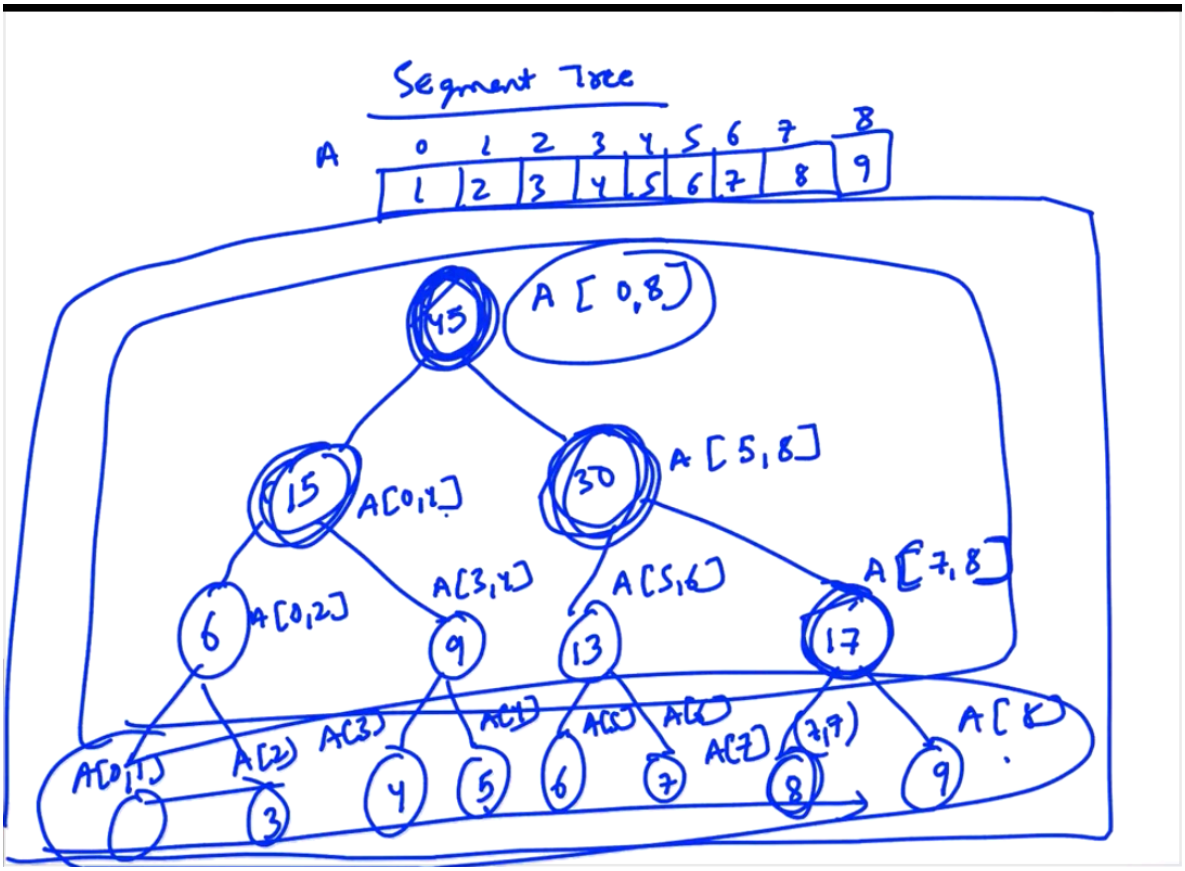
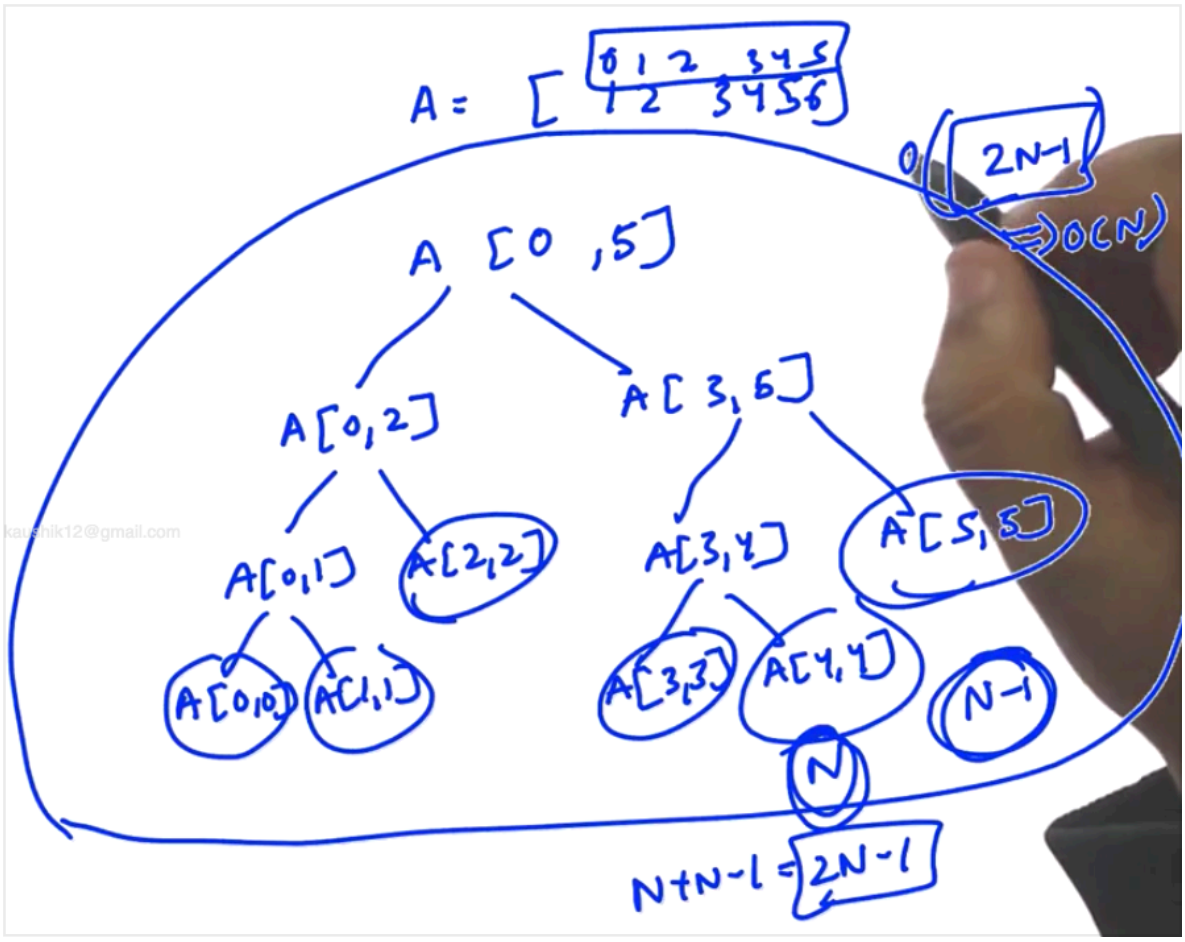
SEGMENT TREES

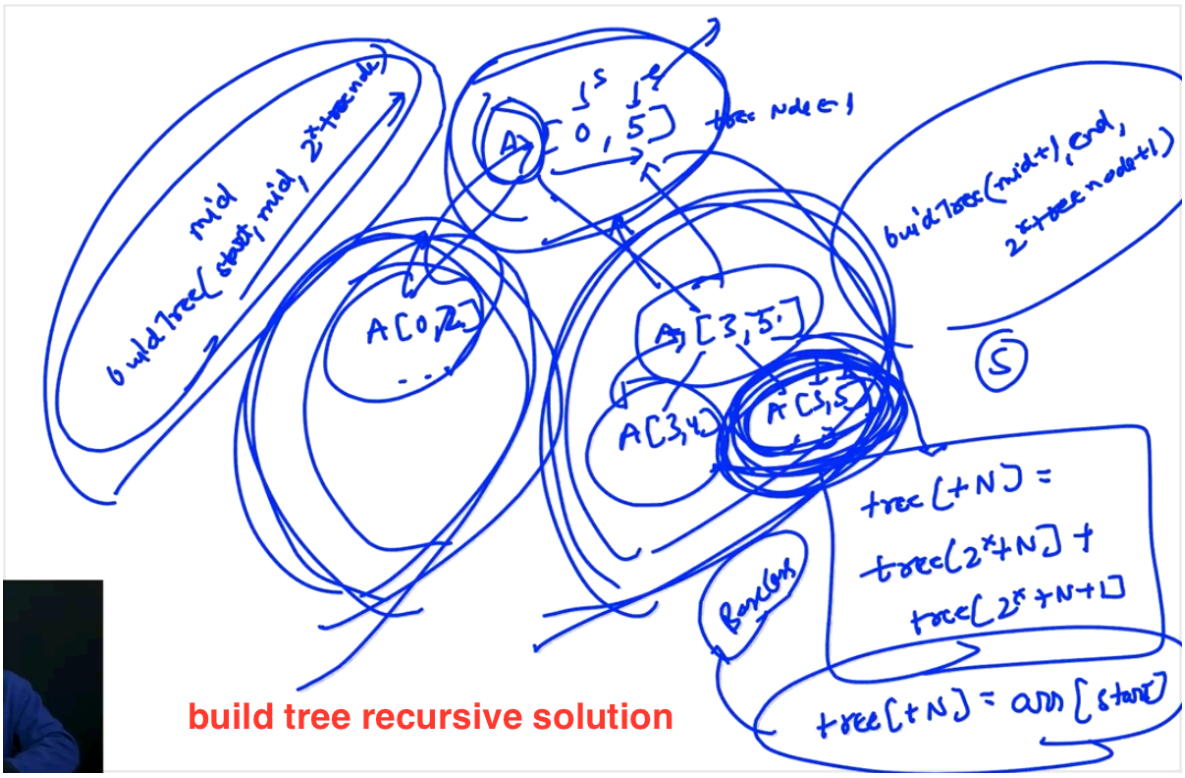
- used when we want output within a range “give me the answer from i to jth index” and this is often accompanied by updates at indexes
- generally used on array problems
- A segment tree updates and queries BOTH in log(n) !
- The leaves of the segment tree are actual indexes of arrray



- 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 3n!




```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;
```