LazyTree.cpp

```cpp
//0 p q v - you have to add v to all numbers in the range of p to q (inclusive), where p and q are two indexes of the array.
//1 p q - output a line containing a single integer which is the sum of all the array elements between p and q (inclusive)

#include<bits/stdc++.h>

using namespace std;

long long *p;

struct SegmentTree{
    SegmentTree *L, *R;
    long long sum = 0;
    long long lazy = 0;
    int l, r;

    long long query2(int a, int b){
        if(a == l && b == r) return sum;
        if(b <= L->r) return L->query(a,b);
        if(a >= R->l) return R->query(a,b);
        return (L->query2(a,L->r) + R->query2(R->l, b));
    }

    void update(int a, int val){
        if(l == r){
            sum += val;
            return;
        }
        int mid = (l + r)/2;
        if(l <= a && a<= mid)
            L->update(a, val);
        else
            R->update(a, val);
        sum = L->sum + R->sum;
    }

    void updateRange2(int a, int b, long long val){
        if(b < l or a > r)
            return;
        if(l == r){
            sum += val;
            return;
        }
        L->updateRange2(a, b, val);
        R->updateRange2(a,b,val);
        sum = L->sum + R->sum;
    }

    void updateRange(int a, int b, long long val){
        if(lazy != 0){
            sum += (r-l+1)*lazy;
            if(l != r){
                R->lazy = lazy + R->lazy;
                L->lazy = lazy + L->lazy;
            }
            lazy = 0;
        }
        if(b < l or a > r)
            return;
        if(l >= a && r <= b){
            sum += (r-l+1)*val;
            if(l != r){
                R->lazy = val + R->lazy;
                L->lazy = val + L->lazy;
            }
            return;
        }
        L->updateRange(a, b, val);
        R->updateRange(a,b,val);
        sum = L->sum + R->sum;
    }

    long long query(int a, int b){
        if(b < l or a > r)
            return 0;
        if(lazy != 0){
            sum += (r-l+1)*lazy;
            if(l != r){
                R->lazy = lazy + R->lazy;
                L->lazy = lazy + L->lazy;
            }
            lazy = 0;
        }
        if(a == l && b == r) return sum;
```

```
        if(b <= L->r) return L->query(a,b);
        if(a >= R->l) return R->query(a,b);
        return (L->query(a,L->r) + R->query(R->l, b));
    }

    SegmentTree(int a, int b): l(a), r(b){
        if(a == b){
            sum = p[a];
            L = R = nullptr;
        }
        else{
            L = new SegmentTree ( a, (a+b)/2 );
            R = new SegmentTree ( (a+b)/2 + 1, b );
            sum = L->sum + R->sum;
        }
    }
};

int main(){
    long long T;
    cin >> T;
    while(T--){
        long long n, c;
        cin >> n >> c;
        long long l[n];
        memset(l,0,sizeof(l));
        p = l;
        SegmentTree *stree = new SegmentTree(0, n-1);
        while(c--){
            long long aux, p, q;
            cin >> aux >> p >> q;
            if(aux == 0){
                long long val;
                cin >> val;
                stree->updateRange(p-1, q-1, val);
            }
            else
                cout << stree->query(p-1, q-1) << endl;
        }
    }
}
```