SEGSQRSS - Sum of Squares with Segment Tree

#tree

Segment trees are extremely useful. In particular "Lazy Propagation" (i.e. see here, for example) allows one to compute sums over a range in O(lg(n)), and update ranges in O(lg(n)) as well. In this problem you will compute something much harder:

The sum of squares over a range with range updates of 2 types:

1) increment in a range

2) set all numbers the same in a range.

Input

There will be **T** (**T** <= 25) test cases in the input file. First line of the input contains two positive integers, **N** (**N** <= 100,000) and **Q** (**Q** <= 100,000). The next line contains **N**integers, each at most 1000. Each of the next **Q** lines starts with a number, which indicates the type of operation:

2 **st nd** -- return the sum of the squares of the numbers with indices in [**st**, **nd**] {i.e., from **st** to **nd** inclusive} (1 <= **st** <= **nd** <= **N**).

1 **st nd x** -- add "x" to all numbers with indices in [**st**, **nd**] (1 <= **st** <= **nd** <= **N**, and -1,000 <= **x** <= 1,000).

0 **st nd x** -- set all numbers with indices in [**st**, **nd**] to "**x**" (1 <= **st** <= **nd** <= **N**, and -1,000 <= **x** <= 1,000).

Output

For each test case output the "`Case <caseno>:`" in the first line and from the second line output the sum of squares for each operation of type 2. Intermediate overflow will not occur with proper use of 64-bit signed integer.

Example

```
Input:
2
4 5
1 2 3 4
2 1 4
0 3 4 1
2 1 4
1 3 4 1
2 1 4
1 1
1
2 1 1

Output:
Case 1:
30
7
13
Case 2:
1
```

Solution Idea:

This problem is a nice example of segment tree with lazy propagation. All you need to solve this kind of problem is to do some experiment with the formula and convert them into a suitable form. From which we can drive the segment tree solution.

In this problem given an integer array A=[a1,a2,a3,a4…]. You need to perform 3 kinds of operation on it.

type 0: Set all the element in the interval l to r to x.
type 1: Increase all the element in the interval l to r by x.
type 2: Print the value of square sum of the interval l to r. For example let l=1 and r=3. then you need to print a1^2 + a2^2 + a3^2.

At first think we have a pre-calculated square sum array. Then if we perform type 1 operation which is increase it by x then what scenario will happen?

Our sum now become (a1+x)^2 + (a2+x)^2 + (a3+x)^2. Let expand this equation-

(a1+x)^2 + (a2+x)^2 + (a3+x)^2
= a1^2+ 2*a1*x + x^2 + a2^2+ 2*a2*x + x^2 + a3^2+ 2*a3*x + x^2
= (a1^2 + a2^2 + a3^2) + 3*x^2 + 2*x*(a1 + a2 + a3)

for a range l to r the generalized equation is –

(a_l^2 + a_l+1^2 +….+ a_r^2) + (r-l+1)*x^2 + 2*x*(a_l + a_l+1 +….+ a_r)

We can store the information about sum, square_sum, type_0 lazy and type_1 lazy of an interval in each segment tree node. And perform type 1 operation then we can calculate then sum value for a node over a range by multiplication and square sum value by above equation.

For every type 0 operation we can simply put the value of sum and square sum by using some basic calculation and arithmetic multiplication operation.

And for every type 2 operation we just need to get the sum of square sum value over a range l to r.

So we can perform each of 3 operation using a segment tree with lazy propagation.

```cpp
/// https://www.spoj.com/problems/SEGSQRSS/
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define mx 100005
#define sqr(x)  (x)*(x)
struct data{
    ll sum,sqrsum,lazy,upd;
};
data tree[3*mx];
ll ara[mx];

void push_down(int n, int b, int e){
    int l=n*2,r=l+1,mid=(b+e)/2;

    if(tree[n].upd){
        tree[l].lazy=0;
        tree[r].lazy=0;
        tree[l].sum=(mid-b+1)*tree[n].upd;
        tree[l].sqrsum=(mid-b+1)*sqr(tree[n].upd);
        tree[r].sum=(e-mid)*tree[n].upd;
        tree[r].sqrsum=(e-mid)*sqr(tree[n].upd);
        tree[l].upd=tree[n].upd;
        tree[r].upd=tree[n].upd;
        tree[n].upd=0;
    }
    if(tree[n].lazy){
        tree[l].sqrsum+=(tree[l].sum*(2*tree[n].lazy))+(mid-b+1)*sqr(tree[n].lazy);
        tree[r].sqrsum+=(tree[r].sum*(2*tree[n].lazy))+(e-mid)*sqr(tree[n].lazy);
        tree[l].sum+=(mid-b+1)*tree[n].lazy;
        tree[r].sum+=(e-mid)*tree[n].lazy;
        tree[l].lazy+=tree[n].lazy;
        tree[r].lazy+=tree[n].lazy;
        tree[n].lazy=0;
    }
}

void init(int n, int b, int e){
    if(b==e){
        tree[n].sum=ara[b];
        tree[n].sqrsum=sqr(ara[b]);
        tree[n].lazy=0;
        tree[n].upd=0;
        return;
    }

    int l=n*2,r=l+1,mid=(b+e)/2;

    init(l,b,mid);
    init(r,mid+1,e);

    tree[n].sum=tree[l].sum+tree[r].sum;
    tree[n].sqrsum=tree[l].sqrsum+tree[r].sqrsum;
    tree[n].lazy=0;
    tree[n].upd=0;
}

void update(int n, int b, int e, int x, int y, ll val, int type){
    if(b>y || e<x) return;
    if(b>=x && e<=y){
        if(type==0){
            tree[n].sum=(e-b+1)*val;
            tree[n].sqrsum=(e-b+1)*sqr(val);
            tree[n].lazy=0;
            tree[n].upd=val;
        }else{
            tree[n].sqrsum+=(tree[n].sum*2*val)+((e-b+1)*sqr(val));
            tree[n].sum+=(e-b+1)*val;
            tree[n].lazy+=val;
        }
        return;
    }
```

```
73
74      push_down(n,b,e);
75
76      int l=n*2,r=l+1,mid=(b+e)/2;
77
78      update(l,b,mid,x,y,val,type);
79      update(r,mid+1,e,x,y,val,type);
80
81      tree[n].sum=tree[l].sum+tree[r].sum;
82      tree[n].sqrsum=tree[l].sqrsum+tree[r].sqrsum;
83  }
84
85  ll query(int n, int b, int e, int x, int y){
86      if(b>y || e<x) return 0;
87      if(b>=x && e<=y) return tree[n].sqrsum;
88
89      push_down(n,b,e);
90
91      int l=n*2,r=l+1,mid=(b+e)/2;
92
93      ll p=query(l,b,mid,x,y);
94      ll q=query(r,mid+1,e,x,y);
95      return p+q;
96  }
97  int main(){
98      int tt;scanf("%d",&tt);
99
100     for(int ks=1; ks<=tt; ks++){
101
102         int n,q; scanf("%d%d",&n,&q);
103         for(int i=1; i<=n; i++) scanf("%lld",&ara[i]);
104
105         init(1,1,n);
106
107         printf("Case %d:\n",ks);
108         while(q--){
109             int typ,x,y; scanf("%d%d%d",&typ,&x,&y);
110
111             if(typ==2){
112                 ll ans=query(1,1,n,x,y);
113                 printf("%lld\n",ans);
114             }else{
115                 int v; scanf("%d",&v);
116
117                 if(typ==0) update(1,1,n,x,y,v,0);
118                 else update(1,1,n,x,y,v,1);
119             }
120         }
121     }
122
123     return 0;
124 }
```