Another Update-Query Problem

Limits: 1s, 512 MB

You will be given an array **A** of length **N**. On that array you will have to do **Q**operations. Operations are of two types.

Operation-1: U(l, r, x): Add x to A_i, where i is in range [l, r]

Operation-2: Q(l, r, d): Print the output of this series modulo 1000000007:

$1A_l + (1+d)A_{l+1} + (1+2d)A_{l+2} + (1+3d)A_{l+3} + \dots + (1+(r-l)d)A_r$

Input

Input starts with an integer **T (1≤T≤10)**, denoting the number of test cases.
First line of each case has two integers, **N (1≤N≤100000)** and **Q (1≤Q≤100000)**.
Second line has N integers **A_i (1≤A_i≤1000000000)**, indicating the values of the array.
Following **Q** lines each has four integers. First of those four integers are **C (1≤C≤2)**.
If **C=1**, then it requests an operation of type-1. The other three integers will be **L (1≤L≤N)**, **R (1≤R≤N, L≤R)**, **X (1≤X≤1000000000)**. And you will have to do operation **U(L, R, X)**.
If **C=2**, then it requests an operation of type-2. The other three integers will be **L (1≤L≤N)**, **R (1≤R≤N, L≤R)**, **D (1≤D≤1000000000)**. And you will have to print the value of **Q(L, R, D)** modulo **1000000007**.

Easy Subtask:
1≤N≤2000
1≤Q≤2000
C=2 for all cases. That means there is no request for operation-1.

Medium Subtask:
1≤N≤100000
1≤Q≤100000
C=2 for all cases. That means there is no request for operation-1.

Hard Subtask:
Full specification

Output

For each case, first print the case number, starting from 1, in a separate line. For each request of operation-2, print the result in a separate line.

Samples

Input	Output

2
5 4
2 8 19 9 1
2 2 5 3
2 2 4 6
2 2 5 3
2 2 5 8
3 3
6 7 1
2 2 2 7
1 1 3 8
2 1 2 6
	Case 1:
	157
	258
	157
	357
	Case 2:
	7
	119

Notes

The main idea behind the solution of this problem is simplifying the equation. For a query operation we need the answer of this equation-
1A_l + (1+d)A_l+1 + (1+2d)A_l+2 + (1+3d)A_l+3 + … + (1+(r-l)d)A_r
Now let the query range l r is 1 3. and the given array is A=[a1,a2,a3,a4,…]

So our quation will be-
1*a1 + (1+d)*a2 + (1+2d)*a3 + (1+3d)*a4
= a1 + a2 + d*a2 + a3 + 2d*a3 + a4 + 3d*a4
= (a1+a2+a3+a4) + d*(a2+ 2*a3 + 3*a4)

Now We can see that first part of this equation is only sum query which can be done using a segment tree and for 2nd part we can use a trick. We can pree calculate the sequence like this-
1*a1 + 2*a2 + 3*a3 + 4*a4 + 5*a5 + 6*a6 + 7*a7 + ……
When we need the 2nd part of the equation we can use this equation. We can get the 2nd part of the equation from this equation by this way-
(2*a2 + 3*a3 + 4*a4)-(a2+a3+a4) = (a2+ 2*a3 + 3*a4).

Now the above equation is also only a sum equation. We can apply any range sum query or range sum update on this equation and the the first sum equation.

So if we store this two equation in two segment tree then we can perform range update or query on the segment tree and get any range sum query from the segment tree and using the equation we can answer each query of the problem correctly.

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define mx 100005
ll MOD = 1000000007;
ll a[mx],lazy[4*mx],tree[4*mx],sum[4*mx];
void setValue(ll nd,ll b,ll e,ll p){
    ll uu = ((e-b+1)*p)%MOD;
    tree[nd] = (tree[nd]+uu)%MOD;
    ll ee = (e*(e+1LL))/2LL;
    ll bb = ((b-1LL)*b)/2LL;
    ll xx = (ee-bb+MOD)%MOD;
    ll vv = (xx*p)%MOD;
    sum[nd] = (sum[nd]+vv)%MOD;
}
void pushDown(ll nd,ll b,ll e){
    ll m = (b+e)/2;
    setValue(2*nd,b,m,lazy[nd]);
    setValue(2*nd+1,m+1,e,lazy[nd]);

    lazy[2*nd] += lazy[nd];
    lazy[2*nd+1] += lazy[nd];
    lazy[2*nd] %= MOD;
    lazy[2*nd+1] %= MOD;
    lazy[nd]=0;
}
void init(ll nd,ll b,ll e){
    if(b==e){
        tree[nd] = a[b]%MOD;
        sum[nd]  = (b*a[b])%MOD;
        lazy[nd] = 0;
        return;
    }

    ll m = (b+e)/2;
    init(2*nd,b,m);
    init(2*nd+1,m+1,e);
    tree[nd] = (tree[2*nd]+tree[2*nd+1])%MOD;
    sum[nd]  = (sum[2*nd]+sum[2*nd+1])%MOD;
    lazy[nd]=0;
}
void update(ll nd,ll b,ll e,ll l,ll r,ll p){
    if(b>r||e<l)return;
    if(b>=l&&e<=r){
        setValue(nd,b,e,p);
        lazy[nd]  = (lazy[nd]+p)%MOD;
        return;
    }

    if(lazy[nd]!=0){
        pushDown(nd,b,e);
    }

    ll m = (b+e)/2;
    update(2*nd,b,m,l,r,p);
    update(2*nd+1,m+1,e,l,r,p);

    tree[nd] = (tree[2*nd]+tree[2*nd+1])%MOD;
    sum[nd]  = (sum[2*nd]+sum[2*nd+1])%MOD;
}
```

```cpp
ll query1(ll nd,ll b,ll e,ll l,ll r){
    if(l>r)return 0;
    if(b>r||e<l)return 0;
    if(b>=l&&e<=r){
        return (tree[nd])%MOD;
    }

    if(lazy[nd]!=0){
        pushDown(nd,b,e);
    }

    ll m = (b+e)/2;
    ll u1 = query1(2*nd,b,m,l,r)%MOD;
    ll u2 = query1(2*nd+1,m+1,e,l,r)%MOD;
    return (u1+u2)%MOD;
}
ll query2(ll nd,ll b,ll e,ll l,ll r){
    if(l>r)return 0;
    if(b>r||e<l)return 0;
    if(b>=l&&e<=r){
        return (sum[nd])%MOD;
    }

    if(lazy[nd]!=0){
        pushDown(nd,b,e);
    }

    ll m = (b+e)/2;
    ll v1 = query2(2*nd,b,m,l,r)%MOD;
    ll v2 = query2(2*nd+1,m+1,e,l,r)%MOD;
    return (v1+v2)%MOD;
}
int main(){
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);

    ll tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++){
        ll n,q; cin>>n>>q;
        for(int i=1; i<=n; i++)cin>>a[i];

        init(1,1,n);

        cout<<"Case "<<ks<<":"<<endl;
        while(q--) {
            ll c,l,r,x; cin>>c>>l>>r>>x;
            if(c==1){
                update(1,1,n,l,r,x);
            }
            else{
                ll u = query1(1,1,n,l,r)%MOD;
                ll w = query1(1,1,n,l+1,r)%MOD;
                ll v = query2(1,1,n,l+1,r)%MOD;
                w = (l*w)%MOD;
                ll vw = (v-w+MOD)%MOD;
                vw = (vw*x)%MOD;
                ll ans = (u+vw)%MOD;
                cout<<ans<<endl;
            }
        }
    }
    return 0;
}
```