```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pii pair < int , int >
#define pb push_back
#define mp make_pair
#define mod 1000000009
template<class T>
class segmentTree{
public:
    segmentTree(){
        height = 1;
        left_most = 1<<height;
        right_most = (left_most<<1) - 1;
        tree = new T[right_most];
    }
    segmentTree(int s){
        size=s;
        height = ceil(log2(s));
        left_most = 1<<height;
        right_most = (left_most<<1) - 1;
        tree = new T[right_most+9];
    }
    void init(T * arr){
        build(arr);
    }
    void fill_ans(){
        initalize(1,left_most,right_most);
        for(int i = left_most ;i< left_most+size ; i++){
            for(int j=1;j<=26;j++)
                if(tree[i].arr[j]){
                    cout<<char(j+96);
                    break;
                }
        }
    }
    void Update(int pos , T val){
        point_update(1 , left_most , right_most , left_most+pos , val);
    }
    void Update(int l , int r , T val){
        range_update(1 , left_most , right_most , left_most+l , left_most+r , val);
    }
    T Query(int pos){
        return point_query(1 , left_most , right_most , left_most+pos);
    }
    T Query(int l ,int r){
        return range_query(1 , left_most , right_most , left_most+l , left_most+r);
    }
private:
    T *tree;
    int size , left_most , right_most , height;
    void build(T * arr){
        for(int i = 0 ; i < size ; i++)
            tree[left_most+i] = arr[i];
        initalize(1 , left_most , right_most);
    }
    void initalize(int root , int left_most , int right_most){
        if(left_most == right_most) return;
        int mid = (left_most + right_most)>>1 , l_child = (root<<1)  , r_child = (root<<1)+1;
        tree[root].split(tree[l_child] , tree[r_child]);
        initalize(l_child , left_most , mid);
        initalize(r_child , mid+1 , right_most);
        tree[root].merge(tree[l_child] , tree[r_child]);
    }
```

```
    void point_update(int root , int left_most , int right_most , int pos , T val){
        if(left_most == right_most && root == pos) { tree[root].update(val); return ;}
        int mid = (left_most + right_most)>>1 , l_child = root<<1 , r_child = (root<<1)+1;
        tree[root].split(tree[l_child] , tree[r_child]);
        if(pos <= mid) point_update(l_child , left_most , mid , pos , val);
        else point_update(r_child , mid+1 , right_most , pos , val);
        tree[root].merge(tree[l_child] , tree[r_child]);
    }
    void range_update(int root , int left_most , int right_most , int l , int r , T val){
        if(l <= left_most && r >= right_most){ tree[root].update(val);return;}
        int mid = (left_most + right_most)>>1 , l_child = root<<1 , r_child = (root<<1)+1;
        tree[root].split(tree[l_child] , tree[r_child]);
        if(l <= mid) range_update(l_child , left_most , mid, l , r , val);
        if(r > mid) range_update(r_child , mid+1 , right_most , l , r , val);
        tree[root].merge(tree[l_child] , tree[r_child]);
    }
    T range_query(int root , int left_most ,int right_most ,int l , int r){
        if( l <= left_most && r >= right_most )
            return tree[root];
        int mid = (left_most + right_most)>>1 , l_child = root<<1 , r_child = (root<<1)+1;
        tree[root].split(tree[l_child] , tree[r_child]);
        T l_node  , r_node , temp;
        if(l <= mid) l_node = range_query(l_child , left_most , mid , l , r );
        if(r > mid) r_node = range_query(r_child , mid+1 , right_most , l , r );
        tree[root].merge(tree[l_child] , tree[r_child]);
        temp.merge(l_node , r_node);
        return temp;
    }
    T point_query(int root , int left_most , int right_most , int pos){
        if(left_most == right_most && root == pos) return tree[root];
        int mid = (left_most + right_most)>>1 , l_child = root<<1 , r_child = (root<<1)+1;
        T temp;
        tree[root].split(tree[l_child] , tree[r_child]);
        if(pos <= mid) temp = point_query(l_child , left_most , mid , pos);
        else temp = point_query(r_child , mid+1 , right_most , pos);
        tree[root].merge(tree[l_child] , tree[r_child]);
        return temp;
    }
};
class node{
public:
    ll sum , sq_sum , lazy1 , lazy2;
    int child_count;
    void merge(node &a , node &b){
        sum = a.sum + b.sum;
        sq_sum = a.sq_sum + b.sq_sum;
        child_count = a.child_count + b.child_count;
        lazy1 = lazy2 = 0;
    }
    void split(node &a , node &b){
        if(lazy1){
            a.sq_sum += lazy1 * lazy1 * (ll)a.child_count + 2LL * lazy1 * a.sum;
            b.sq_sum += lazy1 * lazy1 * (ll)b.child_count + 2LL * lazy1 * b.sum;
            a.sum += lazy1 * a.child_count;
            b.sum += lazy1 * b.child_count;
            a.lazy1 += lazy1;
            b.lazy1 += lazy1;
            lazy1 = 0;
        }
        if(lazy2){
            a.sq_sum = a.child_count * lazy2 * lazy2;
            a.sum = a.child_count * lazy2;
            a.lazy2 += lazy2;
            b.sq_sum = b.child_count * lazy2 * lazy2;
            b.sum = b.child_count * lazy2;
```

```
            b.lazy2 += lazy2;
        }
    }
    void update(node &a){
        if(a.lazy1){
            sq_sum = sq_sum + a.lazy1 * a.lazy1 * child_count + 2LL * a.lazy1 * sum;
            sum += a.lazy1 * child_count;
            lazy1 += a.lazy1;
        }
        if(a.lazy2){
            sq_sum = child_count * a.lazy2 * a.lazy2;
            sum = child_count * a.lazy2;
            lazy2 += a.lazy2;
        }
    }
    node(){
        sum = sq_sum = lazy1 = lazy2 = 0;
        child_count = 0;
    }
    node(ll a , ll l1 , ll l2){
        sum = a;
        sq_sum = a*a;
        child_count = 1;
        lazy1 = l1;
        lazy2 = l2;
    }
};
node arr[100009];
int main(){
    int t;
    scanf("%d",&t);
    for(int test = 1 ; test <= t ; test++){
        int n , temp ,q;
        scanf("%d%d",&n,&q);
        segmentTree<node> s(n);

        for(int i =0;i<n;i++){
            scanf("%d",&temp);
            arr[i]=node(temp , 0 , 0);
        }
        s.init(arr);
        while(q--){
            int l,r,k,val;
            scanf("%d%d%d",&k,&l,&r);
            l-- , r--;
            if(k == 2){
                printf("%lld\n",s.Query(l,r).sq_sum);
            }
            else if(k==1){
                scanf("%d",&val);
                s.Update(l , r , node(0 , val , 0));
            } else{
                scanf("%d",&val);
                s.Update(l , r , node(0 , 0 , val));
            }
        }
    }
    return 0;
}
```