Introduction

Square root decomposition or Sqrt decomposition is a method or a data structure that splits up a structure into smaller chunks which enables faster processing of data. In other words, this technique helps us to perform some common operations like answering queries for finding the **sum/maximal/minimal** of elements in a given **range** efficiently.

Let us try to understand the technique through a problem statement.

Problem Statement

Problem Statement: Given an array **Arr** of size **N** consisting of integers and **Q** queries where a query can be of two types:
Type 1: Change the value of array at index **idx** to **val** i.e A[idx] = val.
Type 2: Find the sum of elements in the array from index **L** to **R** (R >= L and 0-Based Indexed).

As we know, the above problem can be solved using various data structures like segment trees, fenwick trees etc. but we will be solving the above problem using sqrt decomposition.

Let us discuss the naive approach to solve the above problem, followed by sqrt decomposition to process queries efficiently.

Naive Approach
The naive approach handles answering the queries in the following way:

1. Type 1 queries can be performed by simply updating the value of the array at the given index **idx** to **val** which is an **O(1)** operation that is fast enough.
2. Type 2 queries can be answered by simply iterating over the array **Arr** from the **L** to **R** and finding the sum which will be a **O(N)** operation in the worst case for each such query, hence answering such queries will take **O(x*N)** time for **x** such queries which is a costly operation!!

Pseudocode:

```
/*
        The function takes an input array 'Arr' and the range indices 'L' and 'R' and
        returns the sum from L to R.
*/

function getSum(Arr, L, R)

        //  Initializing sum to 0.
        sum = 0

        //  Finding sum from L to R
```

```
        for idx = L to R
                sum = sum + Arr[idx]

        return sum

/*
        The function takes input array 'Arr' and index 'idx' and value 'val' and updates
        the value of the array at the given index.
*/
function update(Arr, idx, val)

        //  Updating the value of array at index idx to val.
        Arr[idx] = val
        return

/*
        The function takes input array 'Arr', the size of the array 'N', and the number of
        queries 'Q'
*/
function solve(Arr, N, Q)

        //  Iterating over the number of queries Q
        for idx = 1 to Q
                //  Taking input of the "type" of query
                input(type)

                //  Taking input index idx and value val
                if type equals 1
                        input(idx,val)
                        update(Arr, idx, val)

                //  Taking input left and right indices L and R
                else
                        input(L, R)
                        sum = getSum(Arr, L, R)
                        print(sum)
```

Time Complexity: O(N*Q), where it takes O(N) time for answering each query in the worst case.

Since this approach is not quite efficient, let us use sqrt decomposition to solve this problem which brings down the time complexity for answering each query of Type 2 from O(N) to **O(sqrt(N))**, where **N** is the size of the array.