the sum of odd indexed elements after recalculating their positions【4:11†typed.md】.

**Real-World Application**

One practical application of prefix sums is in financial computations, such as calculating cumulative profit or loss over a specified range of days, where prefix sums help in efficient query resolutions【4:15†transcript.txt】.

**Conclusion**

Understanding and utilizing prefix arrays is crucial for efficiently handling range queries and related computations. They form the basis for more advanced data processing techniques and have broad applications in both competitive programming and real-world scenarios.

**Revision Notes: Arrays, Subarrays, and Prefix Sums**

**Overview**

This session covered the fundamentals of working with arrays, focusing on subarrays and the prefix sum technique. The lecture aimed to build a strong foundation for understanding these concepts, which are crucial in solving range query problems efficiently.

**Key Concepts**

**Arrays and Subarrays**

* **Array**: A collection of elements identified by index or key.
* **Subarray**: A contiguous part of an array. To print all possible subarrays, you iterate over each element, fixing a starting point and varying the endpoint.

**Printing Subarrays**

* **Nested Loop Approach**:
  + Use three loops:
    - The outer loop fixes the starting index.
    - The middle loop fixes the ending index.
    - The innermost loop prints elements from the start to the end index inclusive.
  + Example:
  + for i from 0 to n-1:
  + for j from i to n-1:
  + for k from i to j:
  + print(array[k])
  + Time Complexity: O(n^3) due to the three nested loops【4:13†transcript】.

**Prefix Sum**

* **Prefix Sum Array (Prefix Array)**: An array where each element at index i is the sum of the elements from the start up to i.
* **Objective**: Quickly calculate the sum of elements between any two indices L and R.

**Prefix Sum Calculation**

* **Initial Approach**:
  + Calculate prefix sums using naive method:
  + prefix\_sum[0] = array[0]
  + for i from 1 to n-1:
  + prefix\_sum[i] = prefix\_sum[i-1] + array[i]
  + Optimized with a single pass to avoid recalculating sums repeatedly【4:17†transcript】.

**Calculating Range Sums**

* **Using Prefix Sums**:
  + To find the sum from index L to R with prefix array pf:
  + if L == 0:
  + sum = pf[R]
  + else:
  + sum = pf[R] - pf[L-1]
  + This allows obtaining sums efficiently in constant time after an O(n) preprocessing step to create the prefix array【4:14†transcript】【4:16†transcript】.

**Analogies for Better Understanding**

* **Cricket Scores Analogy**: Calculating runs scored in a match over specific overs can be compared to calculating subarray sums. Prefix sums can quickly provide scores for specific ranges without recounting from the start every time【4:14†transcript】.

**Practice and Implementation**

* Emphasis was placed on solving problems using pen and paper to mentally build algorithm approaches.
* Understanding problems involving prefix sums, carry forward techniques, and variations like queries on even-indexed elements was encouraged【4:3†transcript】【4:18†transcript】.

These revision notes capture the main ideas and methods discussed in the class, focusing on subarrays and the effective use of prefix sums in solving range query problems. Practicing these techniques on different problems will strengthen grasp over these foundational concepts.