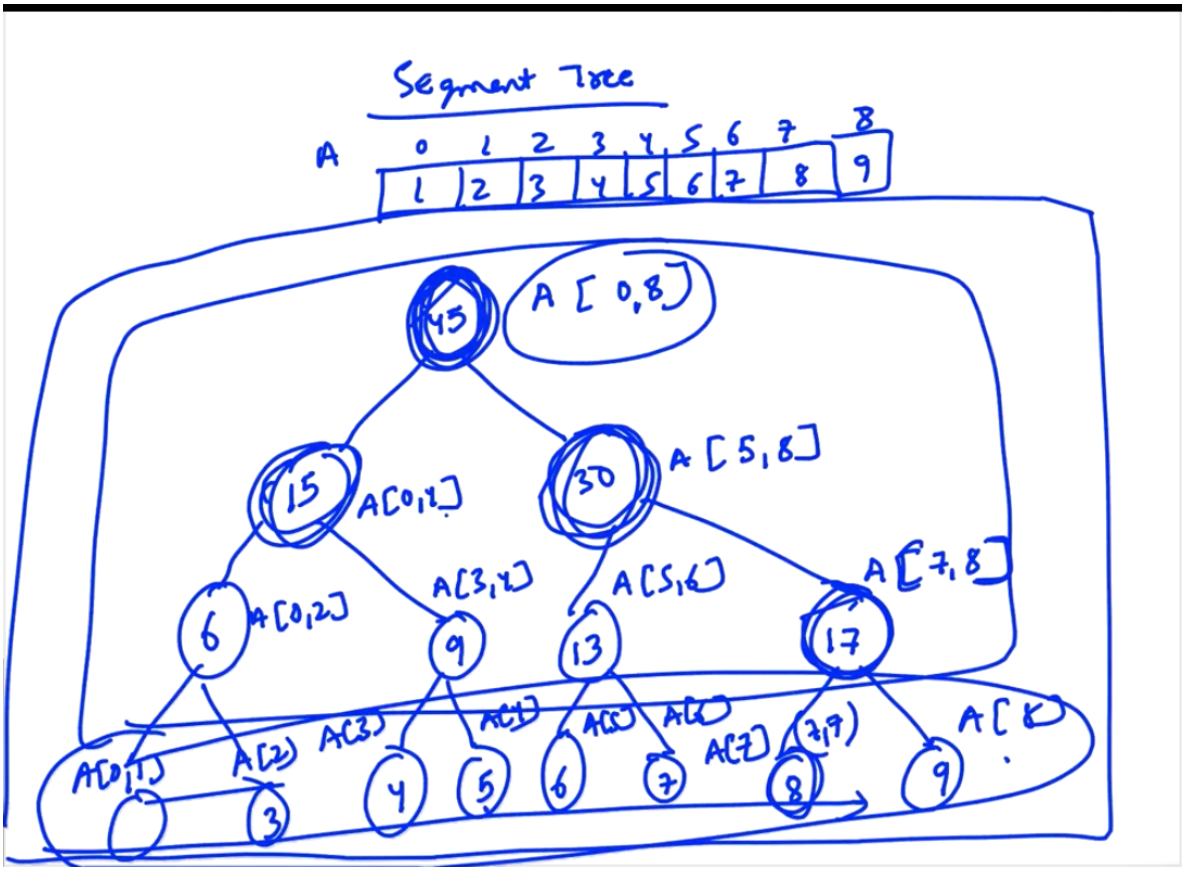
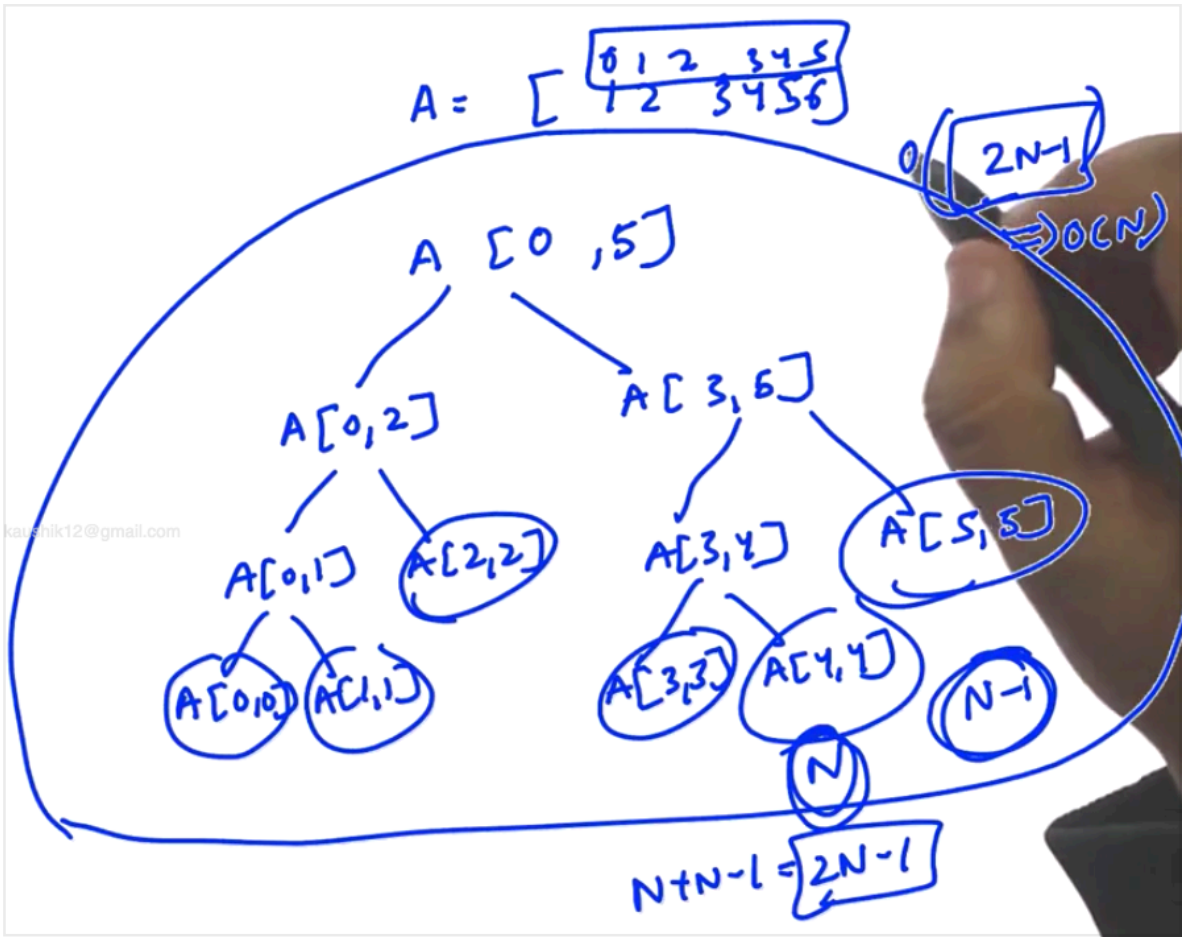
SEGMENT TREES

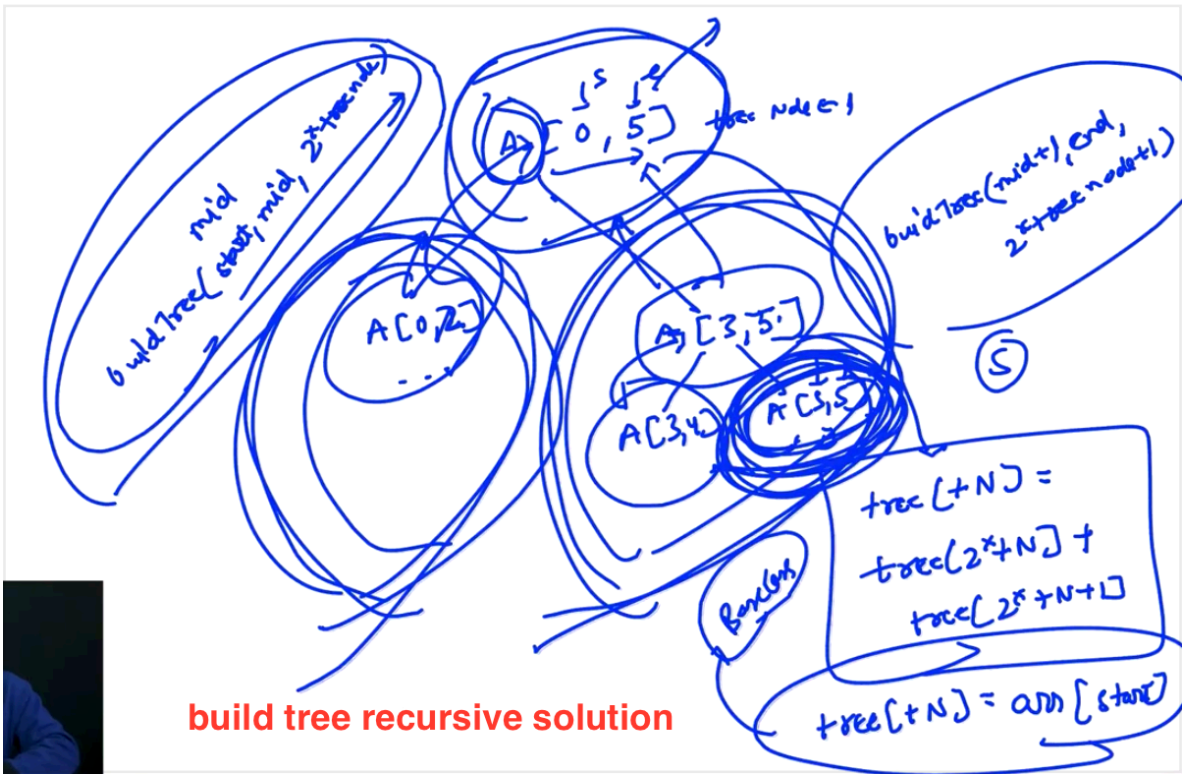
- used when we want output within a range “give me the answer from i to jth index” and this is often accompanied by updates at indexes
- generally used on array problems
- A segment tree updates and queries BOTH in log(n) !
- The leaves of the segment tree are actual indexes of arrray



- The height of the tree in worst case is log(n)
- In a segment tree, we have N-1 internal nodes and n base nodes -> total will be 2N-1
- we will make segment tree via an array IN an array



- We start storing in this array from 1 not 0 to simplify math, -> so instead of 2n-1 WE will have an array of size 2n . But in order to occomodate extra calls also we generally keep in 4n!




```cpp
#include<bits/stdc++.h>
using namespace std;


void buildTree(int* arr,int* tree,int start,int end,int treeNode){

    if(start == end){
        tree[treeNode] = arr[start];
        return;
    }
    int mid = (start+end)/2;

    buildTree(arr,tree,start,mid,2*treeNode);
    buildTree(arr,tree,mid+1,end,2*treeNode+1);

    tree[treeNode] = tree[2*treeNode] + tree[2*treeNode+1];
}
int main(){

    int arr[] = {1,2,3,4,5,6,7,8,9};
    int* tree = new int[18];
    buildTree(arr,tree,0,8,1);

    for(int i=1;i<18;i++){
        cout << tree[i] << endl;
    }
}
```

BUILD

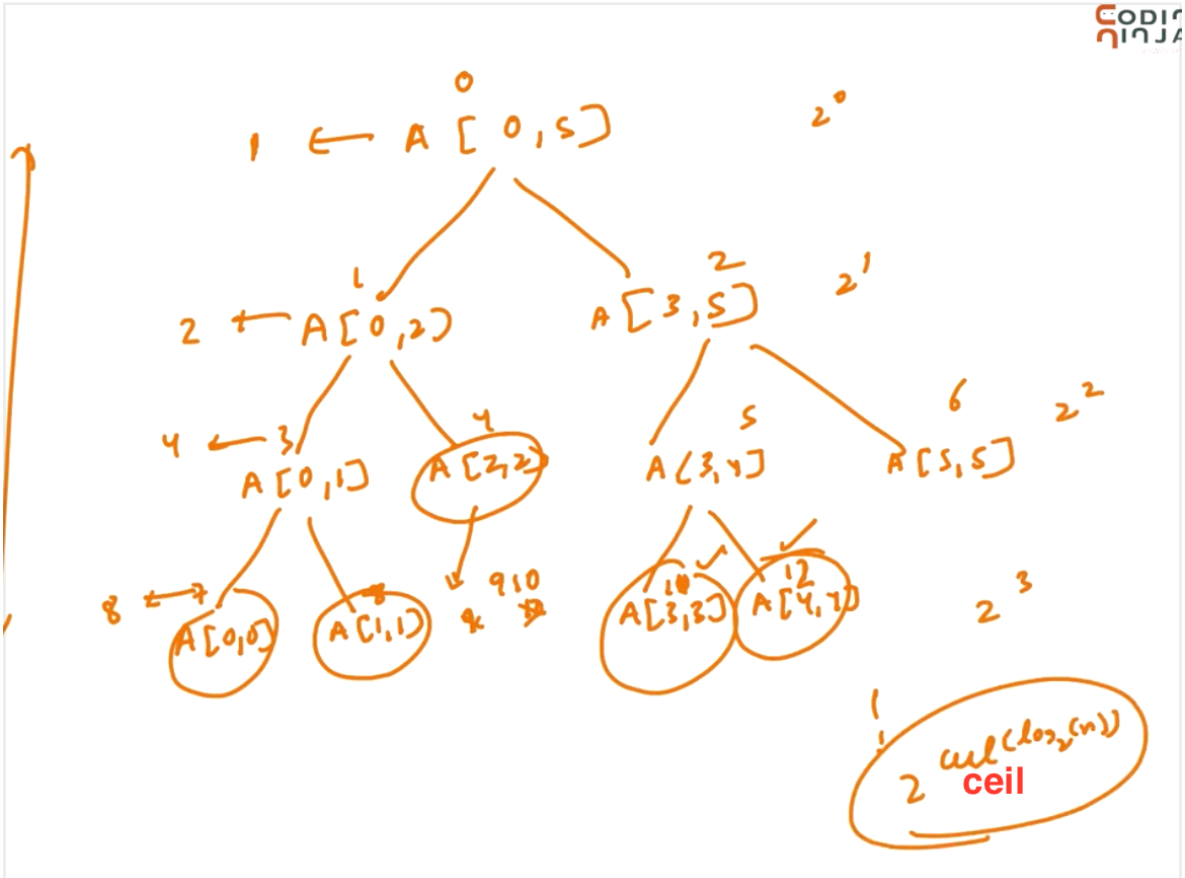
```
void updateTree(int* arr,int *tree,int start,int end,int treeNode,int idx,int value){


    if(start == end){
        arr[idx] = value;
        tree[treeNode] = value;
        return;
    }

    int mid = (start+end)/2;
    if(idx > mid){
        updateTree(arr,tree,mid+1,end,2*treeNode+1,idx,value);
    }else{
        updateTree(arr,tree,start,mid,2*treeNodem,idx,value);
    }
    tree[treeNode] = tree[2*treeNode] + tree[2*treeNode+1];
}
```

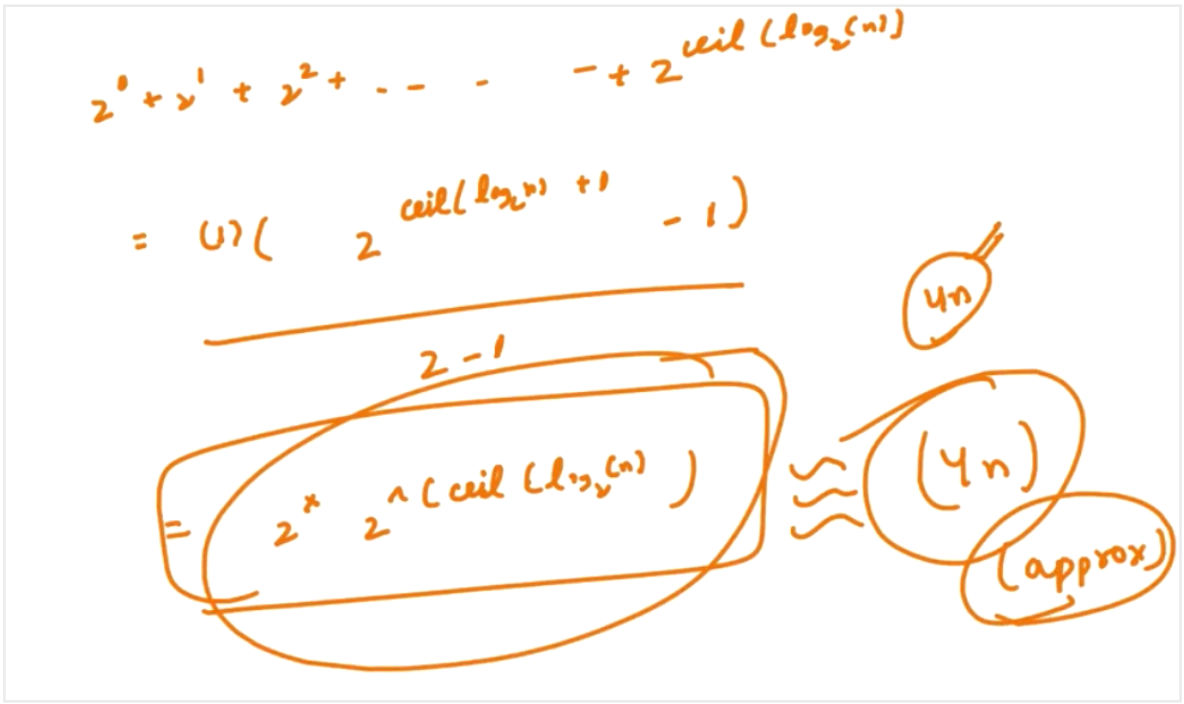
- In segment tree problems variations come from the analysis of what should we store at each index of segment tree
- QUERY ON A SEGMENT TREE
  - Completely inside/outside ->return node_ans/0
  - Partially inside/outside ->further call

```
int query(int* tree,int start,int end,int treeNode,int left,int right){


    //Completely outside given range

    if(start > right || end < left){
        return 0;
    }

    // Completely inside given range
    if(start>=left && end<=right){
        return tree[treeNode];
    }
    // Partially inside and partially outside
    int mid = (start+end)/2;
    int ans1 = query(tree,start,mid,2*treeNode,left,right);
    int ans2 = query(tree,mid+1,end,2*treeNode+1,left,right);

    return ans1 + ans2;
}
```

```
int main(){

    int arr[] = {1,2,3,4,5};
    int* tree = new int[10];
    buildTree(arr,tree,0,4,1);


    updateTree(arr,tree,0,4,1,2,10);

    for(int i=1;i<10;i++){
        cout << tree[i] << endl;
    }

    int ans = query(tree,0,4,1,2,4);
    cout<<"Sum between interval is" << ans <<endl;
```

Size of segment Tree





General term of a GP is $T_n = ar^{n-1}$

Sum of first n terms of G.P:

a. $S_n = \frac{a(r^n - 1)}{r - 1}$ where $r > 1$

b. $S_n = \frac{a(1 - r^n)}{1 - r}$ where $r < 1$

c. $S_n = na$ where $r = 1$

Sum of infinite G.P:

If a G.P. has **infinite terms** and $-1 < r < 1$ or $|x| < 1$,

Sum of infinite G.P is $S_\infty = \frac{a}{1 - r}$.

480 × 469

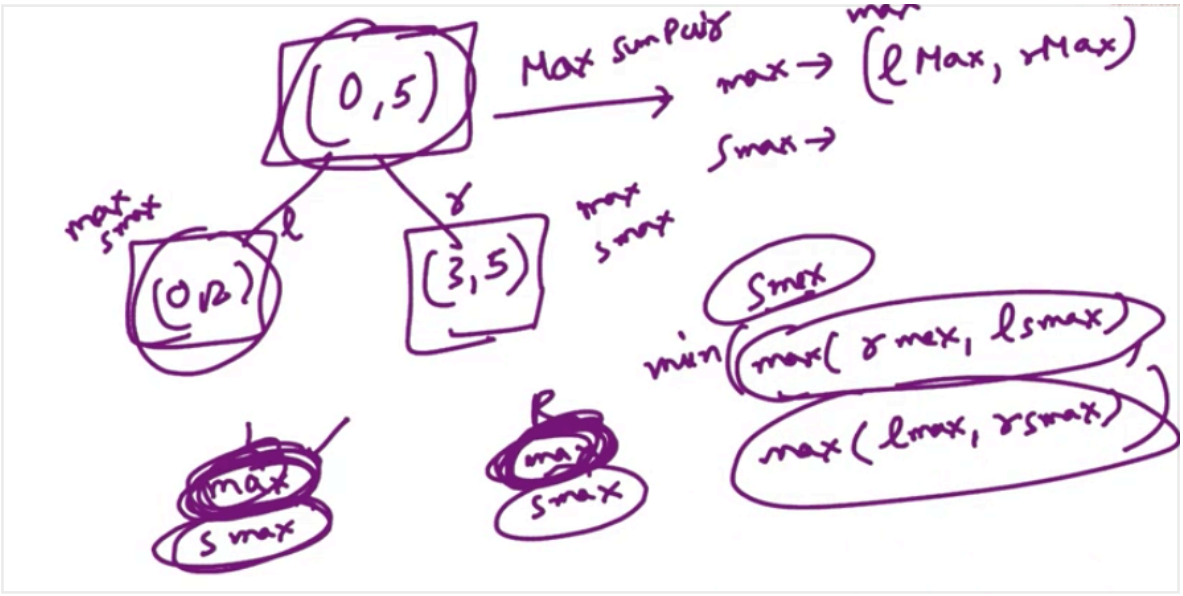
More clean Code- Minimum in SubArray

```cpp
#include <iostream>
#include<climits>
using namespace std;
int arr[100004];
int tree[4*100004];
void build(int l,int r, int st){
    if(l==r){tree[st]=arr[l];return;}
    int mid=((l+r))/2;
    build(l,mid,2*st);
    build(mid+1,r,2*st+1);
    tree[st]=min(tree[2*st],tree[2*st+1]);
}
int query(int l,int r,int x,int y,int st){
    if(l>y || r<x){return INT_MAX;}
    if(l>=x && r<=y){return tree[st];}// the cumulative answer of this full range is needed for final ans
    int mid=((l+r)/2);
    int left_ans=query(l, mid, x, y, 2*st);
    int right_ans=query(mid+1, r, x, y, 2*st+1);
    return min(left_ans,right_ans);
}
void update(int l,int r,int index,int val,int st){
    if(l==r){tree[st]=val;arr[l]=val;return;}
    int mid=((l+r)/2);int lef=st+st;int rit=lef+1;
    if(index>mid){
        update(mid+1, r, index, val,rit);
    }else{
        update(l, mid, index, val,lef);
    }
    tree[st]=min(tree[lef],tree[rit]);
}
```

```cpp
int main() {

    int n,q,a,b;char ch;
    cin>>n>>q;
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    build(0,n-1,1);
    while (q--) {
        cin>>ch>>a>>b;
        if(ch=='q'){
            cout<<query(0,n-1,a-1,b-1,1)<<endl;// notice we bring asked ranged in accordance with zero based indexing
        }else if(ch=='u'){
            update(0,n-1,a-1,b,1); // arr[a-1]=b
        }

    }
    return 0;
}
```

Maximum Pair Sum

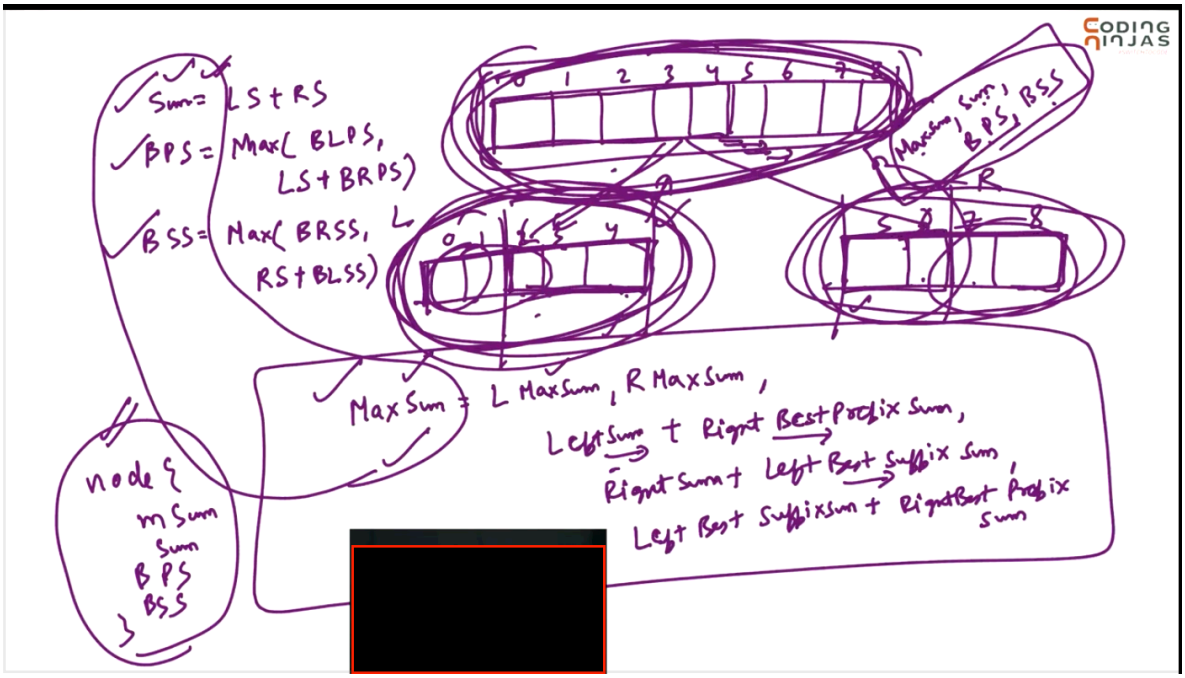
- have to find sum of the max pair within range
- we cant use the two pointer approach here - time problem




```cpp
using namespace std;

struct node{
    int maximum;
    int smaximum;
};

void buildTree(int* arr,node* tree,int treeIndex,int start,int end){

    if(start == end){
        tree[treeIndex].maximum = arr[start];
        tree[treeIndex].smaximum = INT_MIN;
        return;
    }

    int mid = (start+end)/2;
    buildTree(arr,tree,2*treeIndex,start,mid);
    buildTree(arr,tree,2*treeIndex+1,mid+1,end);

    node left = tree[2*treeIndex];
    node right = tree[2*treeIndex+1];
    tree[treeIndex].maximum = max(left.maximum,right.maximum);
    tree[treeIndex].smaximum = min(max(left.maximum,right.smaximum), max(left.smaximum,right.maximum));
    return;
}
```

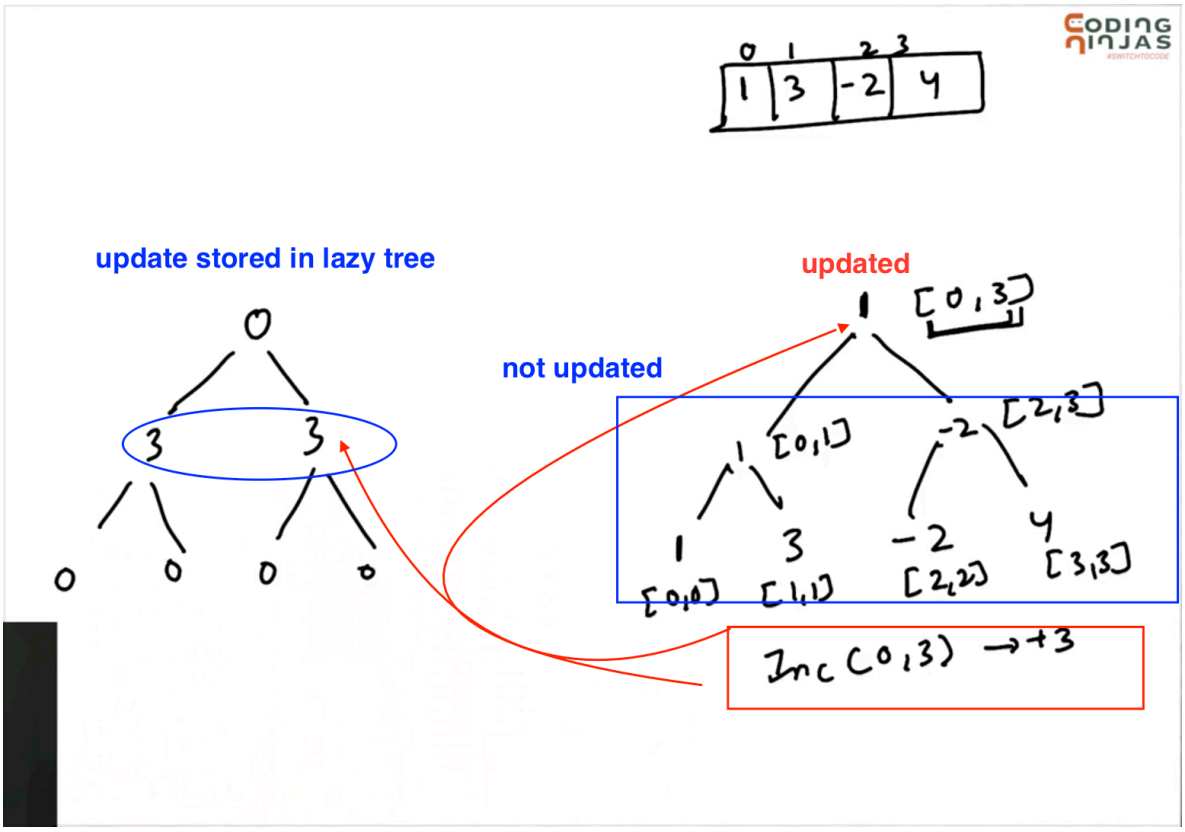
Maximum Sum in a SubArray

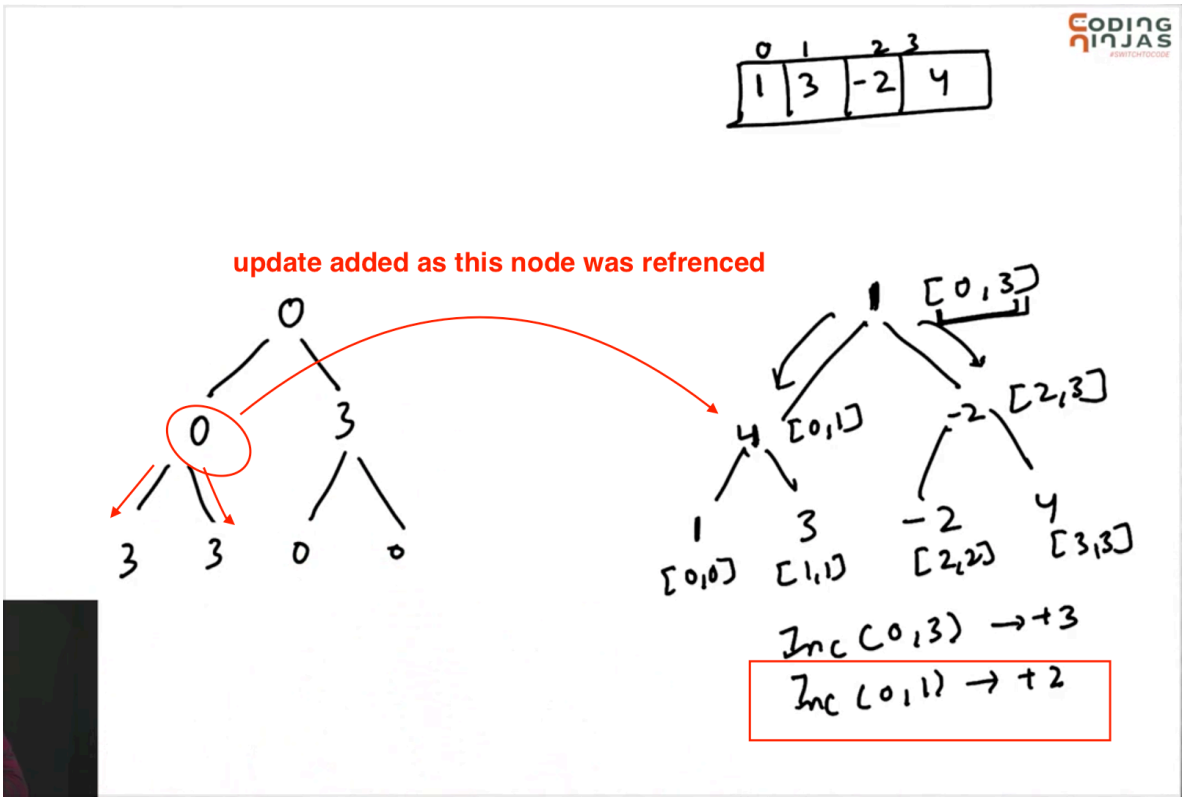
(remember Kadanes ?)

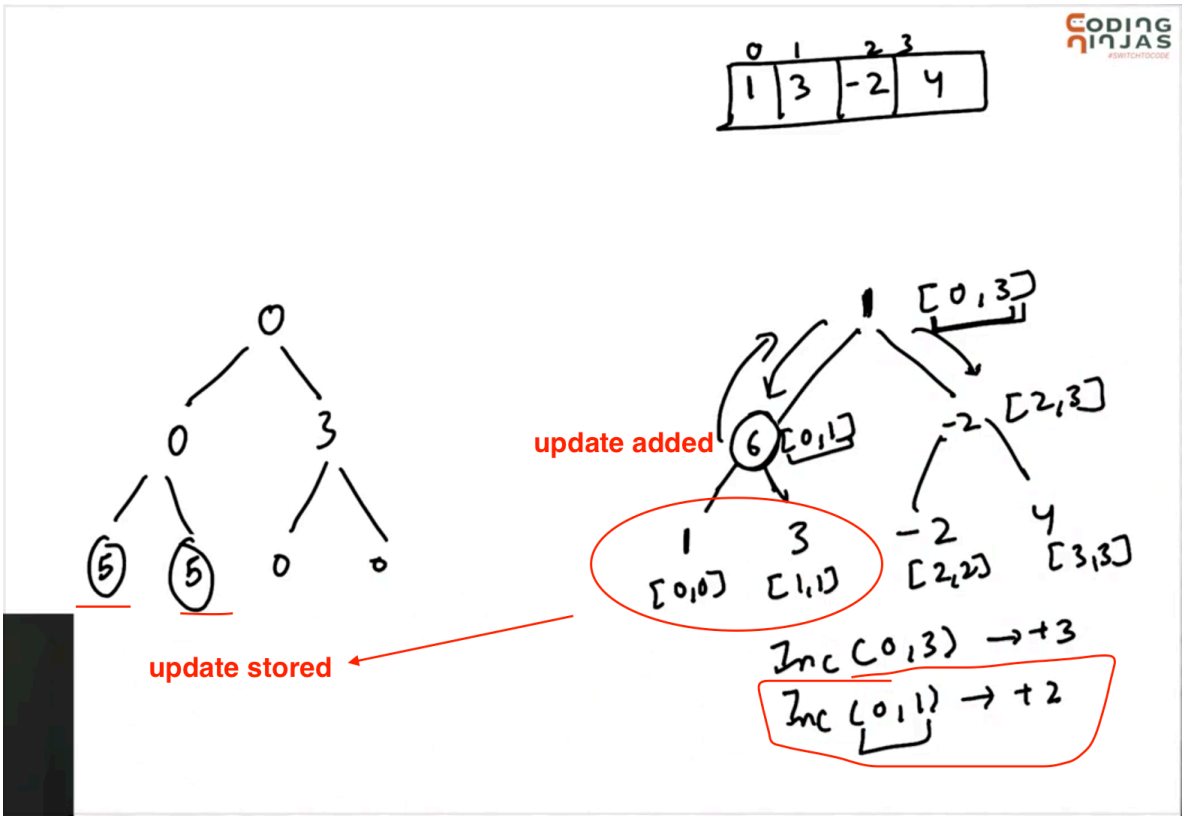


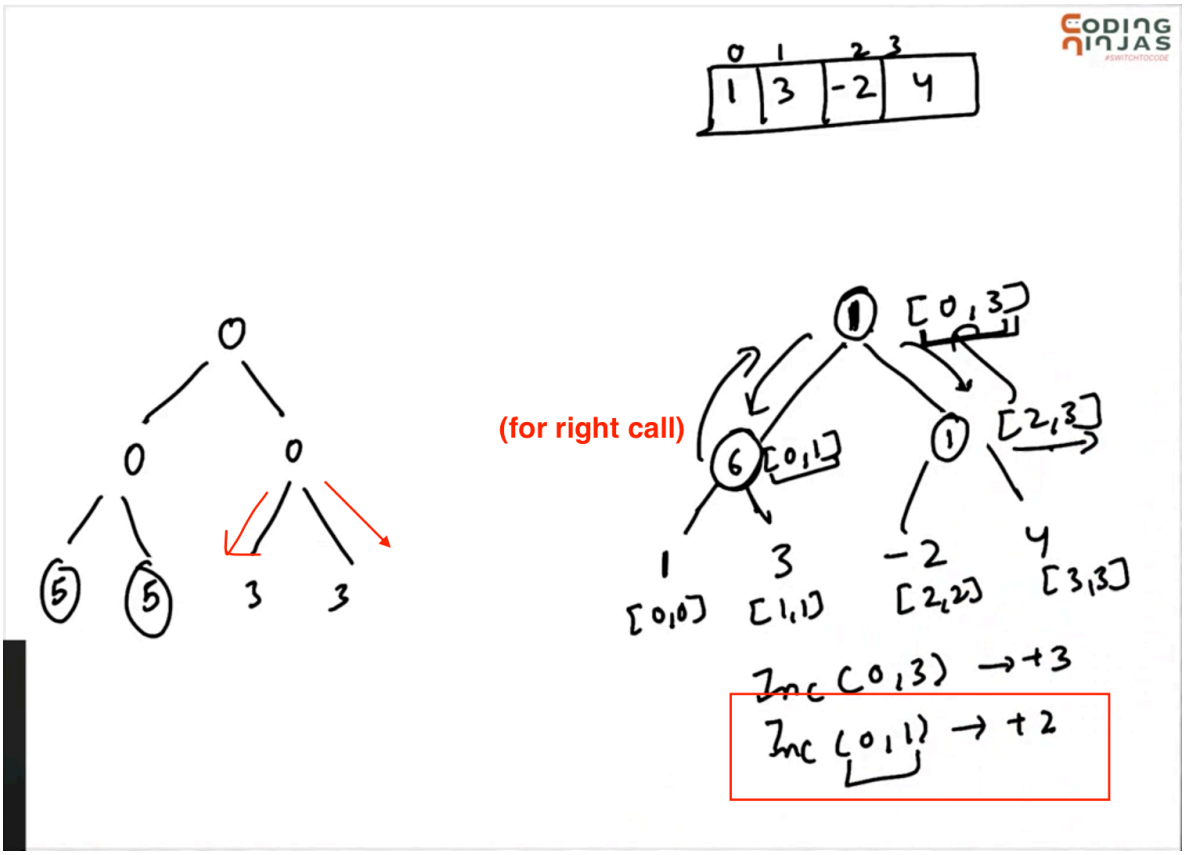
Lazy Propogation

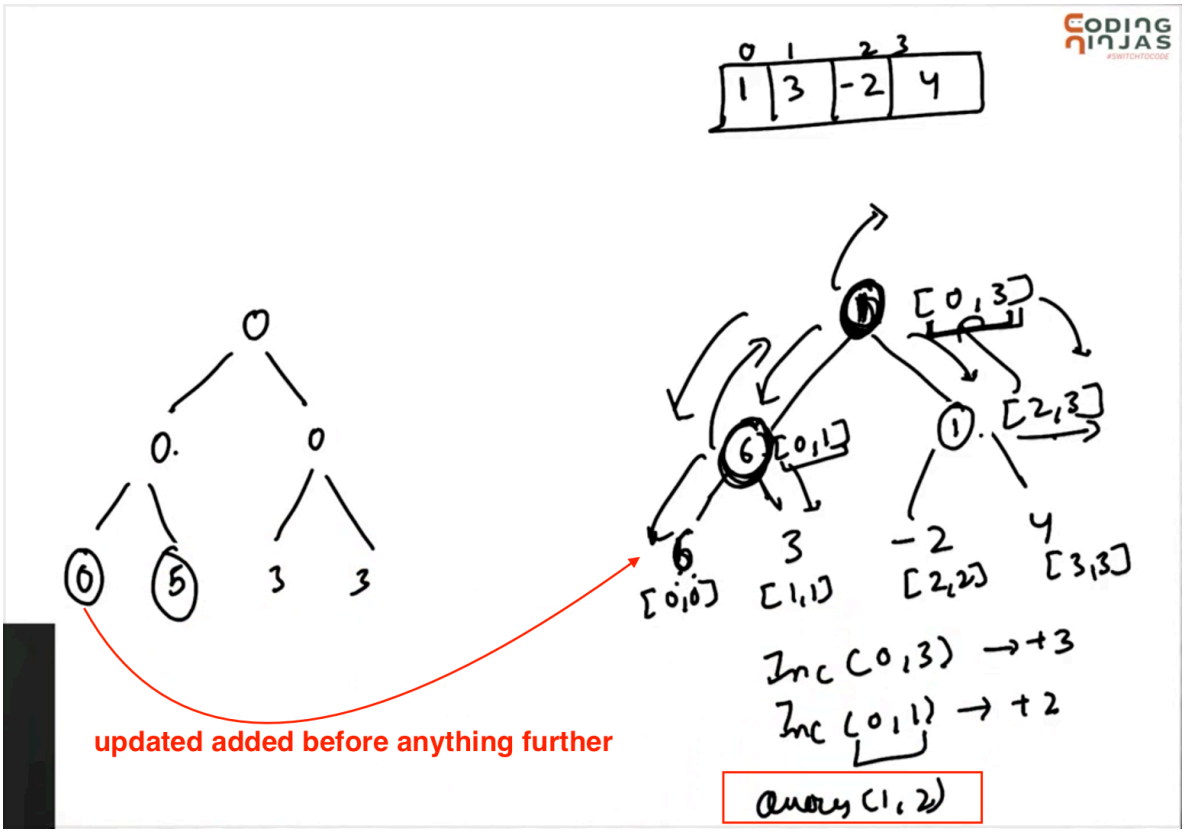
- here updates are applied on intervals
- we make a lazy tree
- we update only when it is used by someone
- we tell its immidiate children in lazy tree- here is update factor -update it when you are refrenced - NOT IN SYNC - so all queries here also keep in mind the lazy tree

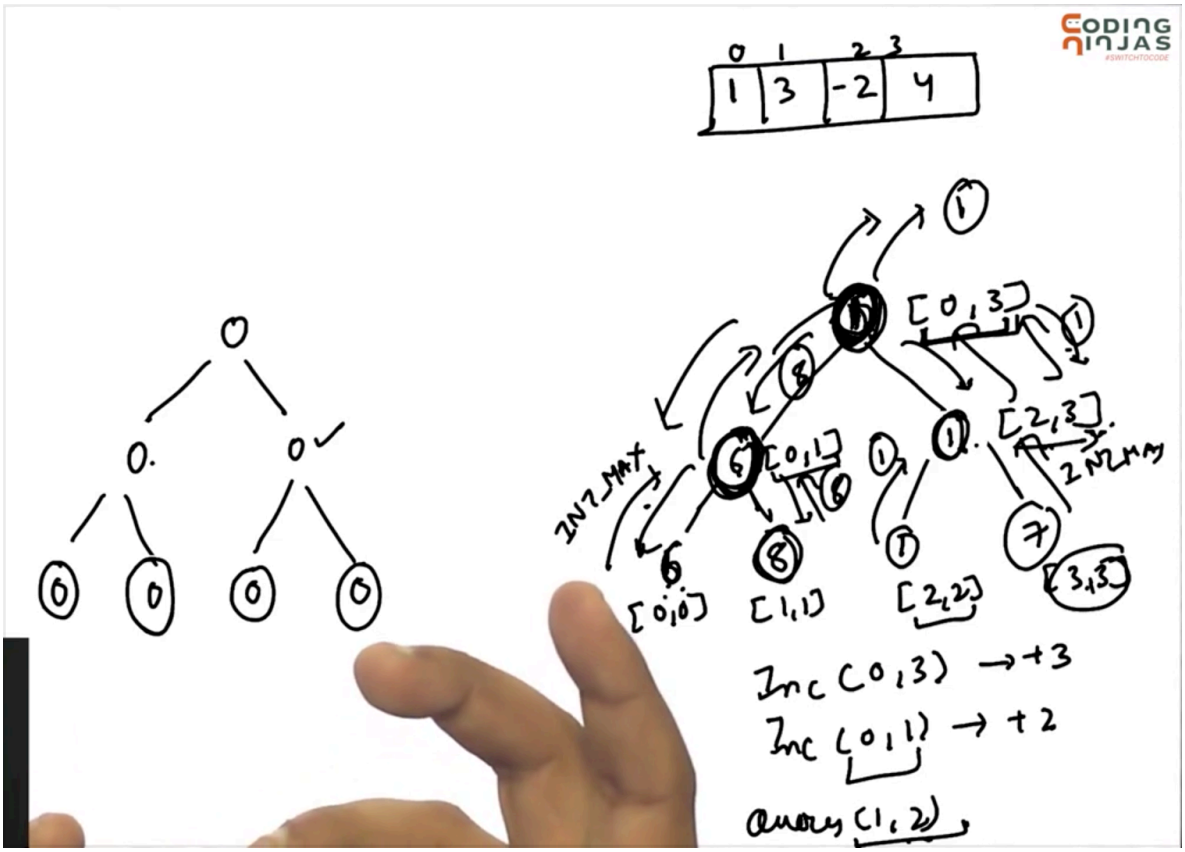

0 1 2 3
1 3 -2 4
update stored in lazy tree
updated
not updated
[0,3]
[0,1]
[2,3]
[0,0]
[1,1]
[2,2]
[3,3]
Inc (0,3) → +3


0 1 2 3
1 3 -2 4
update added as this node was refrenced
[0,3]
[0,1]
[2,3]
[0,0]
[1,1]
[2,2]
[3,3]
Inc (0,3) → +3
Inc (0,1) → +2


0 1 2 3
1 3 -2 4
update added
update stored
[0,3]
[0,1]
[2,3]
[0,0]
[1,1]
[2,2]
[3,3]
Inc (0,3) → +3
Inc (0,1) → +2


0 1 2 3
1 3 -2 4
(for right call)
[0,3]
[0,1]
[2,3]
[0,0]
[1,1]
[2,2]
[3,3]
Inc (0,3) → +3
Inc (0,1) → +2





update function now has to be modfied to handle a range And has a

accompanying lazy tree

- If a node has l and r not eqaual that means its not a leaf node

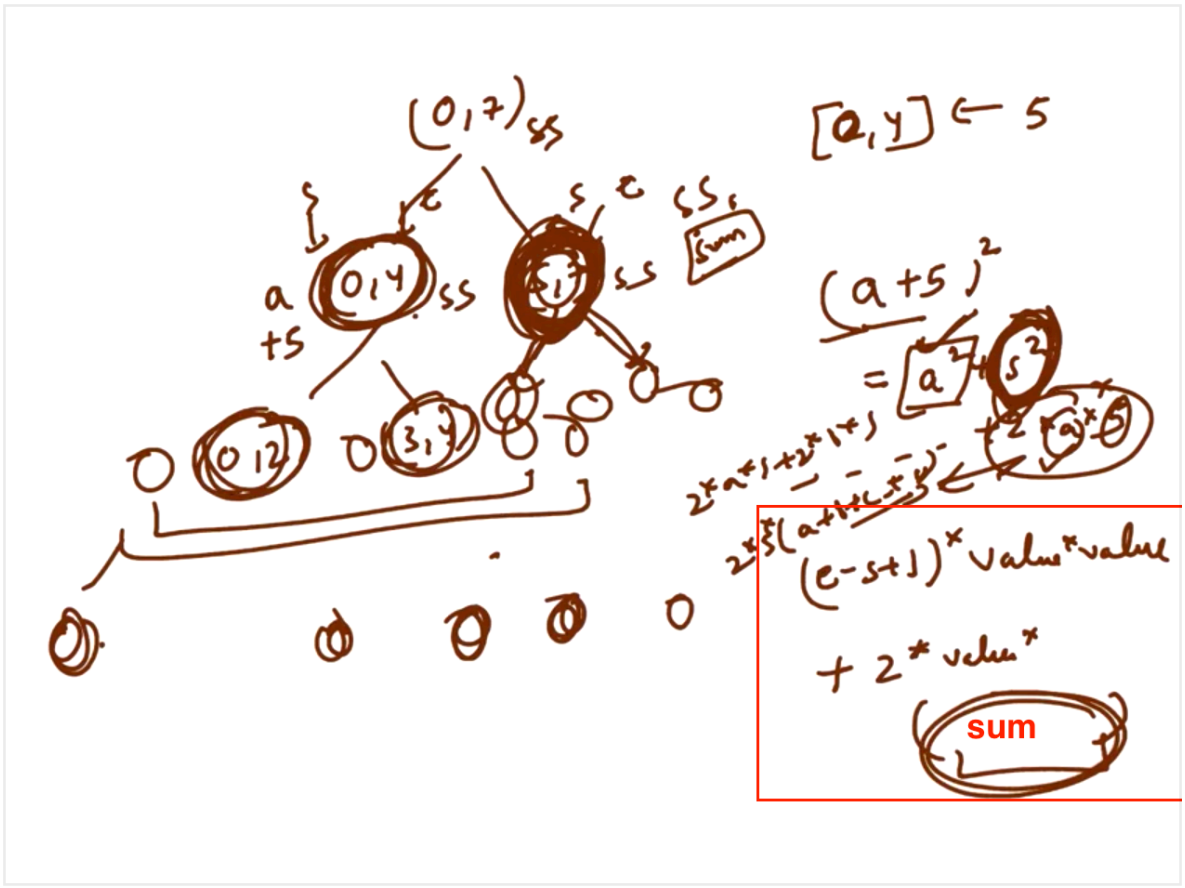
```
17 void updateSegmentTreeLazy(int* tree,int* lazy,int low,int high,int startR,int endR,int currPos,int inc){
18
19     if(low > high){
20         return;
21     }
22
23     if(lazy[currPos] !=0){
24         tree[currPos] += lazy[currPos];
25
26         if(low!=high){
27             lazy[2*currPos] += lazy[currPos];
28             lazy[2*currPos+1] += lazy[currPos];
29         }
30         lazy[currPos] = 0;
31     }
32
33     // No overlap
34     if(startR > high || endR < low){
35         return;
36     }
37
```

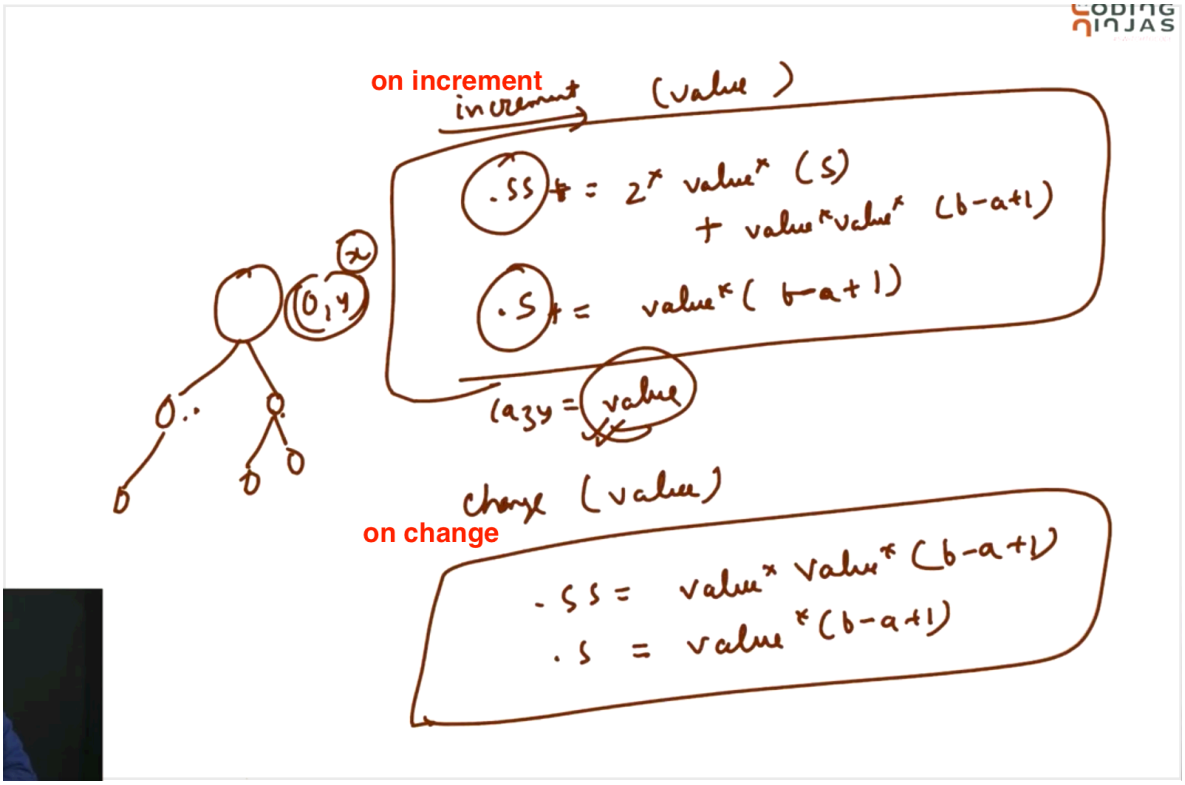
```
38     // Complete Overlap
39
40     if(startR<= low && high <= endR){
41         tree[currPos] += inc;
42         if(low!=high){
43             lazy[2*currPos] += inc;
44             lazy[2*currPos+1] += inc;
45         }
46         return;
47     }
48
49     // Partial Overlap
50
51     int mid = (low+high)/2;
52     updateSegmentTreeLazy(tree,lazy,low,mid,startR,endR,2*currPos,inc);
53     updateSegmentTreeLazy(tree,lazy,mid+1,high,startR,endR,2*currPos+1,inc);
54     tree[currPos] = min(tree[2*currPos],tree[2*currPos+1]);
55 }
56
```

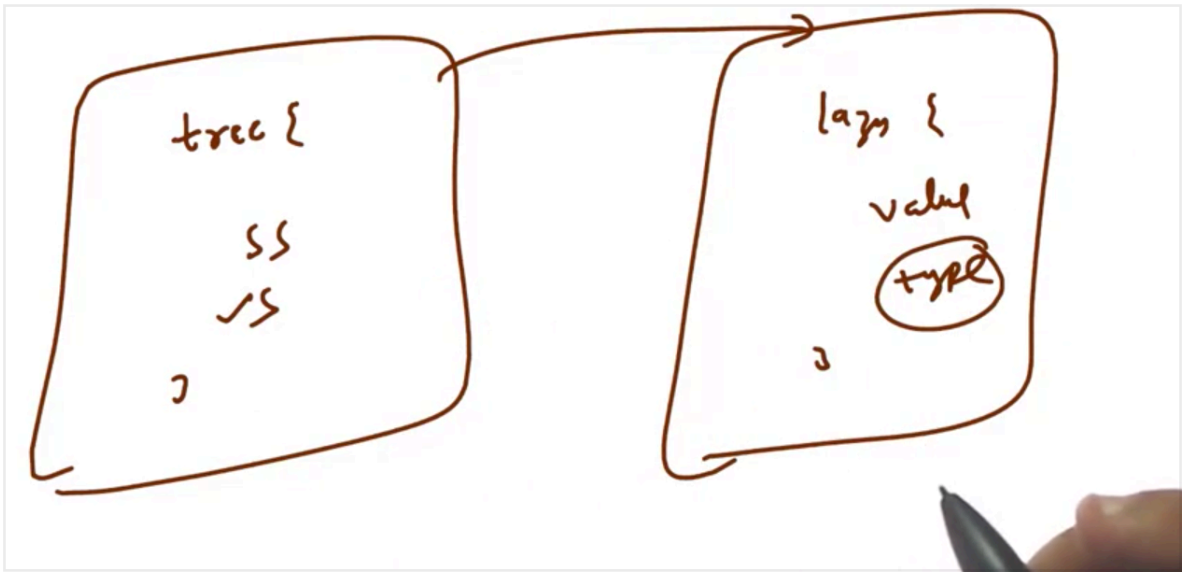
```
void updatelazy(int l,int r,int x,int y  ,int val,int st){
    if(l>r)return;// invalid range l==r would be handled below
    if(lazy[st]!=0){// this node has a pending update
        tree[st]+=lazy[st];
        if(l!=r){// this is not a leaf node -
            //this update has to be propgated to next immidiate children
            lazy[2*st]+=lazy[st];
            lazy[2*st+1]+=lazy[st];
        }
        lazy[st]=0;// no pending update ensured we may continue our work on this node now
    }
    if(l>y || r<x){return;}
    if(l>=x && r<=y){// this full range has received an update
        tree[st]+=val;
        lazy[2*st]+=val;// saving updates for later
        lazy[2*st+1]+=val;// saving updates for later
        return;
    }
    int mid=((l+r)/2);
    updatelazy(l, mid, x, y, val, 2*st);
    updatelazy(mid+1, r, x, y, val, 2*st+1);
    tree[st]=min(tree[2*st],tree[2*st+1]) ;
}
```

SUM OF SQUARES -

- There is interval update as well as you can set all values in an interval TO a specific value


[a,y] ← 5
(a+5)²
sum


on increment
increment (value)
.ss += 2* value* (s)
+ value*value* (b-a+1)
.s += value*(b-a+1)
(a3y = value
change (value)
on change
.ss = value* value* (b-a+1)
.s = value*(b-a+1)



for lazy
0->. no change
1 -> increment
2 -> change