Segment Tree

Concepts & Qns...

Facebook
Instagram → codestorywithMIK

(Twitter) → CSwithMIK

codestorywithMIK →

"No more fear of Segment Tree"

#Motivation

"When you'll realise the benefits of being consistent, you'll be scared to miss even a single day"

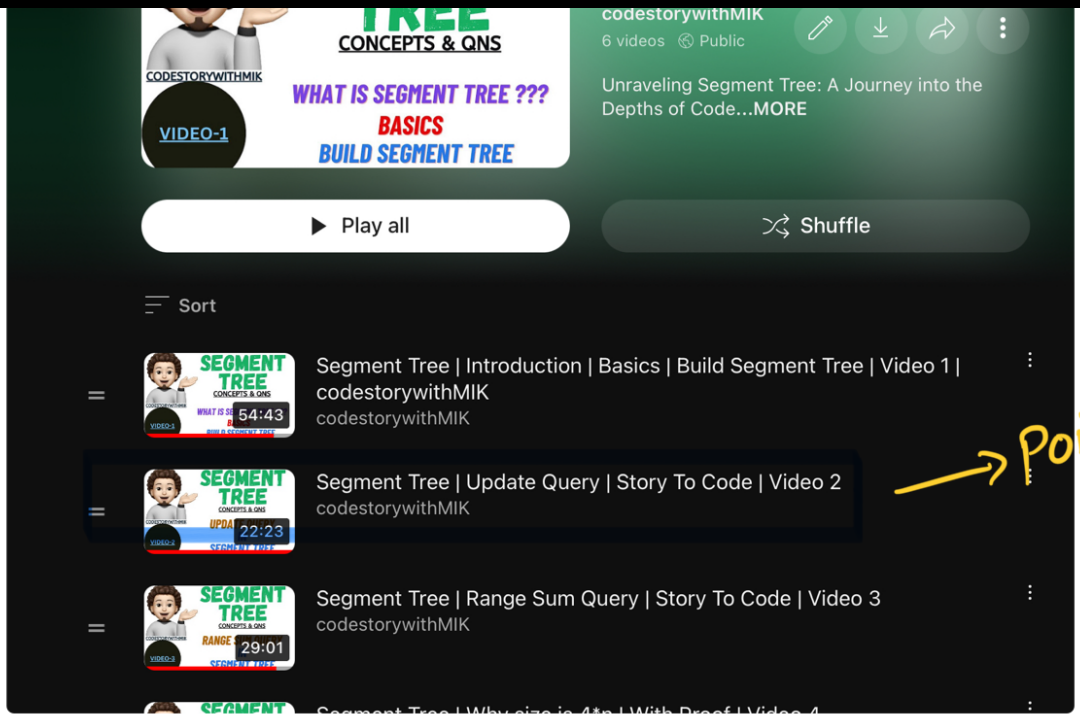
video - 7

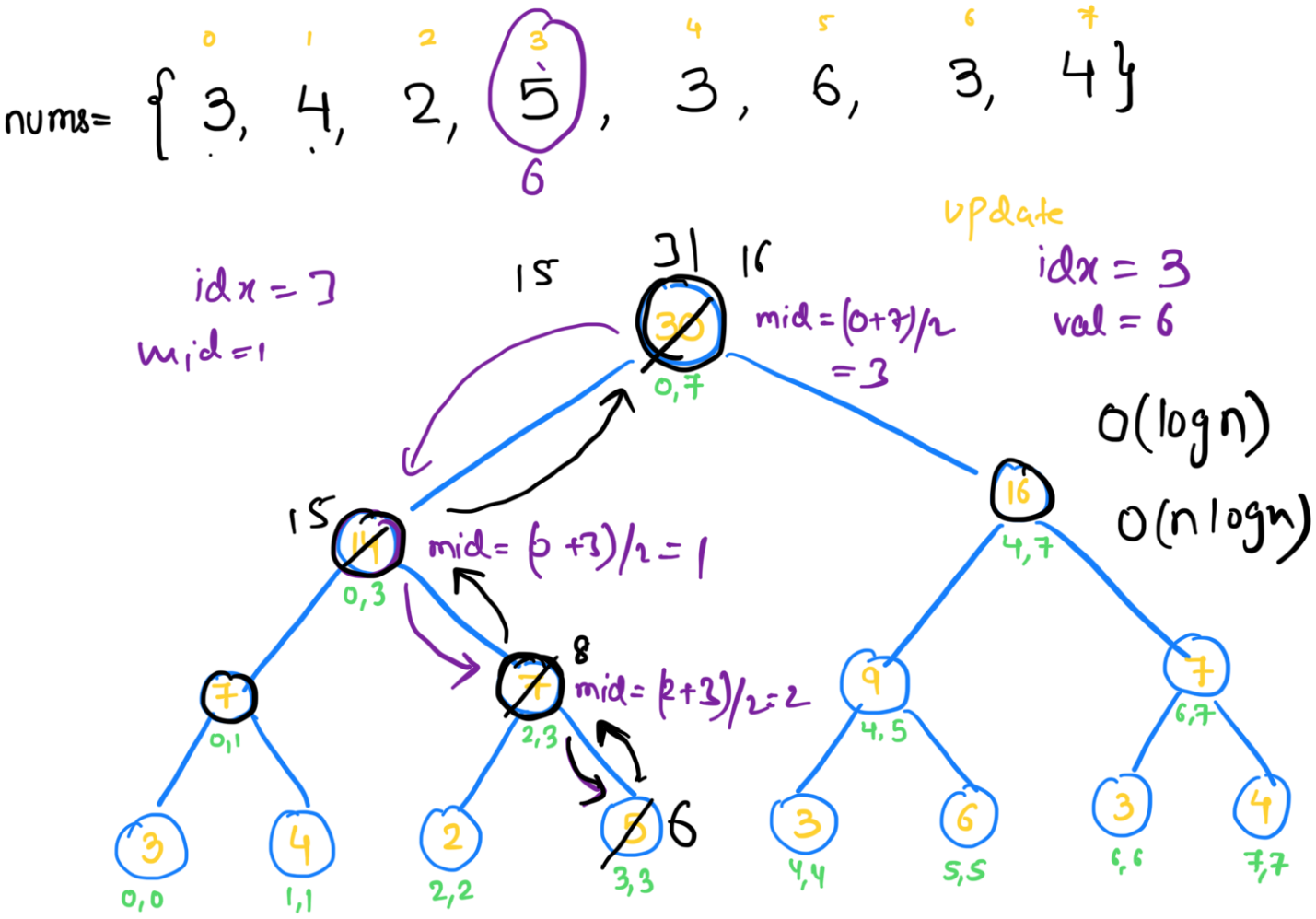
Recap :-

- We understood about segment Tree ? what ? why ? When ?
- buildSegmentTree ←
- Example – Range Sum in an array ←
- Update Query ← (update point) Qn).
- Range Query ←
- Why take 4*n size array ←
- Qn → Query Sum II . ←
  → Range Minimum Query. ←

Range Update (LAZY PROPAGATION)

Video – 2 → update Query...

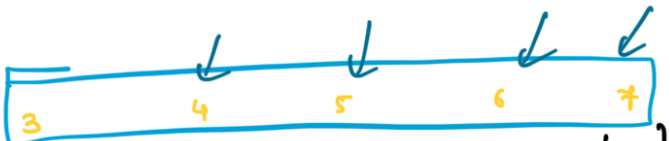

CONCEPTS & QNS
CODESTORYWITHMIK
VIDEO-1
WHAT IS SEGMENT TREE ???
BASICS
BUILD SEGMENT TREE
codestorywithMIK
6 videos Public
Unraveling Segment Tree: A Journey into the Depths of Code...MORE
Play all
Shuffle
Sort
Segment Tree | Introduction | Basics | Build Segment Tree | Video 1 | codestorywithMIK
codestorywithMIK
54:43
Segment Tree | Update Query | Story To Code | Video 2
codestorywithMIK
22:23
Point update.
Segment Tree | Range Sum Query | Story To Code | Video 3
codestorywithMIK
29:01


nums= { 3, 4, 2, 5, 3, 6, 3, 4 }
0 1 2 3 4 5 6 7
6
update
idx = 3
val = 6
idx = 7
mid = 1
15
31
16
mid = (0+7)/2 = 3
0,7
O(log n)
O(n log n)
15
mid = (0+3)/2 = 1
0,3
8
mid = (4+7)/2 = 5
2,3
16
4,7
9
4,5
7
6,7
3
0,0
4
1,1
2
2,2
5
6
3,3
3
4,4
6
5,5
3
6,6
4
7,7

Lazy Propagation.

Range Update optimally

nums [start : end] → add → +2



nums = { 3, 4, ~~2~~ 4, 5, 3, 6, 3, 4 }

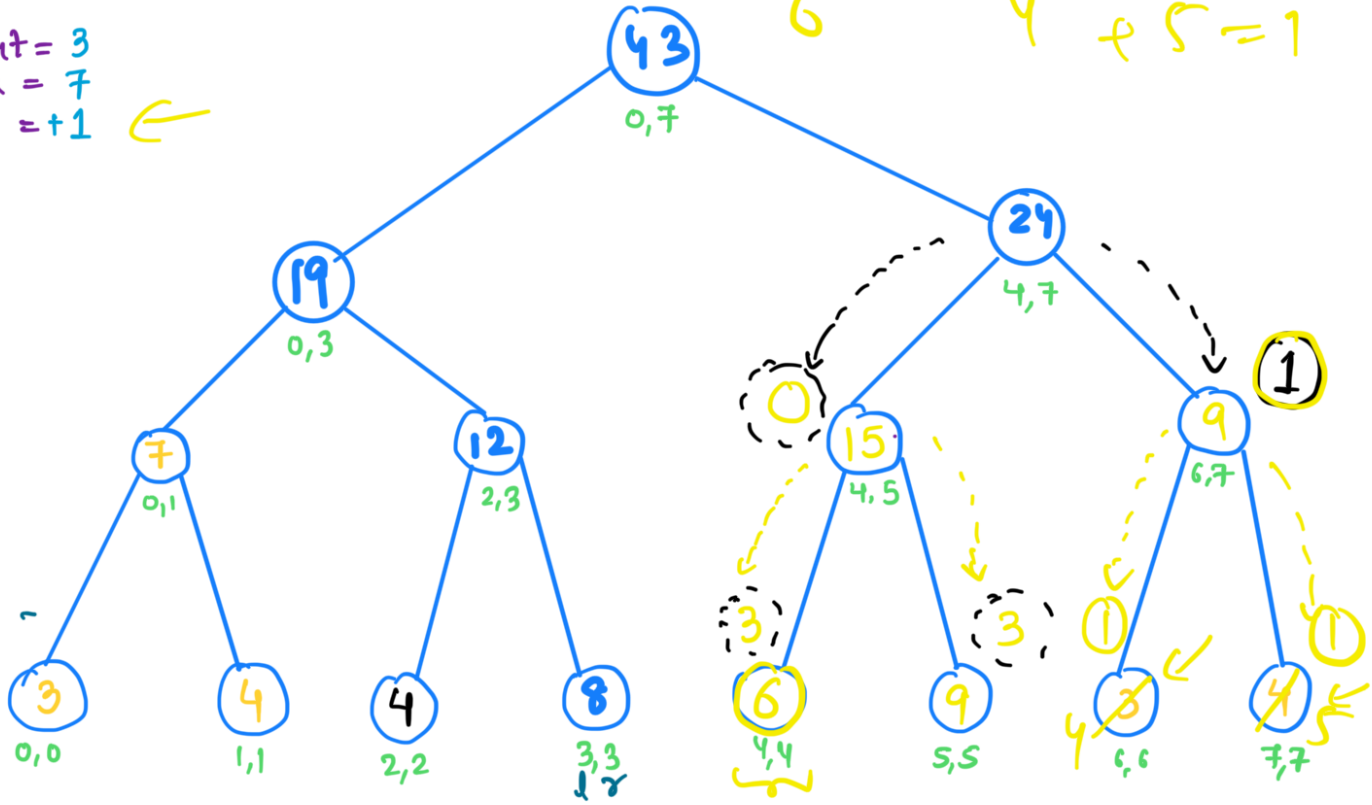
+2 +1 = 6

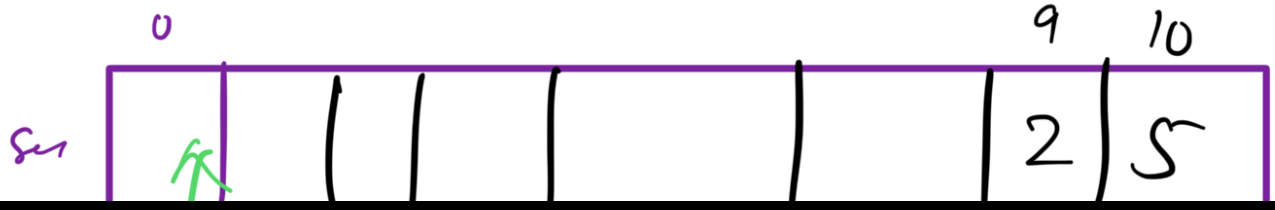
updateRange (2,5)

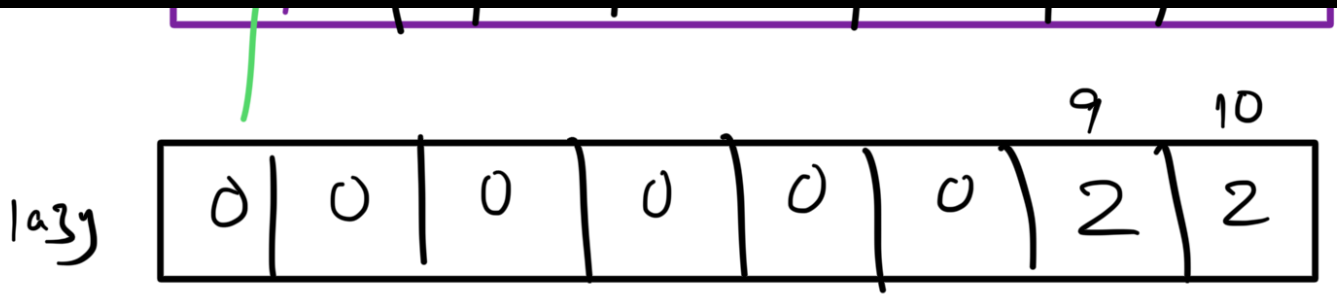
+1 +1

4 + 5 = 1

Start = 3
End = 7
val = +1







Story To Code ...

```
lazy[4*n] = {0};
start =
end   =
val   =
segTree = build:
```

root i, l, r

```
updateRange(start, end, 0, 0, n-1, val, segTree);
```

```
void updateRange(int start, int end, int i, int l, int r,
                 int val, segTree, lazy){

if(lazy[i] != 0){

    segTree[i] += (r-l+1) * lazy[i];

    if(l != r){ // Not a leaf node
```

```
        lazy[2i*1] += lazy[i]; //left
        lazy[2xi+2] += lazy[i]; //right
    }
    lazy[i] = 0;
}

//out of range
if( r < start || l > end || l > r) {
    return;
}
if ( start <= l && end >= r) {
    segTree[i] += (r-l+1) * val;
    if( l != r) {
        lazy[2i+1] += val; //left
        lazy[2i+2] += val; //righ
    }
    return;
}
```