SPOJ-1557::Can you answer these queries II

This is one of the difficult problems that i have solved. So i've decided to write an article about how to solve this problem.

Logically the problem statement is exactly like below.

Problem: Given a array of numbers a[1...n] (where duplicates are allowed), and a query range [x,y].
query(x,y) should return the sub-sequence sum, whose sum is maximum in the range [x,y] by satisfying uniqueness criteria.

Assume that a[x...y] is a sub-sequence having unique elements(i.e. with out duplicates).
Lets analyze the query asked for this range.
Query for range {x...y} can be expressed as below, assuming that all the queries {query(i,j) where j<y} were answered already.

```
query(x,y) =  max {
                 b(x,y),
                b(x+1,y),
                b(x+2,y),
                .
                ..
                ...
                b(y,y)
            }
```

where b(i,j) is defined as below.

```
b(i,j) = max {
            a[i],
            a[i]+a[i+1],
            a[i]+a[i+1]+a[i+2],
            ......
            a[i]+a[i+1]+a[i+2]+......+a[j]
        }
```

Lets go little further, do some sample calculations based on n values.

n-value	possible Queries	b-elements
1	query(1,1)	b(1,1)

2	query(1,1),query(1,2)
	query(2,2)
		b(1,1),b(1,2)
		b(2,2)

3	query(1,1),query(1,2),query(1,3)
	query(2,2),query(2,3)
	query(3,3)
		b(1,1),b(1,2),b(1,3)
		b(2,2),b(2,3)
		b(3,3)

So you might have already observed the pattern that we are looking for.
Now Formal definitions of query(x,y) and b(i,j).

```
query(x,y) -> is the sub-sequence sum, whose sum is maximum in all sub-sequences by satisfying uniqueness criteria.
b(i,j)     -> is the maximum sub-sequence in the range[i...j] by satisfying uniqueness criteria.
```

```
Algorithm:
    Create an array/segment Tree as 'b'. (Segment tree is suggested, will explain that later)
    For i = 1 to N
        ->update the 'b' by inserting the element a[i].
        ->Answer all the queries query(x,y) where x<=y and y=i.
```

How to handle repeativeness of the elements ??

```
 Assume that a[j] is the element to be inserted,
           a[j] will make its contribution towards the elements b(i,j) {where i<=j}.
 Insertion operation is done based on the assumption that,
           all the elements in the range a[i...j] will be unique.

 Lets assume that there is an element a[m] {where i <= m <= j} which is duplicate of a[j].
 By our theory, a[m] might have already made contribution towards the elements b(i,m) {where i<=m}.
 so elements b(k,j)  {where m+1 <= k <= j} can include a[j] which is unique for them now.

 In Other words, a[j] can make contribution to the elmements b(k,j)
             {where m+1 <= k <= j and m-is the last know position of a[j].}
 i.e
 a[j] can be updated in the range b(last[a[i]]+1,j) which preserves our uniqueness condition.
```

Complexity of the Query. ??

```
    To calculate each b(i,j), logN operations are required.(Worst case).
    To find max of b(i,j), for a query it will take y-x+1 no.of calulations under that element.

    ** b(i,j) is calculated based on below, so total j-i+1 child element calculations.
    {b(i+0,i+0), b(i+0,i+1), b(i+0,i+2).........b(i+0,j)
               b(i+1,i+1), b(i+1,i+2).........b(i+1,j)
                             b(i+1,i+2).........b(i+1,j)
                                      ...........
                                       b(j,j)}

    Total Complexity = O(N) = (j-i+1)O(logN) = O((j-i+1)*logN);
    worst case O(NlogN).
    *** Save the processing time by updating child elements by maintaining a segment Tree.
        So the complexity will come down to O(logN)
```

How to maintain the Segment Tree Node.??

```
/*
    Suppose a[j] is to be updated.
    Please be aware that all the queries with y<j are already Calculated before hand.
    So update the tree only to maintain the queries where y==j.

    i.e. At a certain point of time, after updating a[j],
        Leaf Nodes: will contain info about the range queries required query(1,j),query(2,j),query(3,j).....query(j,j).
        Non-Leaf Nodes: will contain info about controling interval mentioned below them.

    So maintain 4-values at each node.
*/
```

```
struct node{
    int max;  //-> Indicates maximum sum in the interval.
    int evermax;  //-> Indicates maximum sum the history of this range.
    int change;   //-> Indicates the value to be modified in the given range.
    int everchange; //-> Indicates the value to be modified in the history of this range.
};
```

For example take the below input.

```
4
4 -2 3 -2
```

Inserting the A[4] = -2 in the sequence, the tree will look like.

```
                        (5,5,0,0)
        (5,5,0,0)                (3,3,-2,0)
(0,0,5,5)       (0,0,1,1)       (0,0,3,3)       (0,0,0,0)
```

Look at the Leaf nodes, these nodes will address these queries with y==4. May be you can name them as result nodes. :D

```
                        (5,5,0,0)
        (5,5,0,0)                (3,3,-2,0)
(0,0,5,5)       (0,0,1,1)       (0,0,3,3)       (0,0,0,0)
 q(1,4)          q(2,4)          q(3,4)          q(4,4)
```

Look at the Non-Leaf nodes, they will have the control information(change,everchange), may be you can name them as control nodes. ;)

```
                        (5,5,0,0) -> control the interval (1,4)
        (5,5,0,0)              (3,3,-2,0)
   control the interval (1,2)      control the interval (3,4) -> -2 will be included in (3,4) range only
(0,0,5,5)       (0,0,1,1)       (0,0,3,3)       (0,0,0,0)
```

Operations required:
You might have figured out that, after inserting a elment a[j] we are no longer looking at queries query(x,y) where y<j.
Hence we are updating the childrens with the current values as they are being updated.
so Operations required will be,

```
        updateChilds(node& left,node& right)//with the current insertion values(i.e changes required are propagated.)
        clearNode()                     // Clear the non-Leaf nodes.
        updateNode(node& left,node& right)  // Make control nodes are also capable of mataining the result information.
        setNode(int val)                 //Set result nodes to the respective values.
```

Following is my code,for which i have got AC after 20 attempts :P.
Be sure to look out for 'long long' ;)

```
// =====================================================================================
//       Filename:  _GSS2.cpp
//    Description:
//        Created:  07/25/2013 08:42:25 PM
//         Author:  BrOkEN@!
// =====================================================================================
#include<fstream>
#include<iostream>
#include<sstream>
#include<bitset>
#include<deque>
#include<list>
#include<map>
#include<queue>
#include<set>
#include<stack>
#include<vector>
#include<algorithm>
#include<iterator>
#include<string>
#include<cassert>
#include<cctype>
#include<climits>
#include<cmath>
#include<cstddef>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<ctime>

#define TIMER 0
#define FOR(i,a,b) for(typeof((a)) i=(a); i <= (b) ; ++i)
#define FOREACH(it,x) for(typeof((x).begin()) it=(x).begin(); it != (x).end() ; ++it)
template< class T > inline T _max(const T a, const T b) { return (!(a < b) ? a : b); }


using namespace std;

const int MAX = 100001;
const int INF = 0x7f7f7f7f;
//const int INF = 0;

typedef pair<int,int> PI;
typedef vector<PI> VI;
typedef long long int __int;

typedef struct node
{
    __int max,evermax,change,everchange;

    node split(node& l, node& r){
    } // No Need of Split Function

    node updateChilds(node& l, node& r){
        l.everchange = _max(l.everchange,l.change+everchange);
        l.change += change;
        r.everchange = _max(r.everchange,r.change+everchange);
        r.change += change;
    }

    node setNode(int val){
        change += val;
        everchange = _max(change,everchange);
    }
    node clearNode(){
        change = 0;
        everchange = -INF;
    }
    node updateNode(node& l, node& r){
        max = _max(_max(l.max,l.evermax+l.everchange),_max(r.max,r.evermax+r.everchange));
        evermax = _max(l.evermax+l.change,r.evermax+r.change);
    }

} node;

typedef struct query{
    int l,r,p;
} query;

int a[MAX],lastp[MAX<<1],lastq[MAX];
__int ans[MAX];
node T[1<<18];
query q[MAX];


void update(int node, int i, int j, int a, int b, int val) {
    if(a <= i && j <= b) {
            T[node].setNode(val);
    }
    else {
        int m = (i + j)/2;
        T[node].updateChilds(T[2*node],T[2*node+1]);
        T[node].clearNode();
        if(a <= m) update(2*node, i, m, a, b, val);
        if(m < b) update(2*node+1, m+1, j, a, b, val);
        T[node].updateNode(T[2*node],T[2*node+1]);
    }
}
```

```

__int range_query(int node, int i, int j, int a, int b) {
    if(a <= i && j <= b){
        return _max(T[node].max,T[node].evermax + T[node].everchange);
    }
    else {
        int m = (i + j)/2;
        T[node].updateChilds(T[2*node],T[2*node+1]);
        T[node].clearNode();
        T[node].updateNode(T[2*node],T[2*node+1]);
        return _max((a <= m ? range_query(2*node, i, m, a, b) : -INF),
                    (m < b ? range_query(2*node+1, m+1, j, a, b) : -INF));
    }
}

int main(){

    int N=0;
    scanf("%d",&N);
    FOR(i,1,N){
        scanf("%d", &a[i]);
    }

    int M=0;
    scanf("%d",&M);
    FOR(i,1,M){
            scanf("%d%d",&q[i].l,&q[i].r);
            q[i].p = lastq[q[i].r];
     lastq[q[i].r]=i;
    }

    FOR(i,1,N){
        update(1, 1, N, lastp[a[i]+100000] + 1, i, a[i]);
        lastp[a[i]+100000] = i;
        for(int j=lastq[i];j;j=q[j].p){
                ans[j]=range_query(1, 1, N, q[j].l, q[j].r);
        }
    }

    FOR(i,1,M){printf("%lld\n", ans[i]);}



    return 0;
}
```