GSS4 - Can you answer these queries IV:

Problem: You are given a sequence A of N(N <= 100,000) positive integers. There sum will be less than 10^{18}. On this sequence you have to apply M (M <= 100,000) operations:

(A) For given x, y, for each elements between the x-th and the y-th ones (inclusively, counting from 1), modify it to its positive square root (rounded down to the nearest integer).

(B) For given x, y, query the sum of all the elements between the x-th and the y-th ones (inclusively, counting from 1) in the sequence.

Solution: A number within 1e18 will become 1 after a maximum of 6 square root operations. Establish a line segment tree. In each update operation, for an interval if the maximum value of the interval is 1, do not need to be updated (or the interval sum and equal to the interval length, we can directly ignore it) because if all the numbers of an interval is 1, then doing square root operations does not change the number of intervals, and the interval sum. For the interval that needs to be squared root, we update to the leaf node, because each leaf node is updated up to 6 times, so this is acceptable, time complexity O(nlogn).

Codeforces – 920F - SUM and REPLACE:

Let $D(x)$ be the number of positive divisors of a positive integer x. For example, $D(2) = 2$ (2 is divisible by 1 and 2), $D(6) = 4$ (6 is divisible by 1, 2, 3 and 6). You are given an array a ($a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^6$)) of n ($1 \le n \le 3 \cdot 10^5$) integers. You have to process two types of m ($1 \le n \le 3 \cdot 10^5$) queries:

1. REPLACE l r — for every $i \in [l, r]$ replace a_i with $D(a_i)$;
2. SUM l r — calculate $\sum_{i=l}^{r} a_i$

Solution: At first let's notice that this function converges very quickly, for values up to 10^6 it's at most 6 steps. Now we should learn how to skip updates on the numbers 1 and 2.

The function values can be calculated from the factorization of numbers in $O(MAXN \log MAXN)$ with Eratosthenes sieve.

Let's write two segment trees — one will store maximum value on segment, the other will store the sum. When updating some segment, check if its maximum is greater than 2. Updates are done in the manner one can usually write build function, you go down to the node corresponding to the segment of length 1 and update the value directly.

Overall complexity: $O(q \log n)$ as we access any segment no more than 6 times.

```cpp
// GSS4 - Can you answer these queries IV
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define INF 10000000000000000LL
#define MAXN 100005
#define lf nd<<1
#define rg (nd<<1)+1
#define m (int)((b+e)>>1)
#define N tree[nd]
#define L tree[lf]
#define R tree[rg]
struct data{
    ll sum,maxx;
}tree[4*MAXN];
ll a[MAXN];
void build(int nd,int b,int e){
    if(b==e){ N.sum = N.maxx = a[b];return; }
    build(lf,b,m);
    build(rg,m+1,e);
    N.sum = L.sum + R.sum;
    N.maxx = max(L.maxx, R.maxx);
}
void update(int nd,int b,int e,int x,int y){
    if(N.maxx==1)return;
    if(b==e){ N.sum = N.maxx = a[b] = (ll)sqrt(a[b]); return; }
    if(m>=y)update(lf,b,m,x,y);
    else if(x>m) update(rg,m+1,e,x,y);
    else{
        update(lf,b,m,x,m);
        update(rg,m+1,e,m+1,y);
    }
    N.sum = L.sum + R.sum;
    N.maxx = max(L.maxx, R.maxx);
}
ll query(int nd,int b,int e,int x,int y){
    if(b==x && e==y)return N.sum;
    if(m>=y)return query(lf,b,m,x,y);
    else if(x>m) return query(rg,m+1,e,x,y);
    else return query(lf,b,m,x,m) + query(rg,m+1,e,m+1,y);
}
int main(){
    int n,ks=0;
    while(scanf("%d",&n)==1){
        printf("Case #%d:\n",++ks);
        for(int i=1; i<=n; i++)scanf("%lld",&a[i]);

        build(1,1,n);

        int q; scanf("%d",&q);
        while(q--){
            int t,x,y; scanf("%d%d%d",&t,&x,&y);
            if(x>y)swap(x,y);
            if(t==0){
                update(1,1,n,x,y);
            } else{
                ll ans = query(1,1,n,x,y);
                printf("%lld\n",ans);
            }
        }
        printf("\n");
    }
    return 0;
}
```