Today's Agenda:-

1. Understanding Prefix Sum
2. Sum of even indexed elements
3. Special Index.

$\boxed{\text{Starting} \quad 7:05 \text{ AM}}$

1. Given N elements & Q queries. For each query, calculate sum of all elements from L to R.

Ex:-

	0	1	2	3	4	5	6	7	8	9
Array =	-3	6	2	4	5	2	8	-9	3	1

Array = [-3, 6, 2, 4, 5, 2, 8, -9, 3, 1]

Queries

L	R	Solution
4	8	9.
3	7	10.
1	3	12
0	4	14
7	7	-9.

Q x 2.

Queries

	0	1
0	4	8
1	3	7
2	1	3
3	0	4
4	7	7

i

L = 4.
R = 8

sum = 0;

Brute Force Approach

```
void    printSum( int[] arr,   int[][] queries){
                              int j = 0;
Q →  for(int i = 0; i < queries.size(); i++){
          L = Queries[i][0];
          R = Queries[i][1];

          sum = 0;
N →       for( j = L; j <= R; j++){
               sum += arr[j];
          }
          print(sum);
     }
```

TC → $O(Q * N)$
SC → $O(1)$.

Scoreboard

Overs	1	2	3	4	5	6	7	8	9	10
Score	2	8	14	29	31	49	65	79	88	97

Quiz 1

Runs in 7^{th} over

score[7] − score[6] = 65 − 49

= 16.

Quiz 2

Runs from 6^{th} to 10^{th}

97 − 31 = 66.

Score[10] − Score[6−1]

Quiz 3

Runs in 10^{th} over.

Score[10] − Score[9] = 97 − 88

= 9.

Quiz 4

Runs from over 3rd to 6th

score[6] − score[2] = 49 − 8
= 41.

Quiz 5

Runs from 4th to 9th

score[9] − score[4−1]
= 88 − 14
= 74.

Observation for optimised solution

→ In a scoreboard, queries can be answered in constant time since we have a cumulative score.

→ In a similar manner, if we have a cumulative array for the above problem, we should be able to answer queries in constant time.

→ We need to store cumulative array or Prefix Sum array of the given input.

How to create prefix sum array?

$$
\begin{array}{c} \\ \text{Array} = \end{array}
\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ [2 & 5 & -1 & 7 & 1] \end{array}
$$

Definition:-

pF(i) → Sum of all element from index 0 to i.

$$
\begin{array}{c} \\ arr \rightarrow \\ pF[\,] \rightarrow \end{array}
\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ [2 & 5 & -1 & 7 & 1] \\ [2 & 7 & 6 & 13 & 14.] \end{array}
$$

Quiz 6:-

$$
\begin{array}{c} \\ arr :- \end{array}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ [10, & 32, & 6, & 12, & 20, & 1] \end{array}
$$

pF → [10, 42, 48, 60, 80, 81].

Brute Force Code

```
pF[N];

for (i = 0; i < N; i++) {
    sum = 0;
    for (j = 0; j <= i; j++) {
        sum += a[j];
    }
    pF[i] = sum;
}
```

Observation

pF[0] = A[0];
pF[1] = A[0] + A[1]; = pF[0] + A[1];
pF[2] = A[0] + A[1] + A[2] = pF[1] + A[2];
pF[3] = A[0] + A[1] + A[2] + A[3];
= pF[2] + A[3];

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

Optimised code:-

```
pF[N];
pF[0] = arr[0];
for (i=1; i<N; i++) {
    pF[i] = pF[i-1] + A[i];
}
```

T.C → O(N).
S.C. → O(N).

How to answer queries?

	0	1	2	3	4	5	6	7	8	9
Array =	-3	6	2	4	5	2	8	-9	3	1
pF =	-3	3	5	9	14	16	24	15	18	19

Queries

L	R	Solution
4	8	pF[8] − pF[4−1] = 18 − 9 = 9.
3	7	pF[7] − pF[3−1] = 15−5 = 10.
1	3	pF[3] − pF[1−1] = 9−(−3) = 12.
0	4	pF[4] = 14.
7	7	pF[7] − pF[7−1] = 15 −24 = −9.

Generalised Equation

$sum[L, R] = pF[R] - pF[L-1]$

\# if $L == 0$;

$= pF[R]$;

$0 \le L \le R \le N$

Complete Code

```
void rangeSum( int[] arr, int[][] queries){
        pf[N];          // Calculate prefix sum.
N       pF[0] = arr[0];
        for ( i=1; i<N; i++) {
            pF[i] = pF[i-1] + A[i];
        }

    for ( i=0; i< Queries.size() ; i++){
            L = Queries [i][0];
            R = Queries [i][1];
Q.
        if ( L==0) {
            print ( pF[R]; }
        else {
            print ( pF[R] - pf[L-1]);
        }
}
```

T.C → O(N+Q)

SC → O(N)

Q:- Given an array of size N and Q queries with start (s) & end (e) indices. For every query, return the sum of all even indexed elements from s to e.

Example

$$
\begin{array}{ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
A[] = & \{2, & 3, & 1, & 6, & 4, & 5\}
\end{array}
$$

Queries

S	E	Solution
1	3	1
2	5	5
0	4	7
3	3	0

Brute Force.

$O(Q * N)$

Observation & Optimisation

$$\begin{array}{cccccccc} & & 0 & 1 & 2 & 3 & 4 & 5 \\ arr[\,] & \rightarrow & \{2, & 3, & 1, & 6, & 4, & 5\} \end{array}$$

$$pF_E[\,] \rightarrow \{2, 2, 3, 3, 7, 7\}.$$

$$pF_E[i] = pfe[i-1] + A[i];$$

$A[i] = 0$ if i is odd;

Queries:

S	E	Solution
1	3	$pFe[3] - pF(1-1) = 1$.
2	5	$pFe[5] - pFe[2-1] = 5$
0	4	$pFe[4] = 7$.
3	3	$pFe[3] - pFe[3-1] = 0$

$$Sum[L, R] = pF_E[R] - pF_E[L-1]$$

if $(L == 0)$

$$= pFe[R];$$

Quiz 7

$$\begin{array}{cccccccc} & & 0 & 1 & 2 & 3 & 4 \\ arr[\,] & \rightarrow & [2, & 4, & 3, & 1, & 5] \\ pFe[\,] & \rightarrow & [2, & 2, & 5, & 5, & 10] \end{array}$$

if (i is odd)

$pF_e[i] = pF_e[i-1]$

else

$pF_e[i] = pF_e[i-1] + A[i]$

Pseudocode

```
void    rangeSum(int A[], int Queries[][]){

        int  pfE[N];

   N    pfE[0] = Arr[0];
        for (i=1; i<N; i++){
            if (i % 2 == 0){
            pfE[i] = pfE[i-1] + A[i];
          }
            else
            pfE[i] = pfE[i-1];
        }

   Q.   for (i=0;  i< Queries.size(); i++){

              L = Queries[i][0];
              R = Queries[i][1];

              if (L==0) {
                 print( pfE[R]; }

              else {
               print( pfE[R] - pfE[L-1]);
              }
        }
```

T.C → O(N+Q)

SC → O(N)

8:45

Extension

Sum of all odd indexed elements

```
pf0[0] = 0;

for (i=1; i<N; i++) {
    if (i % 2 == 1) {
        pf0[i] = pf0[i-1] + A[i];
    }
    else
        pf0[i] = pf0[i-1];
}
```

Problem :- Special Index

Given an array of size N, count the number of special index in the array.

Special indices are those after removing which, sum of all EVEN indexed elements is equal to the sum of all ODD indexed elem.

Ex:-

	0	1	2	3	4	5
A[] =	4	3	2	7	6	-2

i	Array 0	1	2	3	4	S_e	S_o
0	3	2	7	6	-2	8	8
1	4	2	7	6	-2	9	8
2	4	3	7	6	-2	9	9
3	4	3	2	6	-2	4	9
4	4	3	2	7	-2	4	10
5	4	3	2	7	6	12	10

Ans = 2.

Quiz 8

Sum of odd indices after removing index 2.

	0	1	2	3	4
arr[] =	4	1	3	7	10

arr → { 4, 1, 7, 10 }.

= 11.

Quiz 9 Remove index 3.
Sum of odd indexed elements

	0	1	2	3	4	5	6	7	8	9
arr[] →	2	3	1	4	0	-1	2	-2	10	8

arr[] → {2, 3, 1, 0, -1, 2, -2, 10, 8}.

= 15.

Sum of even indexed element after removing index 3.

= Sum of even indexed elements from 0 to 2
+
Sum of odd indexed element from 4 to 9.

$$S_E[i] = S_E[0 \text{ to } i-1] + S_O[i+1 \text{ to } N-1];$$

Quiz 10

Sum of EVEN indexed elements after removing index 3.

	0	1	2	3	4	5	6	7	8	9
arr[] →	2	3	1	4	0	-1	2	-2	10	8

arr[] → {2, 3, 1, 0, -1, 2, -2, 10, 8}.

= 8.

Observation

Sum of odd indexed element after removing index 3.

= Sum of odd indexed elements from 0 to 2
+
Sum of even indexed element from 4 to 9.

$S_O[i] = S_O[0 \text{ to } i-1] + S_E[i+1 \text{ to } N-1]$.

Approach.

1. Create Prefix Sum arrays for both odd indices & even indices.

2. Iterate from 0 to N-1; Check whether S_O is equal to S_E or not.

3. If equal, increment count.

Pseudocode

```
int countSI( int arr[] ) {

    int PS_E [N];
    int PS_O [N];
 // TODO - Populate the above 2
                       arrays.

    int count = 0;
  for ( i = 0; i < N; i++ ) {
        int Se, So;                    T.C -> O(N)
                                       S.C -> O(N)
     if (i == 0) {
          S_E = PS_O [N-1] - PS_O[0];
          S_O = PS_E [N-1] - PS_E [0];
     } else {
          S_E = PS_E[i-1]
                + PS_O [N-1] - PS_O[i];

          S_O = PS_O [i-1]
                + PS_E [N-1] - PS_E [i];
     }
     if (S_E == S_O) {
             count++;
     }
  }
          return count;
```

Next class

In next session, we'll learn 2 things -

1. Carry Forwards Technique

- In the context of Data Structures and Algorithms (DSA), the carry-forward technique is like a magic wand that helps us utilize the previously calculated results.
- It's a clever way of solving problems so that we don't have to recalculate the same results repeatedly, thus optimizing the Time Complexity.

2. Basics of Subarrays

- Subarrays are contiguous segments or slices of an array. They play a crucial role in various aspects of computer science, data analysis, and algorithm design.
- Dynamic Programming: Many dynamic programming algorithms use subarrays to build solutions incrementally, leading to efficient solutions for complex problems.
- Efficient Subarray Operations: Many array-related operations like sorting, searching, and updating can be optimized using techniques that rely on subarrays, such as divide-and-conquer algorithms.

Overall, subarrays are a powerful concept that simplifies algorithmic problem-solving and data manipulation.

Doubts :-