#include<bits/stdc++.h>

using namespace std;

using ll =long long;

template<class Info, class Merge = std::plus<Info>>

struct SegmentTree {

const int n;

const Merge merge;

vector<Info> info;

vector<int> last;

SegmentTree(int n) : n(n), merge(Merge()), info(4 << std::\_\_lg(n)) ,last(4<<std::\_\_lg(n)){}

SegmentTree(vector<Info> init) : SegmentTree(init.size()) {

function<void(int, int, int)> build = [&](int p, int l, int r) {

if (r ==l) {

info[p] = init[l];

return;

}

int mid =l+r>>1;

build(p<<1, l,mid);

build(p<<1|1, mid+1, r);

pull(p);

};

build(1, 0, n);

}

void pull(int p) {

info[p] = merge(info[p<<1], info[p<<1|1]);

}

void modify(int p, int l, int r, int x, const Info &v) {

if (r == l) {

info[p] = v;

return;

}

int mid = (l + r) / 2;

if (x <= mid) {

modify( p<<1,l, mid, x, v);

} else {

modify(p<<1|1, mid+1, r, x, v);

}

pull(p);

}

void pushdown(int p,int l,int r)

{

if(last[p]!=0)

{

int mid=l+r>>1;

info[p<<1].sum+=last[p]\*(mid-l+1);

info[p<<1|1].sum+=last[p]\*(r-mid);

last[p<<1]+=last[p];

last[p<<1|1]+=last[p];

last[p]=0;

}

}

void rangeupdate(int p,int l,int r,int x,int y,int k)

{

if (l >y || r < x) {

return ;

}

if(l>=x&&r<=y)

{

last[p]+=k;

info[p].sum+=k\*(r-l+1);

return ;

}

int mid=l+r>>1;

pushdown(p,l,r);

rangeupdate(p<<1, l, mid, x, y,k), rangeupdate(p<<1|1, mid+1, r, x, y, k);

pull(p);

}

void modify(int p, const Info &v) {

modify(1, 1, n, p, v);

}

Info rangeQuery(int p, int l, int r, int x, int y) {

if (l >y || r < x) {

return Info();

}

if (l >= x && r <= y) {

return info[p];

}

int m = (l + r) / 2;

pushdown(p,l,r);

return merge(rangeQuery(p<<1, l, m, x, y), rangeQuery(p<<1|1, m+1, r, x, y));

}

Info rangeQuery(int l, int r) {

return rangeQuery(1, 1, n, l, r);

}

};

struct Info {

ll sum=0;

};

Info operator+(Info a, Info b) {

a.sum=a.sum+b.sum;

return a;

}

int main() {

int N, Q;

cin >> N >> Q;

vector<int> A(N+1);

for (int i = 1; i <=N; i++) {

cin >> A[i];

}

SegmentTree<Info> seg(N);

for (int i = 1; i <= N; i++) {

seg.modify(i, Info{A[i]});

}

while (Q--) {

int type;

cin >> type;

if (type == 1) {

int l,r;

cin>>l>>r;

int k;

cin>>k;

seg.rangeupdate(1,1,N,l,r,k);

} else {

int l,r;

cin>>l>>r;

cout<<seg.rangeQuery(l,r).sum<<'\n';

}

}

return 0;

}