Segment Trees

Youtube link :

https://youtu.be/iksoKpujJPI

Contents:

Introduction to Segment Trees

Q. You are given an array of n integers. For q queries.
Queries are of 2 types:
(i) change (i,v): Change: a[i] to v
(ii) min (L,R) : Print minimum in range [L,R) or [L,R-1]

Can u solve it using sparse table ? Time complexity?
- Yes, but for update, we need to modify the whole sparse table in the worst case. It would be approx O(n log n) for 1 update query

Can u solve it using Fenwick tree or Binary Indexed Tree ?
- Minimum is not reversible. So, we can't use BIT.
We use segment trees to perform both queries in O(log N).

Link:
https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/B

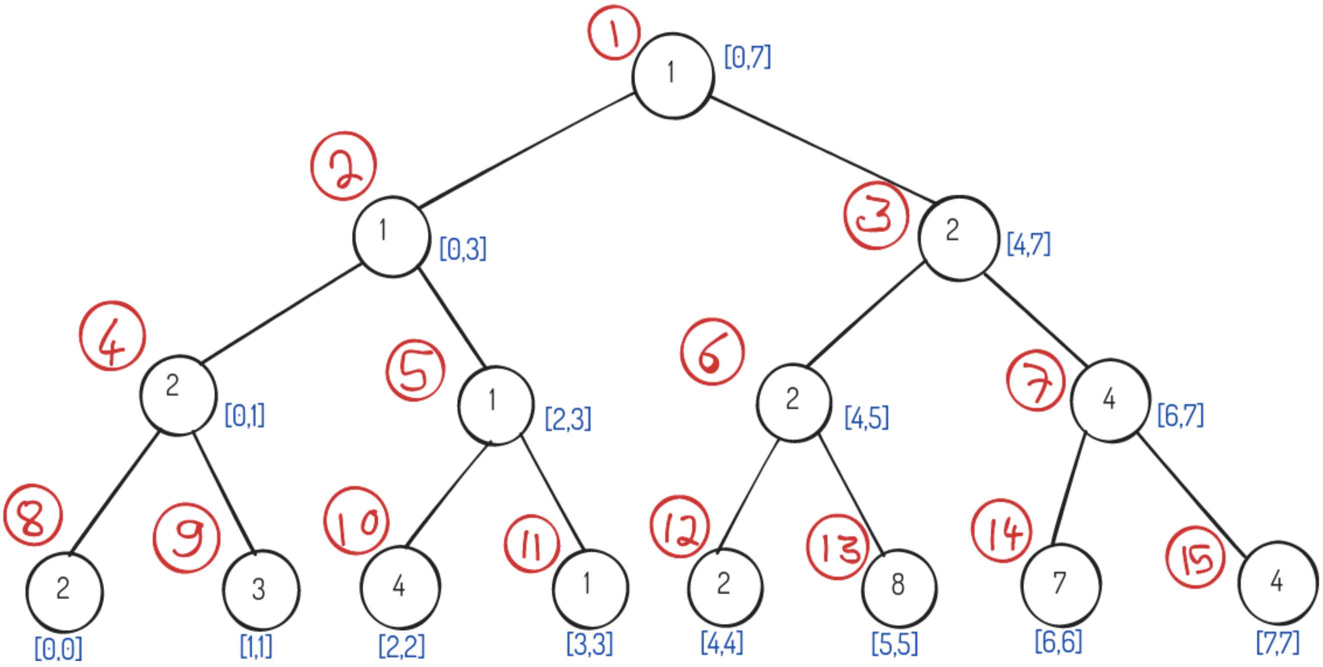
Let's build the segment tree for minimum queries.
Consider Arr=[2,3,4,1,2,8,7,4]

These elements will be the leaves of segment tree.

We are numbering nodes, starting from root, from 1.

Which node is left child of node i ?
- 2*i

Which node is right child of node i?
- 2*i+1



For visualization, use
https://csacademy.com/lesson/segment_trees/

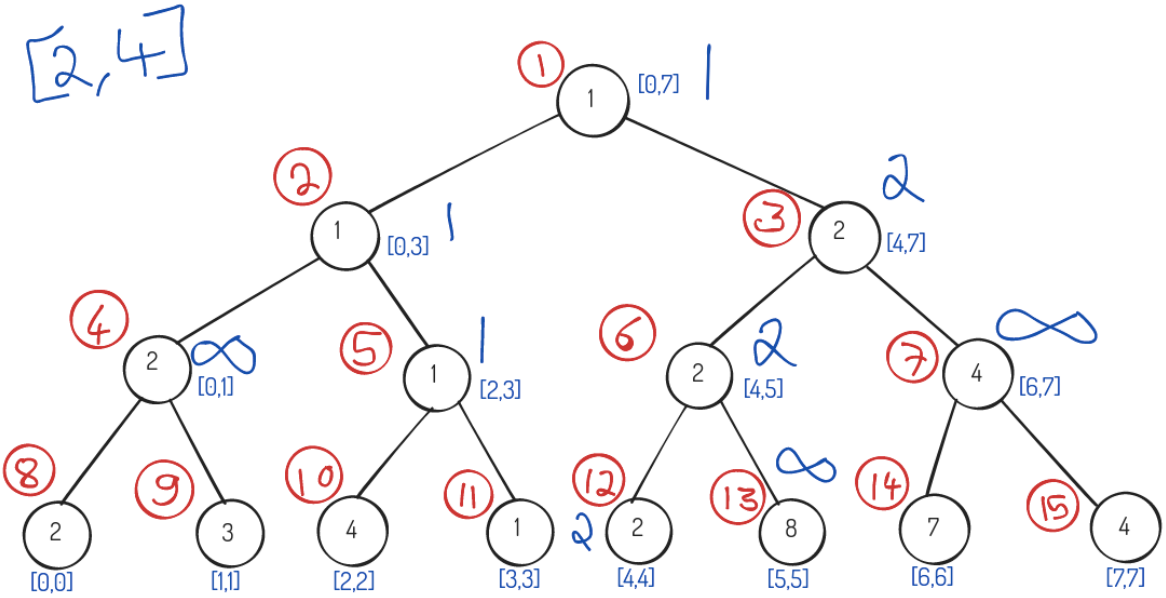
For query,

Just start checking from root.

3 cases:
1. If the interval of nodes is entirely inside the given range, return ans.

2. If interval of node is entirely outside the given range,
return infinity (As it should not affect our answer).

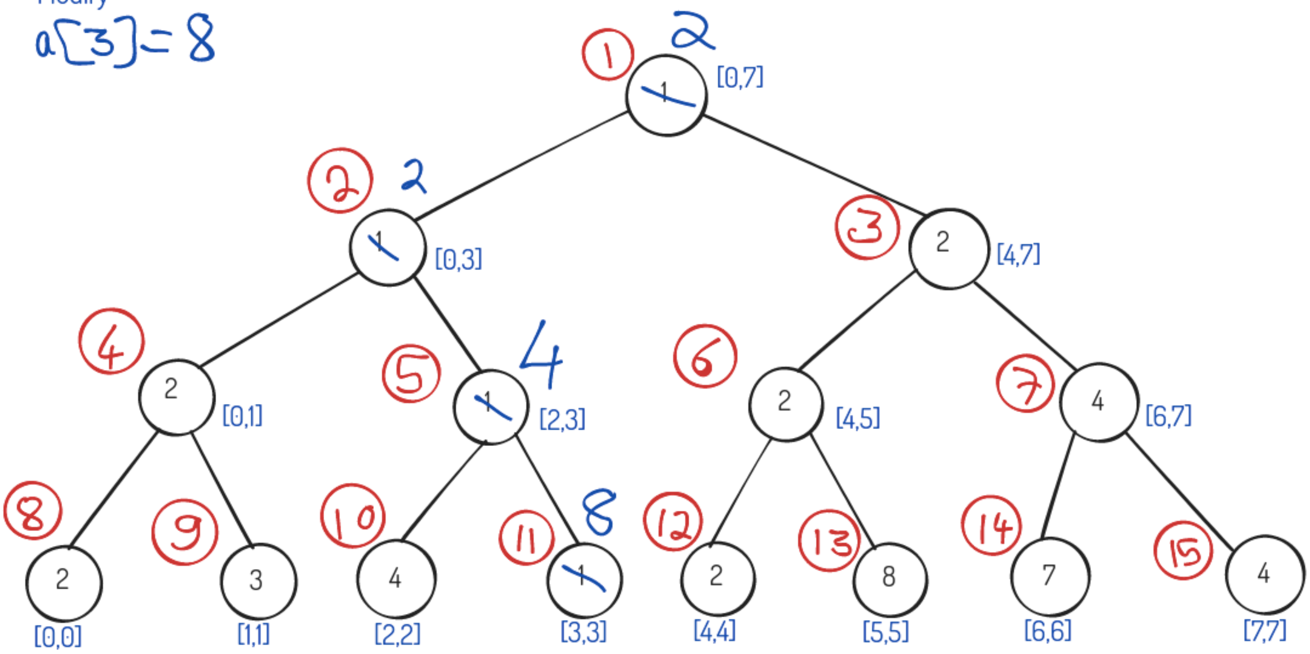
3. And if the interval of node is partially inside and partially outside, we call the recursive function for both its children and return the minimum of both of their answers.



Update

Modify

a[3]=8



First, building segment tree:
min(min(a,b),c) = min(a,min(b,c))
[Associative operation]

If we have n elements in array, how many nodes will it take to build the segment tree ?

- n + n/2 + n/4 + …. 1

($1 + 2 + 2^2 + 2^3 + 2^4 + \dots 2^{\text{ceil}(\log n)}$) = 4*n nodes in segment tree

Q. How many levels are there in the segment tree ?

- log n

// Array indexing is from 0 to n-1
// Range [L,R)

```
int n; // size of arr
vector<int> arr;  // original array
vector<int> seg; // segment array
```

// Give a size of 4*n+1 to seg[]
// id = id of node in segment tree
// [L,R)

```
void build(int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
      // Leaf node
      seg[id]=arr[L];
      return;
   }
   int mid=(L+R)/2;
   build(2*id, L, mid); // left child
   build(2*id+1, mid, R); // right child
   seg[id] = min(seg[id*2], seg[id*2+1]);
}
```

Time Complexity: O(n)
[For each node, we will calculate seg[id] only once. There are total 4*n nodes]
// In main(), call build()
// [0,n)
// Given range is [x,y)

```
const inf=1e9+1;
int query(int x, int y, int id=1, int L=0, int R=n)
{
   if( L>=x && R<=y  ) {
        // When [L,R) is entirely inside [x,y)
       return seg[id];
   }
  if ( L>=y || R<=x ) {
      // When [L,R) is entirely outside [x,y)
       return inf;
   }
    int mid=(l+r)/2;
    return min(query(x,y,id*2,L,mid ),
    query(x,y,id*2+1,mid,R ));
}
```

// In main(), cout<<query(2,4);
// Time complexity: O(log n)
For every level, there can be maximum 2 nodes which are partially inside given range and partially node.
There are log2 n levels.

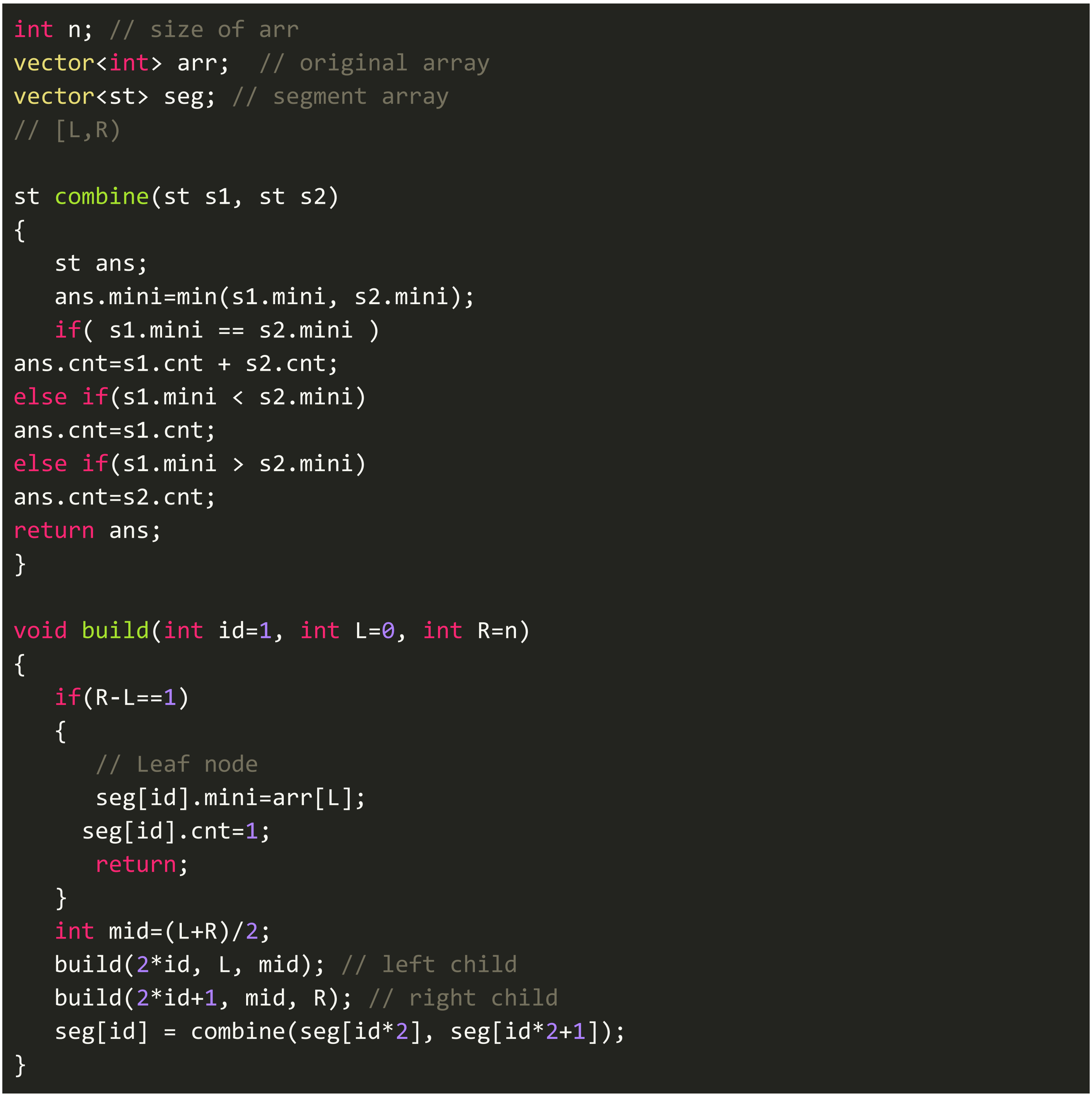
```
void modify(int pos, int val, int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
       // leaf node
      arr[pos]=val;
      seg[id]=val;
      return;
   }
  int mid=(L+R)/2;
  if(pos<mid)
  {
    modify(pos,val,id*2, L, mid);
  }
else {
    modify(pos,val,id*2+1, mid, R);
  }
seg[id]=min(seg[id*2], seg[id*2+1]);
}
```

Time Complexity: O(log N)
Since, we are going to maximum 1 node in each level.

Segment tree will work for every associative operation.
(a @ b) @ c = a @ (b @ c)

Q1.)
https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/C

```
struct {
  int mini; // minimum element in the range of node
  int cnt; // count of minimum element in range of node
} st;
```

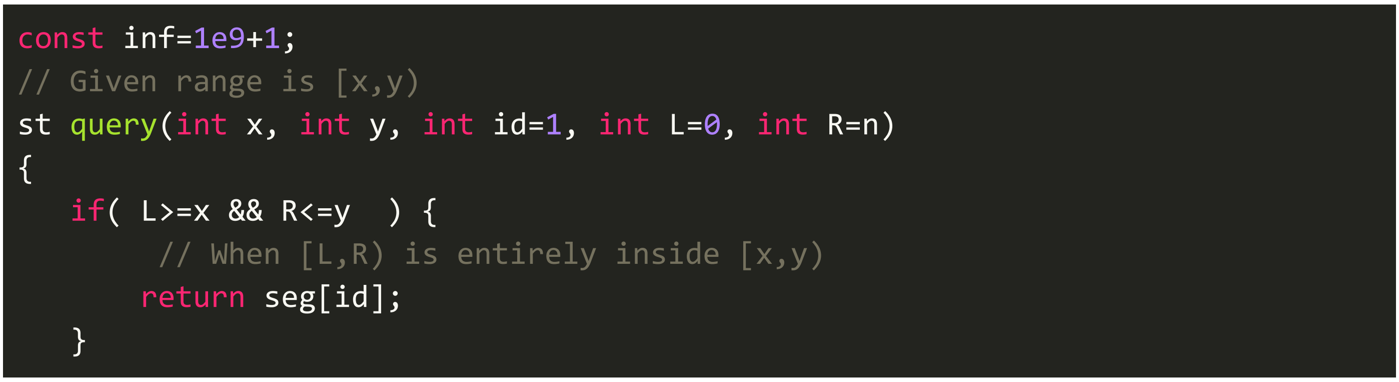


```
int n; // size of arr
vector<int> arr;  // original array
vector<st> seg; // segment array
// [L,R)

st combine(st s1, st s2)
{
   st ans;
   ans.mini=min(s1.mini, s2.mini);
   if( s1.mini == s2.mini )
ans.cnt=s1.cnt + s2.cnt;
else if(s1.mini < s2.mini)
ans.cnt=s1.cnt;
else if(s1.mini > s2.mini)
ans.cnt=s2.cnt;
return ans;
}

void build(int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
      // Leaf node
      seg[id].mini=arr[L];
     seg[id].cnt=1;
      return;
   }
   int mid=(L+R)/2;
   build(2*id, L, mid); // left child
   build(2*id+1, mid, R); // right child
   seg[id] = combine(seg[id*2], seg[id*2+1]);
}
```

Time Complexity of build : O(n)



```
const inf=1e9+1;
// Given range is [x,y)
st query(int x, int y, int id=1, int L=0, int R=n)
{
   if( L>=x && R<=y  ) {
        // When [L,R) is entirely inside [x,y)
       return seg[id];
   }
```

```
  if ( L>=y || R<=x ) {
      // When [L,R) is entirely outside [x,y)
       st temp={inf,0};
       return temp;
    }
    int mid=(l+r)/2;
    return combine(query(x,y,id*2,L,mid ),
    query(x,y,id*2+1,mid,R ));
}
```

// In main(), st ans= query(2,4);

// cout<<ans.mini << ' '<<ans.cnt<<'\n';

Time complexity of query: O(log n)

```
void modify(int pos, int val, int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
       // leaf node
      arr[pos]=val;
      st temp={val, 1};
      seg[id]=temp;
      return;
   }
  int mid=(L+R)/2;
  if(pos<mid)
  {
    modify(pos,val,id*2, L, mid);
  }
else {
    modify(pos,val,id*2+1, mid, R);
  }
seg[id]=combine(seg[id*2], seg[id*2+1]);
}
```

Practice Questions:

1. https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/A
2. https://codeforces.com/edu/course/2/lesson/4/2/practice/contest/273278/problem/A

Q. CSES - Hotel Queries

https://cses.fi/problemset/task/1143

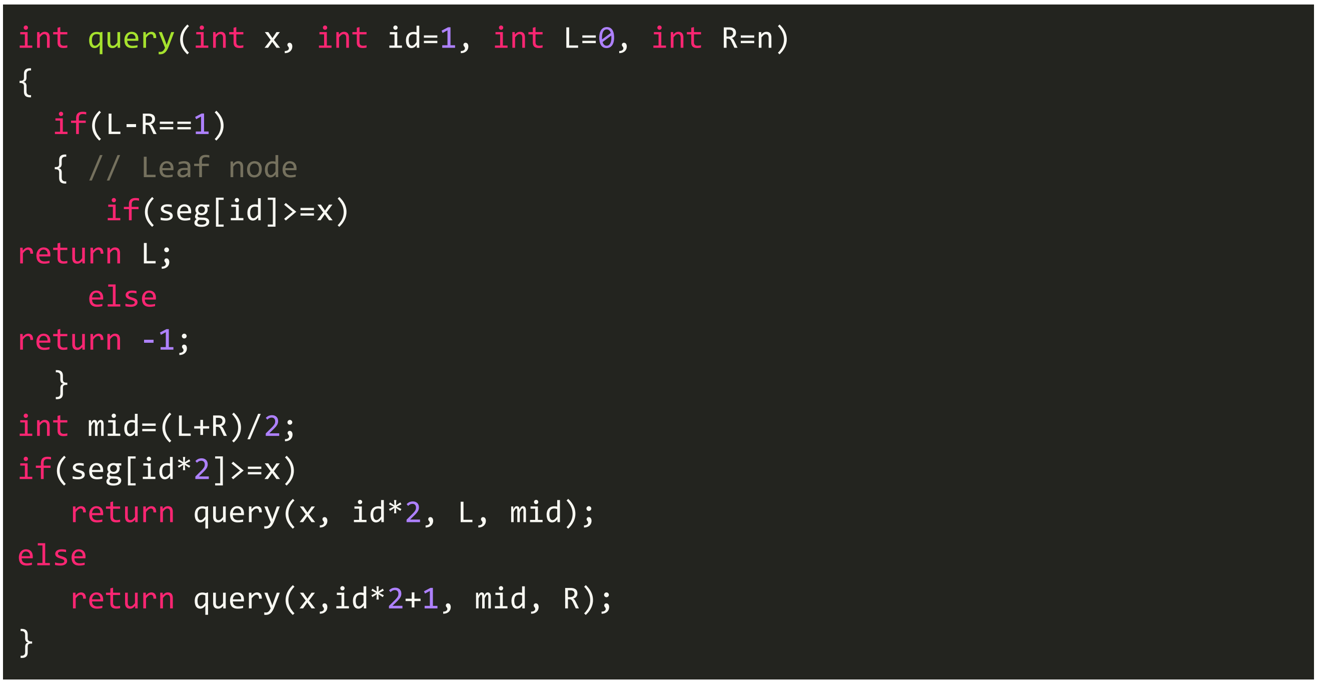
Given an array of free rooms in hotels,
arr[i]= Free rooms in hotel[i]
we need to perform 2 operations:
i. Modify an element in array
ii. For number of rooms x, What is the minimum i, such that arr[i]>=x

Brute Force Approach:
Update operation in O(1)
But 1 query for finding sufficient rooms will take O(n)
Total time complexity is: O(n * m) = 10^10 [TLE]

Optimised Idea:
Build a max segment tree.
If seg[id]<x, then answer doesn't exist
Move to left child, if seg[id*2] >=x
Otherwise move to right child

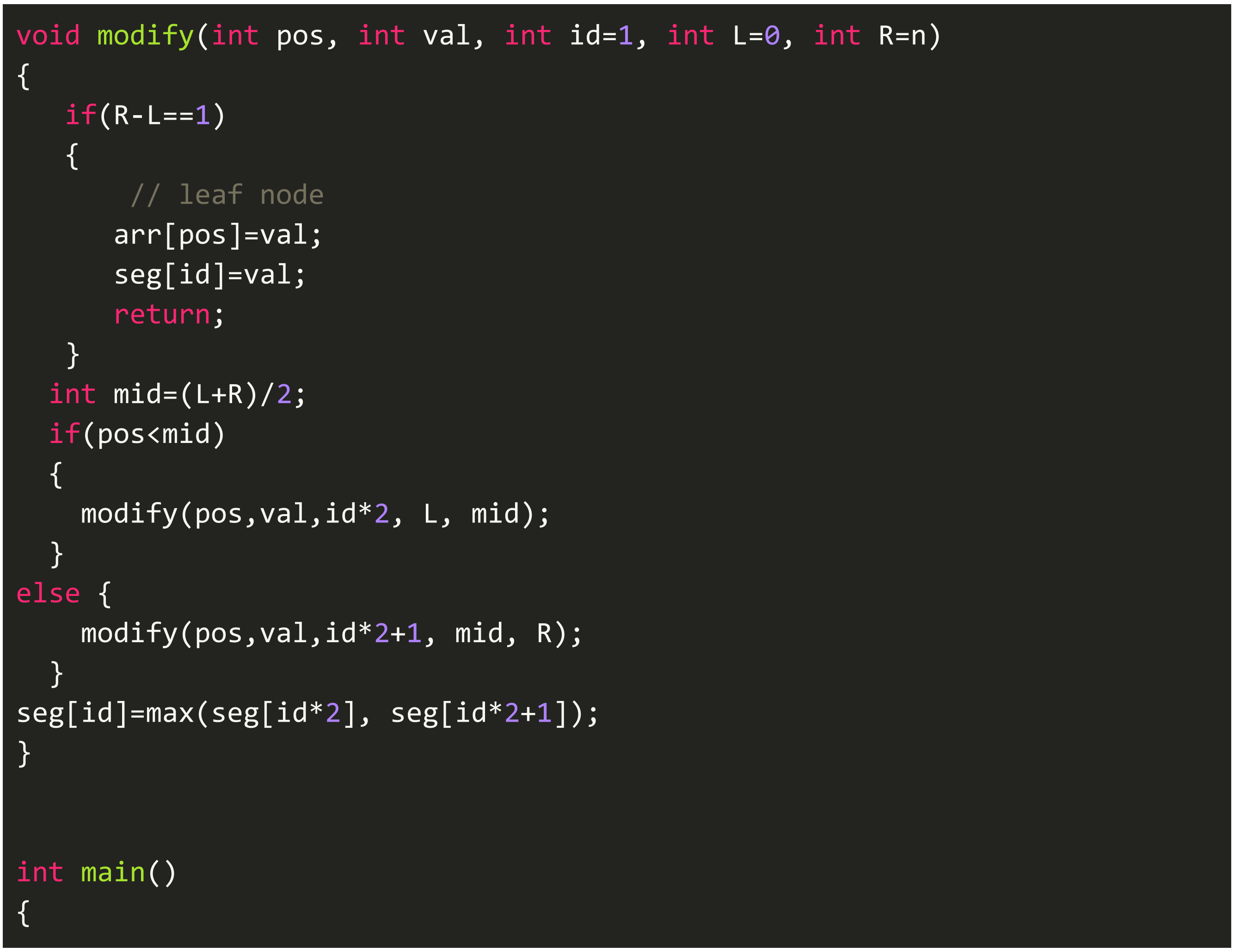
```
int n; // size of arr
vector<int> arr;  // original array
vector<int> seg; // segment array
// Give a size of 4*n+1 to seg[]
// id = id of node in segment tree
// [L,R)
void build(int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
      // Leaf node
      seg[id]=arr[L];
      return;
   }
   int mid=(L+R)/2;
   build(2*id, L, mid); // left child
   build(2*id+1, mid, R); // right child
   seg[id] = max(seg[id*2], seg[id*2+1]);
}
```

Time Complexity: O(n)
[For each node, we will calculate seg[id] only once. There are total 4*n nodes]
// In main(), build()
// [0,n)



```
int query(int x, int id=1, int L=0, int R=n)
{
  if(L-R==1)
  { // Leaf node
     if(seg[id]>=x)
return L;
    else
return -1;
  }
int mid=(L+R)/2;
if(seg[id*2]>=x)
   return query(x, id*2, L, mid);
else
   return query(x,id*2+1, mid, R);
}
```

// Time complexity: O(log n)



```
void modify(int pos, int val, int id=1, int L=0, int R=n)
{
   if(R-L==1)
   {
       // leaf node
      arr[pos]=val;
      seg[id]=val;
      return;
   }
  int mid=(L+R)/2;
  if(pos<mid)
  {
    modify(pos,val,id*2, L, mid);
  }
else {
    modify(pos,val,id*2+1, mid, R);
  }
seg[id]=max(seg[id*2], seg[id*2+1]);
}


int main()
{
```

```
int n,m;
cin>>n>>m;
vec.resize(n);
seg.resize(4*n+1);

for(int i=0; i<n; i++)
     cin>>vec[i];

build();

while(m--)
{
int x;
cin>>x;
int ans = query(x);
cout<< ans +1<<' ';
if(ans==-1)
continue;
modify(ans, vec[ans]-x);
}
return 0;
}
```

Q. SPOJ KQUERY
https://www.spoj.com/problems/KQUERY/

For each node in the segment tree, store a vector of all the elements in the range of node in sorted (increasing) order.
If any sorted is given ,we can find the number of elements >= x, in O(log N) by binary search.
n-(upper_bound(vec.begin(),vec.end(),x)-vec.begin());

For merging 2 sorted vectors, time complexity: O(n)

a: [1, 4, 7]
b: [2, 5, 88]
int i=0,j=0
while(i<n && j<n)
{
a[i]<b[j]: store a[i] in new array and i++
else store b[j] in new array and j++
}

```
while(i<n)
{
insert all elements of a in new array
}

while(j<n)
{
insert all elements of b in new array
}
```

Total space complexity: O(n log n)
[There are log n levels. Each element would appear in log n ranges only.]

Code:

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

vector<int> a;
vector<vector<int>> seg;
int n, m;
// Root node covers the range [0,n-1] or [0,n)

vector<int> create(int num) {
 vector<int> temp;
 temp.push_back(num);
 return temp;
}

vector<int> combine(const vector<int>& a, const vector<int>& b) {
 int m = a.size(), n = b.size();
 int i = 0, j = 0;
 vector<int> c;

 while (i < m && j < n) {
   if (a[i] <= b[j]) {
     c.push_back(a[i]);
     i++;
   } else {
     c.push_back(b[j]);
     j++;
   }
 }

 while (i < m) {
   c.push_back(a[i]);
   i++;
 }

 while (j < n) {
   c.push_back(b[j]);
   j++;
 }
 return c;
}

// Range is [l,r-1] or [l,r)
void build(int id = 1, int l = 0, int r = n) {
 if (r - l == 1) {
   seg[id] = create(a[l]);
   return;
 }
 int mid = (l + r) / 2;
 build(id * 2, l, mid);
 build(id * 2 + 1, mid, r);
 seg[id] = combine(seg[id * 2], seg[id * 2 + 1]);
}
// Time complexity: O(n*log(n))

// x : no. of rooms
int query(int x, int y, int k, int id = 1, int l = 0, int r = n) {
 if (r <= x || l >= y) return 0;
 if (l >= x && r <= y) {
   int ans =
       (int)seg[id].size() -
       (upper_bound(seg[id].begin(), seg[id].end(), k) - seg[id].begin());
   return ans;
 }
 int mid = (l + r) / 2;
 int l_ans = query(x, y, k, id * 2, l, mid);
 int r_ans = query(x, y, k, id * 2 + 1, mid, r);
 return l_ans + r_ans;
}
```

```
// Time complexity: O( log^2(N) )

int32_t main() {
 cin >> n;

 a.resize(n);
 seg.resize(4 * n + 1);

 for (int i = 0; i < n; i++) {
   cin >> a[i];
 }

 build();
 cin >> m;

 while (m--) {
   int i, j, k;
   cin >> i >> j >> k;
   i--;
   int ans = query(i, j, k);
   cout << ans << '\n';
 }

 return 0;
}
```

Range update on a segment tree:
[Lazy propagation]

(i) increase(L,R,val) : Increase a[i] by val for all i in [L,R)
(ii) minimum(L,R) : Find minimum in [L,R)

Link:
https://codeforces.com/edu/course/2/lesson/5/2/practice/contest/279653/problem/A

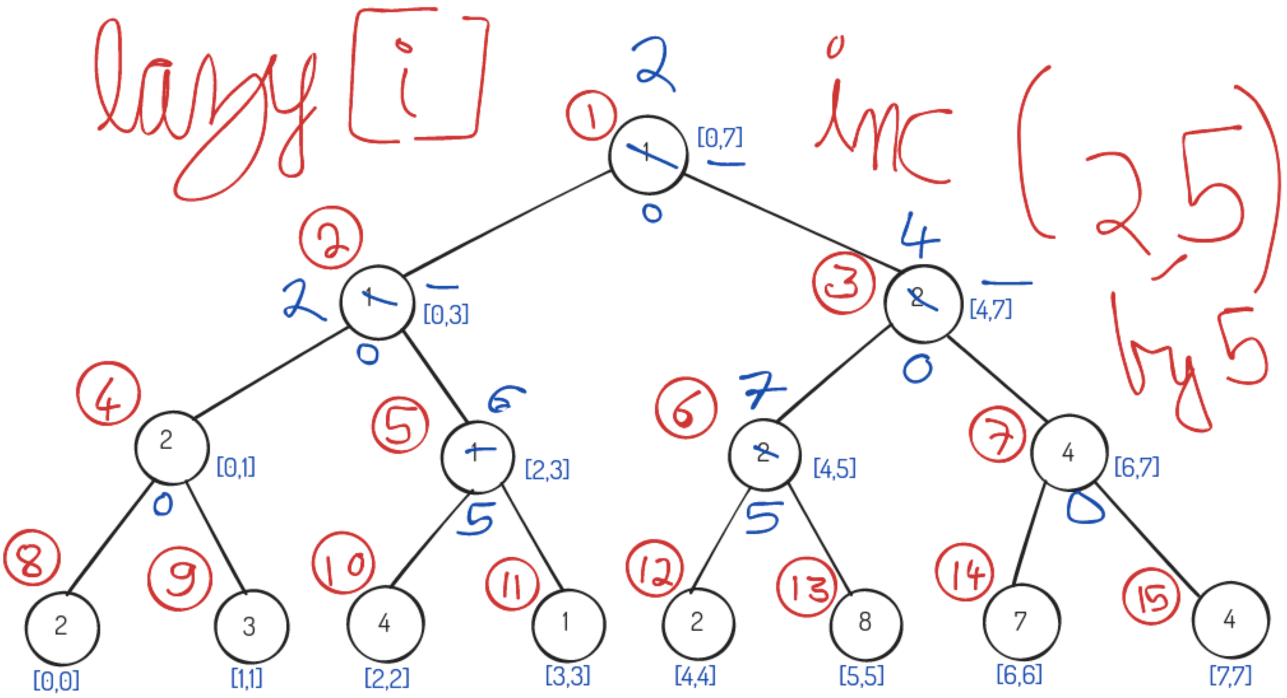
When updating nodes, just update the nodes entirely inside the interval and update the lazy of that node.
Whenever we move to the child nodes in some other query, we will first propagate the lazy values to the children and then, we will call the recursive function for the child nodes.

3 cases for the range of a node:

1. entirely inside the desired range
(update both node value and the lazy value and return)

2. entirely outside the desired range
(do nothing and return)

3. partially inside and partially outside
(Propagate lazy values, then call the recursive function for both its children and re-calculate node value)

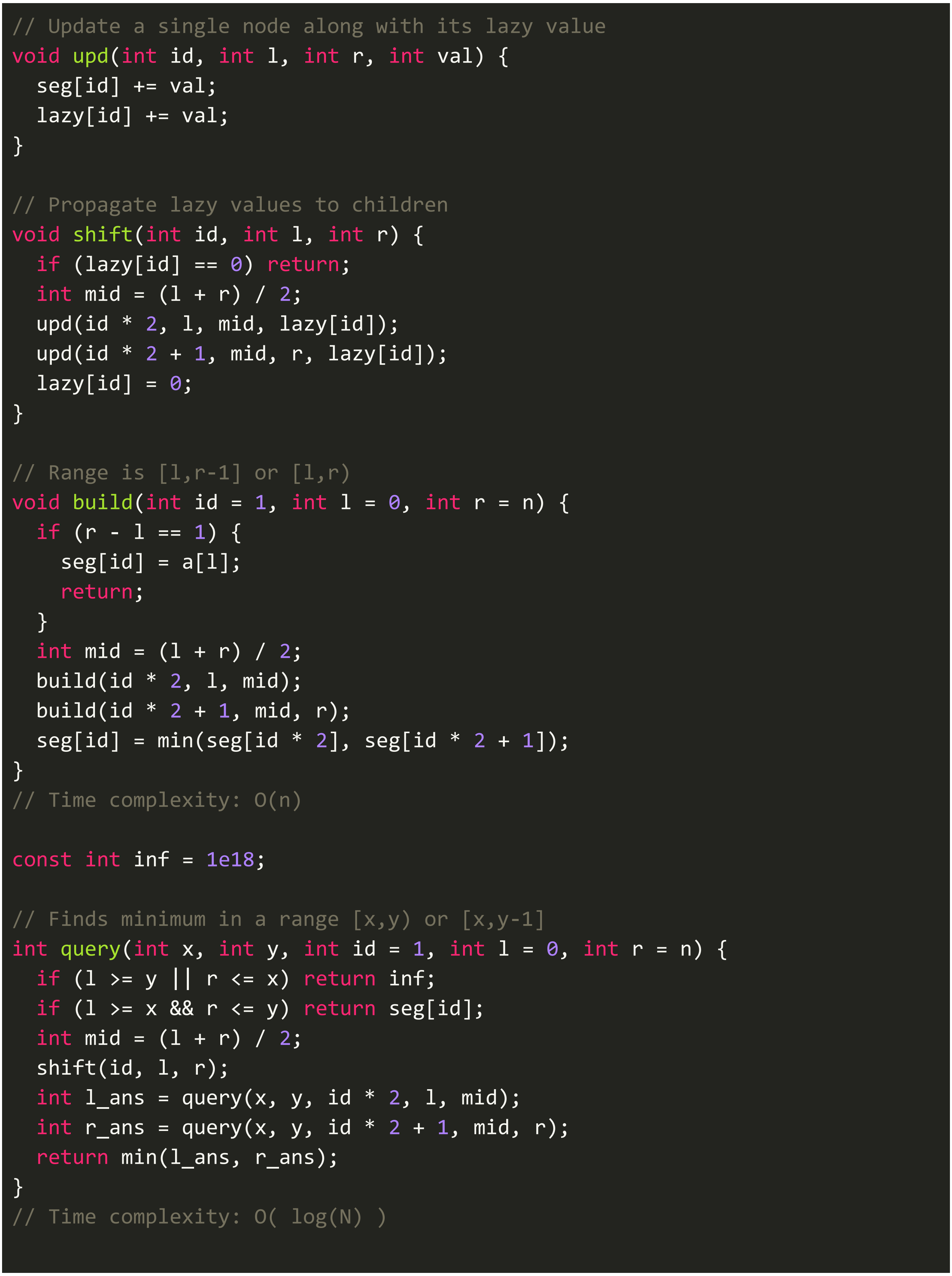


Create a separate lazy array.

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

vector<int> a, seg, lazy;
int n, m;
```



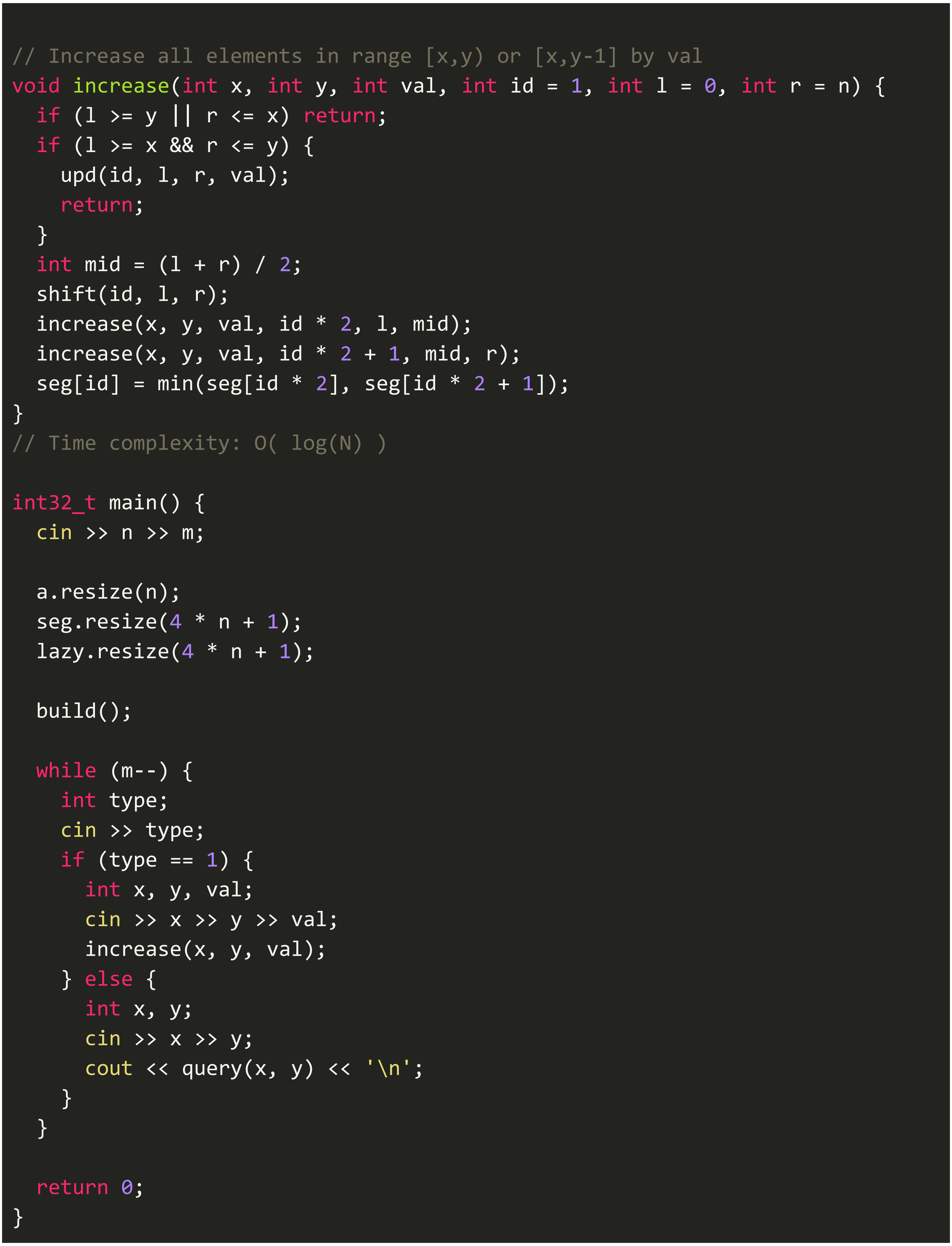
```
// Update a single node along with its lazy value
void upd(int id, int l, int r, int val) {
  seg[id] += val;
  lazy[id] += val;
}

// Propagate lazy values to children
void shift(int id, int l, int r) {
  if (lazy[id] == 0) return;
  int mid = (l + r) / 2;
  upd(id * 2, l, mid, lazy[id]);
  upd(id * 2 + 1, mid, r, lazy[id]);
  lazy[id] = 0;
}

// Range is [l,r-1] or [l,r)
void build(int id = 1, int l = 0, int r = n) {
  if (r - l == 1) {
    seg[id] = a[l];
    return;
  }
  int mid = (l + r) / 2;
  build(id * 2, l, mid);
  build(id * 2 + 1, mid, r);
  seg[id] = min(seg[id * 2], seg[id * 2 + 1]);
}
// Time complexity: O(n)

const int inf = 1e18;

// Finds minimum in a range [x,y) or [x,y-1]
int query(int x, int y, int id = 1, int l = 0, int r = n) {
  if (l >= y || r <= x) return inf;
  if (l >= x && r <= y) return seg[id];
  int mid = (l + r) / 2;
  shift(id, l, r);
  int l_ans = query(x, y, id * 2, l, mid);
  int r_ans = query(x, y, id * 2 + 1, mid, r);
  return min(l_ans, r_ans);
}
// Time complexity: O( log(N) )
```



```
// Increase all elements in range [x,y) or [x,y-1] by val
void increase(int x, int y, int val, int id = 1, int l = 0, int r = n) {
  if (l >= y || r <= x) return;
  if (l >= x && r <= y) {
    upd(id, l, r, val);
    return;
  }
  int mid = (l + r) / 2;
  shift(id, l, r);
  increase(x, y, val, id * 2, l, mid);
  increase(x, y, val, id * 2 + 1, mid, r);
  seg[id] = min(seg[id * 2], seg[id * 2 + 1]);
}
// Time complexity: O( log(N) )

int32_t main() {
  cin >> n >> m;

  a.resize(n);
  seg.resize(4 * n + 1);
  lazy.resize(4 * n + 1);

  build();

  while (m--) {
    int type;
    cin >> type;
    if (type == 1) {
      int x, y, val;
      cin >> x >> y >> val;
      increase(x, y, val);
    } else {
      int x, y;
      cin >> x >> y;
      cout << query(x, y) << '\n';
    }
  }

  return 0;
}
```

If query is for sum, then in upd() function, it becomes: s[id]=s[id]+(R-L)*x;

Practice Problems:

1. https://cses.fi/problemset/task/1735
2. https://www.spoj.com/problems/POSTERS/
3. https://www.codechef.com/problems/IITK1P10
4. https://www.codechef.com/problems/TAQTREE
5. https://codeforces.com/problemset/problem/383/C
6. https://www.codechef.com/problems/TYTACTIC