Prefix Sums

- Harsh Gupta

Goal

- Learn about 1D prefix sums
- Learn about 2D prefix sums

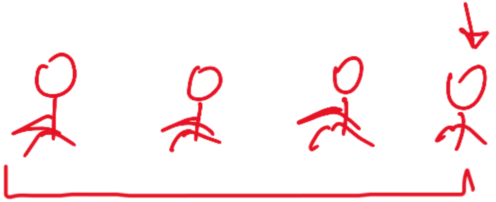
Prefix Sums

- Prefix sum is a powerful technique that can be used to preprocess an array to facilitate fast subarray sum queries without modifying the original array.

- Prefix[i] = sum of all elements in array from index **0** to **i**
  - Example
    - Arr = [1, 2, 9, -1, -2, 3]
    - Prefix = [1, 3, 12, 11, 9, 12]

arr → [2, 6, 9, 1, 4]

prefix sum → [2, 2+6, 2+6+9, 2+6+9+1, 2+6+9+1+4]



arr → [4, 6, 9, 12, 11]

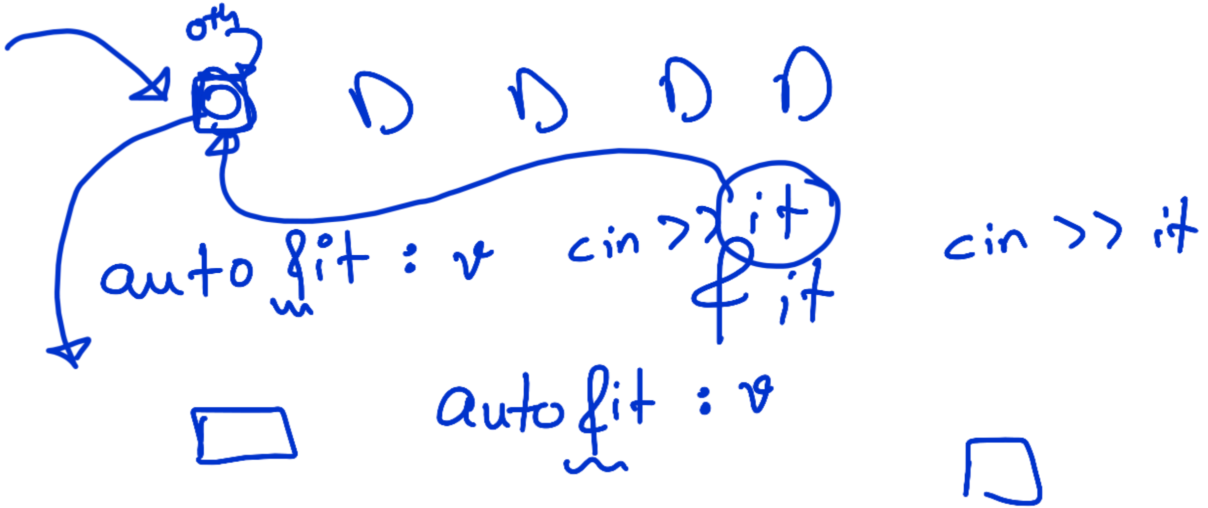
q queries

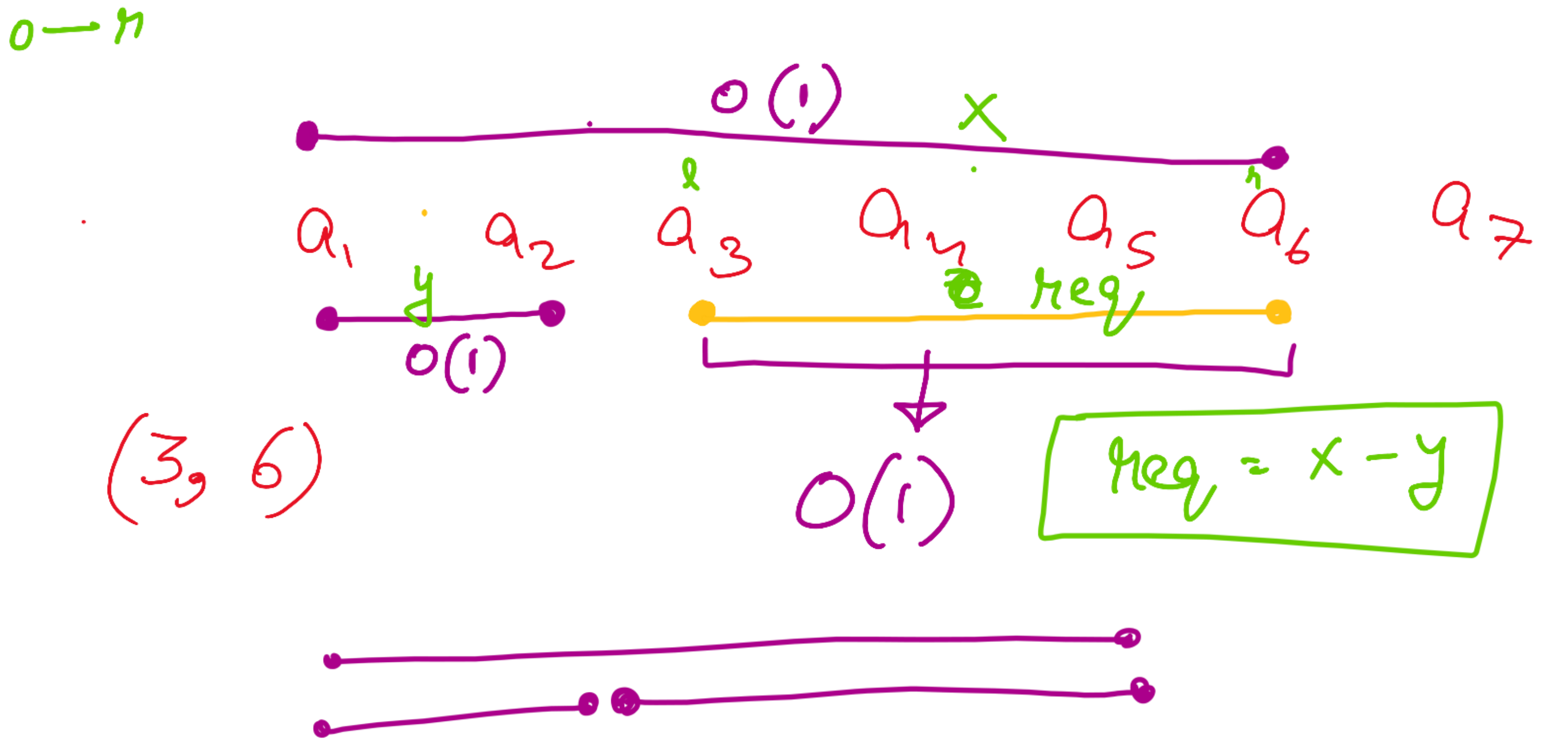
(l, r) sum of elements from l to r

- (0, 4)
- (1, 3)
- (2, 5)

→ Input of the array.

→ For every query run from l to r and get the sum.





$$\boxed{req = x - y}$$

			2		4		
arr →	2	9	6	3	7	2	1
pre →	2	11	17	20	27	29	30

$[2, 4]$

$\downarrow$

$pre[4] - pre[1]$

$27 - 11 = \underline{\underline{16}}$

If I have prefix sum vector with me

I can answer every query in $O(1)$.

$$\downarrow$$

$[l, r]$

$a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5$

$a_0 \quad a_0+a_1 \quad a_0+a_1+a_2 \quad a_0+a_1+a_2+a_3 \quad a_0+a_1+a_2+a_3+a_4 \quad \dots$

$l=0$
$r=5$

$a_0+a_1+a_2 \quad p[2]+a_3 \quad p[3]+a_4$

$p[3]$

$p[0] = a[0]$

$$\boxed{p[i] = p[i-1] + a[i]}$$

$l=0$
$r=5$

$pre[5]$

$$\boxed{pre[r] - pre[l-1]}$$

Implementation

Brute Force: $O(N^2)$

```
vector<int> arr(n), prefix(n);
for(int i = 0; i < n; i++){
    cin >> arr[i];
}
for(int i = 0; i < n; i++){
    for(int j = 0; j < i; j++){
        prefix[i] += arr[j];
    }
}
```

Optimised: $O(N)$

```
vector<int> arr(n), prefix(n);
for(int i = 0; i < n; i++){
    cin >> arr[i];
}
prefix[0] = arr[0];
for(int i = 1; i < n; i++){
    prefix[i] = arr[i] + prefix[i - 1];
}
```

Problem: Subarray sum queries in O(1)

Given an array of N elements, find the sum of subarrays for Q queries. Each query will contain 2 integers L and R, find the sum of all values in the array from L to R

why prefix sum.

Solution

Sum from L to R = Sum from 0 to R - Sum from 0 to (L - 1)

Sum from L to R = Prefix[R] - Prefix[L - 1]

Time complexity to answer every query: O(1)

Precomputation time: O(N)

Total time complexity: O(N + Q)

2D Prefix Sums

2D prefix sums are similar to 1D prefix sums, but extended to 2 dimensional arrays or grids.

n^2

2	6	5	9	10
3	(l_1, r_1) 4	1	6	1
2	7	3	2	2
1	3	5	(l_2, r_2) 4	3
6	7	9	11	4

q $\boxed{q \times n}$

$\boxed{q \times n^2}$

Find the sum of elements in coloured rectangle in O(1)

Approach

1+2+4+9
+3+1+1+2
+6+4+2+9
+5+4+0+0

j

	(0, 0) 1	2	4	9	2
	3	1	1	2	1
	6	4	2	9	0
i	5	4	0	0 (i, j)	2
	6	7	4	3	1

P[i][j] = sum of all elements of the form grid[x][y] such that x <= i and y <= j

How is this useful?

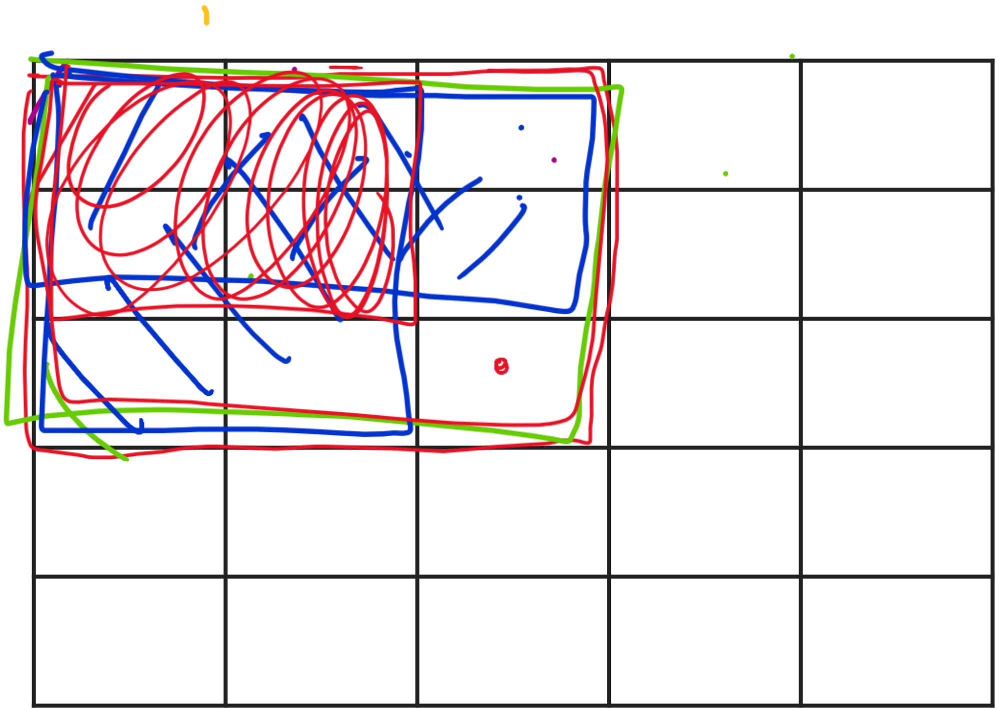
1	2	0	1	4
3	3	0	2	2
0	3	(l_1, r_1) 1	0	2
3	0	2	(l_2, r_2) 1	0
1	2	0	1	2

$$P[l_2][r_2] - P[l_1-1][r_2] - P[l_2][r_1-1] + P[l_1-1][r_1-1]$$

$22 - 12 - 15$

9

$= 4$



n^4

n^2

$$P[i][j] = P[i][j-1] + P[i-1][j] - P[i-1][j-1] + a[i][j]$$

Constructing P[i][j] in O(n * m)

```
// Assume arr[n][m] is already populated
vector<vector<int>> P(n, vector<int>(m));

for(int i = 0; i < n; i++)
    P[i][0] = arr[i][0] + (i > 0 ? P[i - 1][0] : 0);

for(int i = 0; i < m; i++)
    P[0][i] = arr[0][i] + (i > 0 ? P[0][i - 1] : 0);

for(int i = 0; i < n; i++)
    for(int j = 0; j < m; j++)
        P[i][j] = arr[i][j] + P[i][j - 1] + P[i - 1][j] - P[i - 1][j - 1];
```

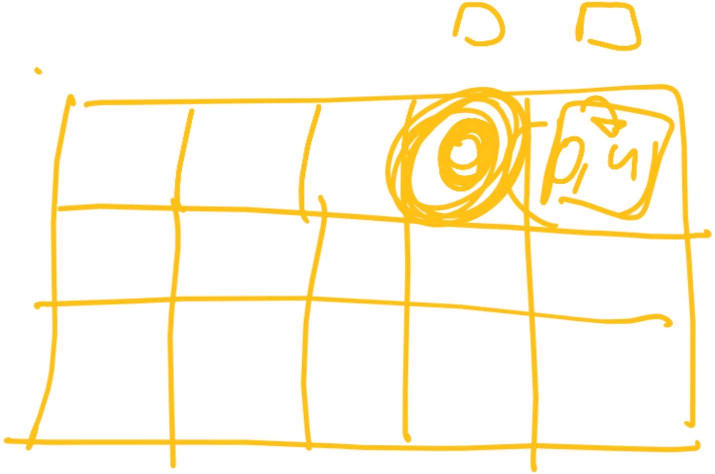
How to answer a query in O(1)

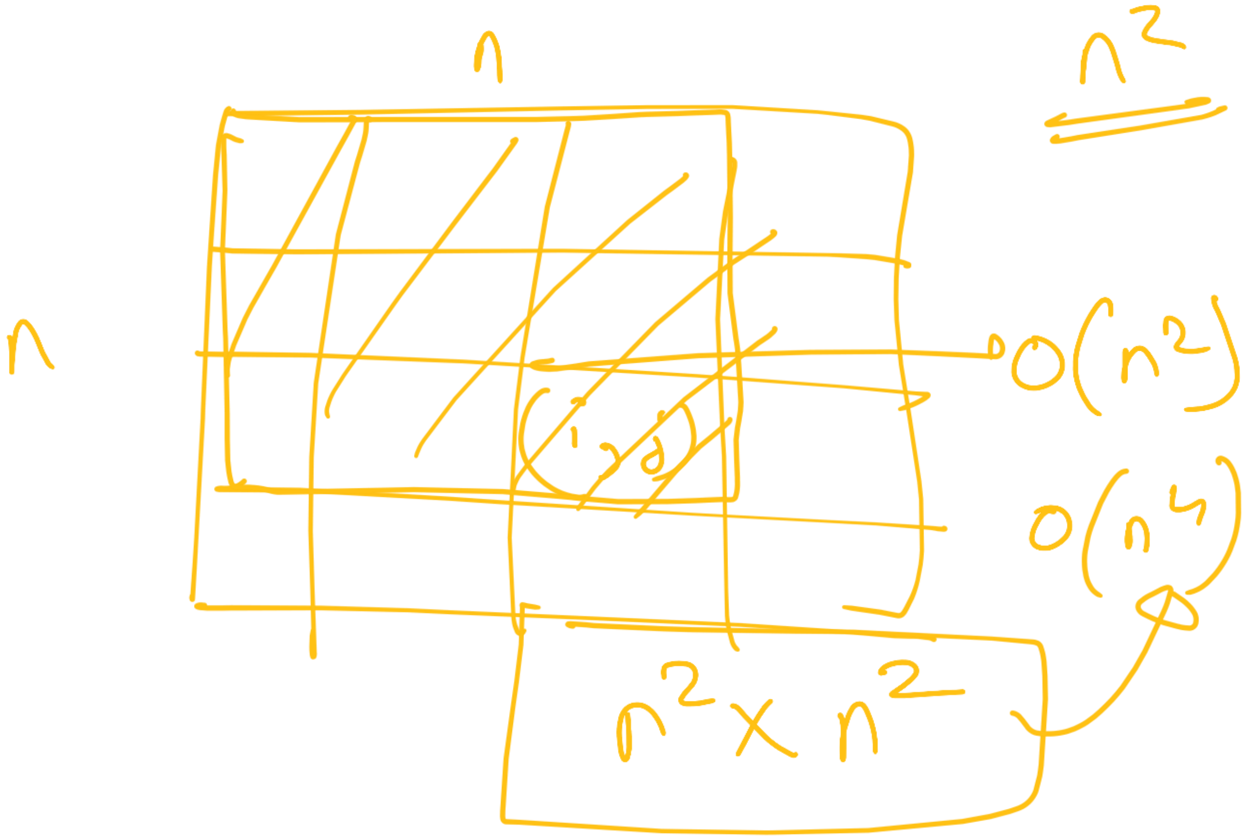
$(l_1, r_1) \quad (l_2, r_2)$

$$ans = P[l_2][r_2] - P[l_1 - 1][r_2] - P[l_2][r_1 - 1] + P[l_1 - 1][r_1 - 1]$$

Code to answer queries in O(1)

```
int query(int r1, int c1, int r2, int c2, vector<vector<int>>& P){
    // (r1, c1) = upper diagnol end, (r2, c2) = lower diagonal end
    int ans = P[r2][c2];
    if(r1 > 0)
        ans -= P[r1 - 1][c2];
    if(c1 > 0)
        ans -= P[r2][c1 - 1];
    if(r1 > 0 && c1 > 0)
        ans += P[r1 - 1][c1 - 1];
    return ans;
}
```





n
n
n²
(i,j)
O(n²)
O(n⁴)
n² x n²

Problem Forest Queries

Bonus: [Think about it for HW]

Given a 2D grid of N * M dimension filled with positive numbers, answer queries of the following form:

- Given coordinates of upper diagonal (r1, c1) and lower diagonal (r2, c2), find out the XOR of all elements in the rectangle.

Solution

Answer for a query with (r1, c1, r2, c2)

- Include all the values from (0, 0) to (r2, c2)
- Remove all values from (0, 0) to (r1 - 1, c2)
- Remove all values from (0, 0) to (r2, c1 - 1)
- Include values deleted twice (0, 0) to (r1 - 1, c1 - 1)

Xor[r2][c2] ^ Xor[r1 - 1][c2] ^ Xor[r2][c1 - 1] ^ Xor[r1 - 1][c1 - 1]