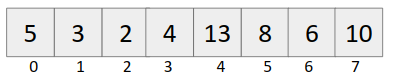
**Segment Tree - Max**

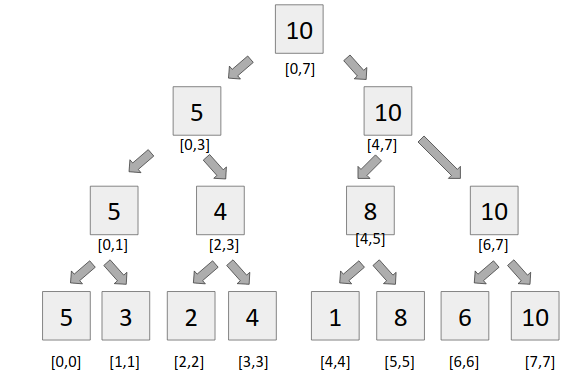
**Problem**

Build a segment tree with max build, query and update.

Example



**Structure**



**Approach**

**Building a segment tree**

It is very simple to build a segment tree, we use divide and conquer approach to build the segment tree.

**Query**

For query, we see two types of segments

* Complete overlapping segments - When our st Partial overlapping segments and en lies completely in the range [l,r], it is called complete overlapping segment.
* Partial overlapping segments - When our st and en does not lie completely in the range [l,r], it is called partial overlapping segment.

**Code**

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

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

int tree[4\*N], a[N];

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

if(st == en){

tree[node] = a[st];

return;

}

int mid = (st + en)/2;

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

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

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

}

int query(int node, int st, int en, int l, int r){

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

return MOD;

}

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

return tree[node];

int mid = (st + en)/2;

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

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

return max(q1, q2);

}

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

if(st == en){

a[st] = val;

tree[node] = val;

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

}

signed main(){

int n,m;

cin >> n >> m;

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

cin >> a[i];

}

build(1,0,n-1);

while(m--){

int type;

cin >> type;

if(type == 1){

int idx,val;

cin >> idx >> val;

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

}

else{

int l,r;

cin >> l >> r;

int ans = query(1,0,n-1,l,r-1);

cout << ans << endl;

}

}

return 0;

}