**Segment with Maximum Sum**

**Problem**

You are given an array of size n. You will be given m point updates on the array. Queries will be to return the maximum sum of numbers on a segment before all operations and after each operation.

**Constraints**

*1 <= n, m <= 105*

*-109 <= a[i] <= 109*

**Approach**

In these kind of problems we require four variables for each segment of the segment tree

1. Sum - Stores the sum of the segment
2. Suff -Stores the maximum suffix sum of the segment
3. Pref - Stores the maximum prefix sum of the segment
4. Ans - Stores the ans that is the maximum sum of the segment

**Updation**

Left child - L, Right child - R. Then the parent P’s properties are

1. P.sum = L.sum + R.sum
2. P.suff = max(R.suff, R.sum + L.suff)
3. P.pref = max(L.pref, L.sum + R.pref)
4. P.ans = max(L.ans, R.ans, L.suff + R.pref)

**Code**

#include "bits/stdc++.h"

using namespace std;

#define int long long

const int N = 1e5+2, MOD = 1e9+7;

struct grp

{

int sum, pref, suff, ans;

};

grp tree[4\*N];

int a[N];

void build(int node, int st, int en)

{

if(st == en){

if(a[st]<=0){

tree[node].sum = a[st];

tree[node].pref = tree[node].suff = tree[node].ans = 0;

}

else{

tree[node].sum = tree[node].pref = tree[node].suff = tree[node].ans = a[st];

}

return;

}

int mid = (st + en)/2;

build(2\*node, st, mid);

build(2\*node+1, mid+1, en);

tree[node].sum = tree[2\*node].sum + tree[2\*node+1].sum;

tree[node].pref = max(tree[2\*node].pref, tree[2\*node].sum + tree[2\*node+1].pref);

tree[node].suff = max(tree[2\*node+1].suff, tree[2\*node+1].sum + tree[2\*node].suff);

tree[node].ans = max(tree[2\*node].suff+tree[2\*node+1].pref, max(tree[2\*node].ans, tree[2\*node+1].ans));

}

// pair<int,int> query(int node, int st, int en, int l, int r)

// {

// if(st>r || en<l)

// return {MOD, -1};

// if(l<=st && en<=r)

// return tree[node];

// int mid = (st+en)/2;

// pair<int,int> q1 = query(2\*node, st, mid, l, r);

// pair<int,int> q2 = query(2\*node+1, mid+1, en, l, r);

// pair<int,int> q;

// if(q1.first < q2.first){

// q.first = q1.first;

// q.second = q1.second;

// }

// else if(q2.first < q1.first){

// q.first = q2.first;

// q.second = q2.second;

// }

// else{

// q.first = q1.first;

// q.second = q1.second + q2.second;

// }

// return q;

// }

void update(int node, int st, int en, int idx, int val){

if(st == en){

a[st] = val;

if(a[st]<=0){

tree[node].sum = a[st];

tree[node].pref = tree[node].suff = tree[node].ans = 0;

}

else{

tree[node].sum = tree[node].pref = tree[node].suff = tree[node].ans = a[st];

}

return;

}

int mid = (st+en)/2;

if(idx <= mid){

update(2\*node, st, mid, idx, val);

}

else

{

update(2\*node+1, mid+1, en, idx, val);

}

tree[node].sum = tree[2\*node].sum + tree[2\*node+1].sum;

tree[node].pref = max(tree[2\*node].pref, tree[2\*node].sum + tree[2\*node+1].pref);

tree[node].suff = max(tree[2\*node+1].suff, tree[2\*node+1].sum + tree[2\*node].suff);

tree[node].ans = max(tree[2\*node].suff+tree[2\*node+1].pref, max(tree[2\*node].ans, tree[2\*node+1].ans));

}

signed main()

{

int n,m;

cin >> n >> m;

for(int i=0; i<n; i++){

cin >> a[i];

}

build(1,0,n-1);

cout << tree[1].ans << endl;

while(m--){

int idx,val;

cin >> idx >> val;

update(1,0,n-1,idx,val);

cout << tree[1].ans << endl;

}

return 0;

}