Segment Trees -1

Question: Range Sum Queries and updates

	0	1	2	3	4
A =	1	3	8	10	7

1) Range Sum: [l, r] => Sum of the range [l, r]

2) Update Query: [ind, val] => A[ind] = val

Example

sum(1:3) = 3+8+10 = 19

update [2, 8] =

sum(1:3) = 3+8+10 = 21

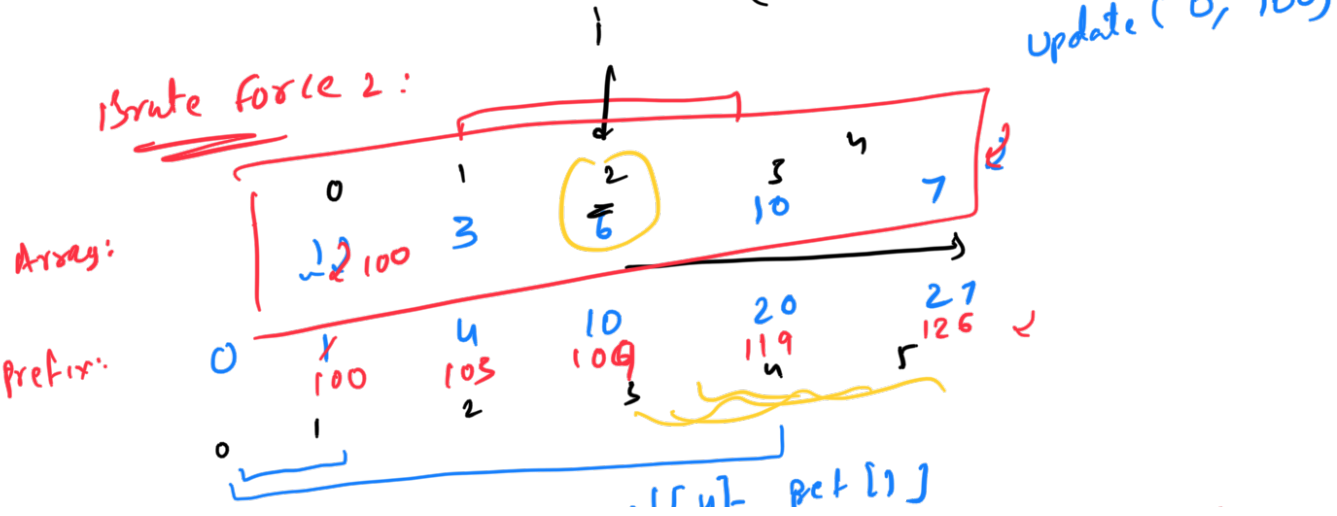
Brute force 1:

sum(l, r) = Iterate over the range and get the sum O(n)

update [ind, val] = O(1)

(i.....n)

Brute force 2:



Sum(1:3) ⇒ pref[3] − pref[1] ⇒ O(1)

Sum(l:r) ⇒ pref[r+1] − pref[l]

Range Query: O(1)

Update Query: O(n)

Approach1

Range O(n)

Update O(1)

More Update

Approach2

O(1)

O(n)

More Range Query

10^5

Square Root Decomposition

Divide the array into . blocks of same size

N = 15

K = 3

Sum =

→ Size of each Block = $\frac{K}{}$ → (3)

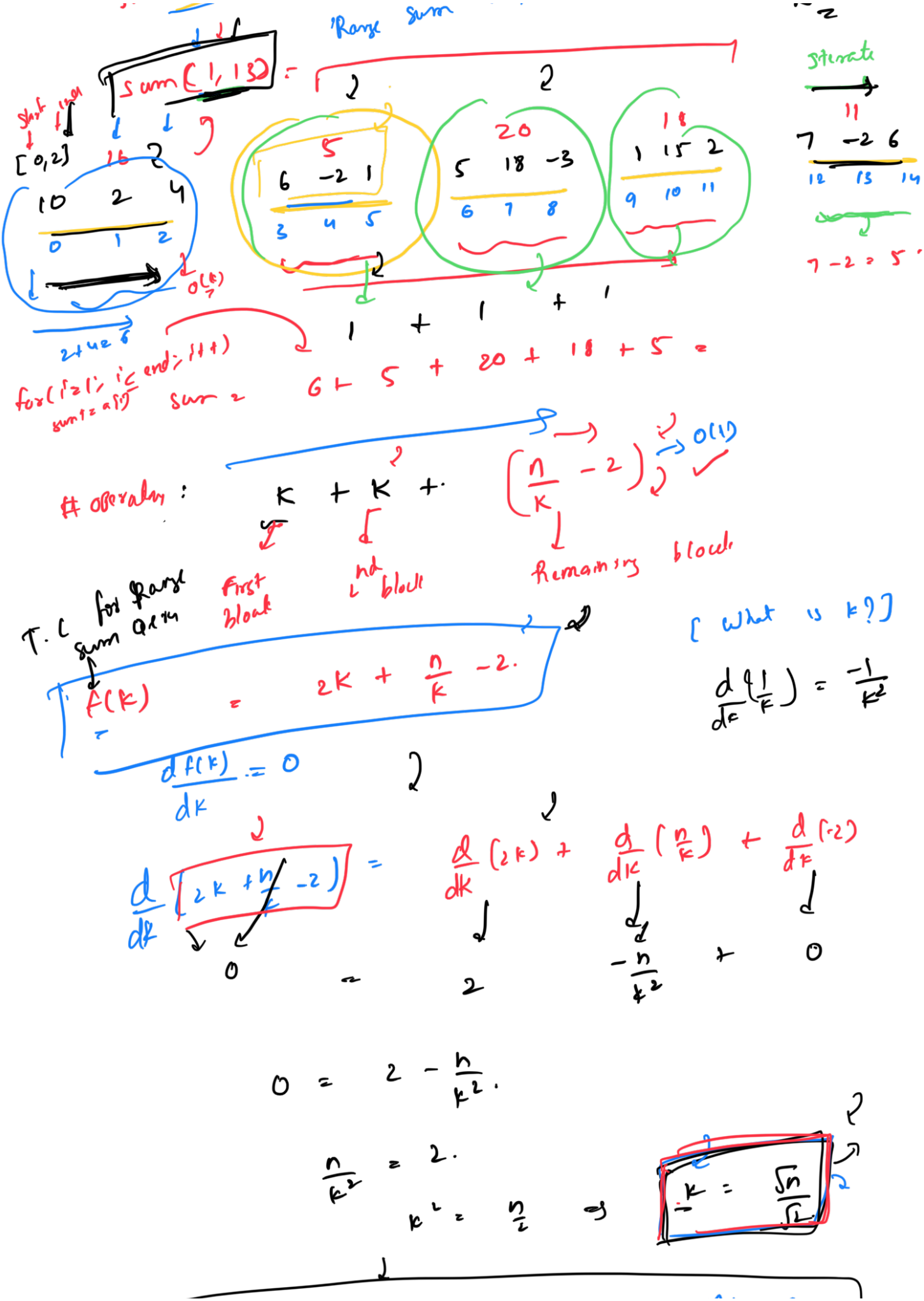
→ No. of blocks = $\frac{N}{K}$

sum[] = → N/K size

Sum(3:5) ⇒ Ans = 5 → O(1)

Query

N = 15


Range sum
sum(1, 13) :=
start index
[0, 2]
10 2 4
0 1 2
O(k)
2+42
5
6 -2 1
3 4 5
20
5 18 -3
6 7 8
18
1 15 2
9 10 11
for(i=1; i<end; i++)
sum = a[i]
sum = 6 + 5 + 20 + 18 + 5 =
1 + 1 + 1
strategy
11
7 -2 6
12 13 14
7 - 2 = 5
operations :
K + K + .
(n/K - 2)
O(1)
First block
nd block
Remaining blocks
T.C for Range sum Qn
f(K) = 2K + n/K - 2.
[What is k?]
d/dK (1/K) = -1/K²
df(K)/dK = 0
d/dK [2K + n/K - 2] = d/dK (2K) + d/dK (n/K) + d/dK (-2)
0 = 2 - n/K².
n/K² = 2.
K² = n/2
K = √n/√2

$\frac{dx}{dt} = 0$ ss $\frac{d^2x}{dt^2} > 0 \rightarrow$ Minima

$\frac{d^2x}{dt^2} < 0 \rightarrow$ Maximo.

$\frac{d}{dk} f(k) = 2 - \frac{n}{k^2}$

$\frac{d^2}{dk^2} f(k) = \frac{d}{dk}(2) - \frac{d}{dk}\left(\frac{n}{k^2}\right)$

$- \left(\frac{-2n}{k^3}\right) = \frac{2n}{k^3}$

$\frac{d^2}{dk^2} f(k) = 0$ ⇒ $\frac{2n}{k^3}$ ⇒ $n > 0$, $k > 0$

$k = \frac{\sqrt{n}}{\sqrt{2}}$ ⇒ $k = \frac{\sqrt{n}}{\sqrt{2}}$ ✓

T.C: $k + \frac{k}{2} + \frac{n}{k}$

$\frac{n}{\sqrt{n}} = \sqrt{n}$

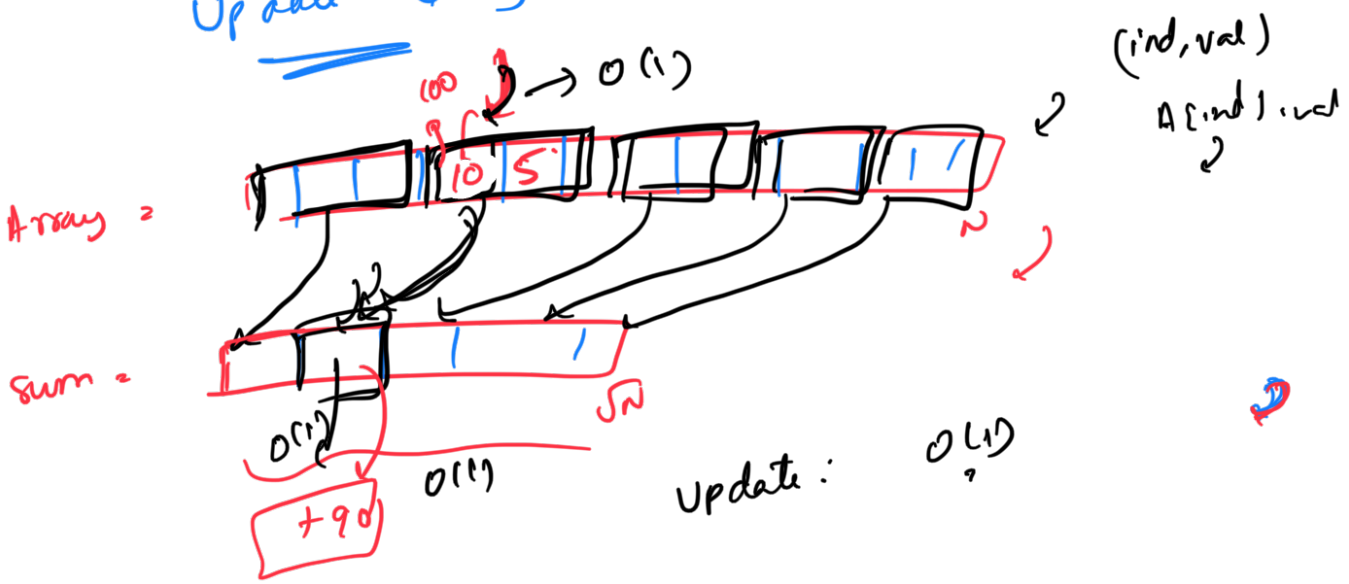
$\frac{\sqrt{n}}{\sqrt{2}} + \frac{\sqrt{n}}{\sqrt{2}} + \frac{n}{\sqrt{n}} \cdot \sqrt{2}$

$= O(\sqrt{n} + \sqrt{n} + \sqrt{n})$

T.C: $O(\sqrt{n})$

$k = \sqrt{N}$

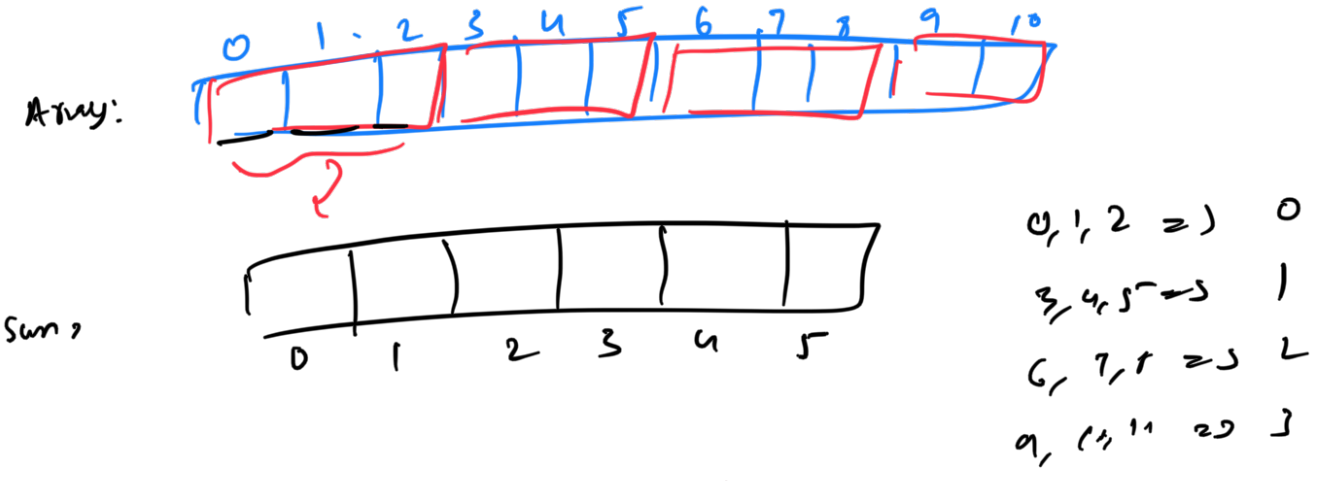
Update Query :



Sq Root Decomposition

T.C : for Range sum : $O(\sqrt{N})$

Update Query : $O(1)$



Segment Tree

Range Sum Query : $O(\log n)$

update Query : $O(\log n)$

log(n) ?

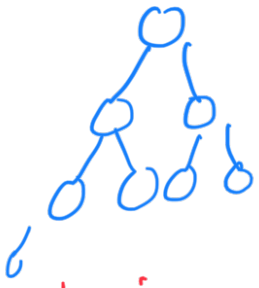
→ Binary Search

→ B.S.T

→ Divide & conquer

→ Divide problem into smaller problems and solve them

→ Divide array into 2 equal halfs



→ Representing Segment tree using an array

parent : i

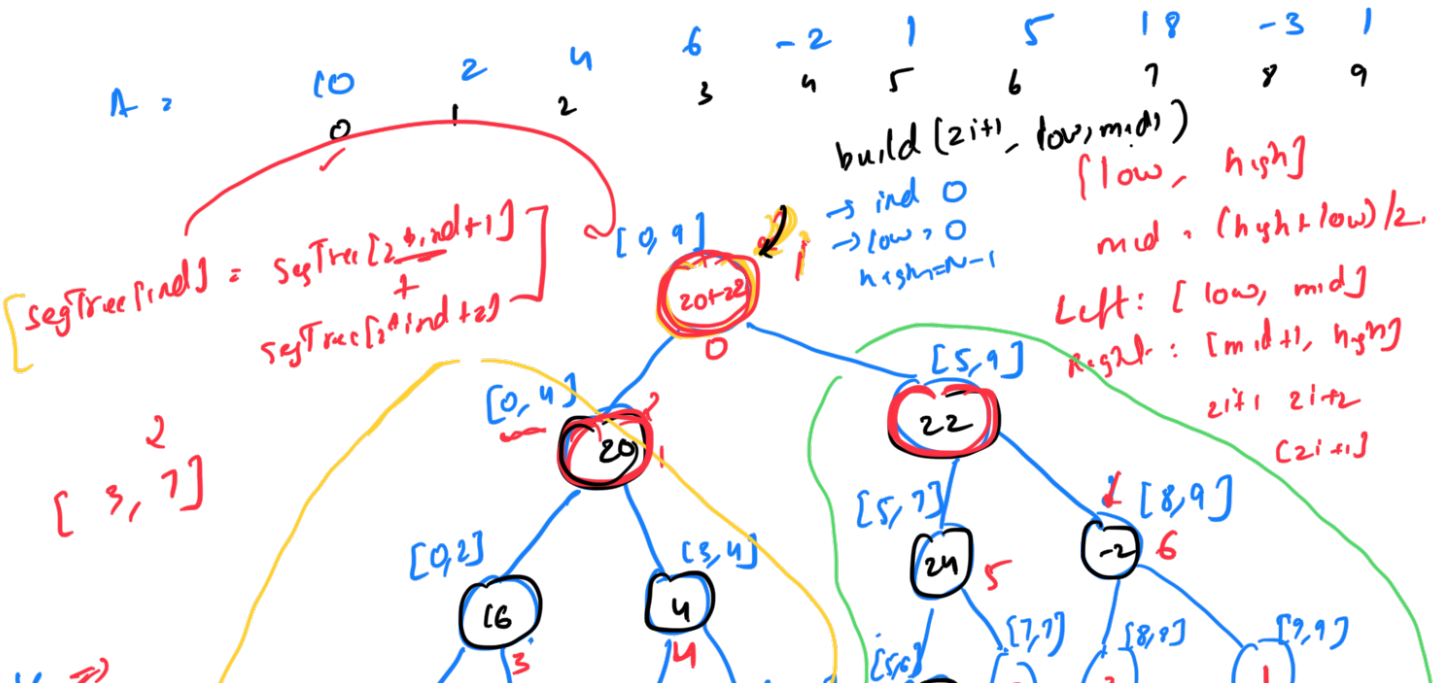
children : 2i+1, 2i+2.

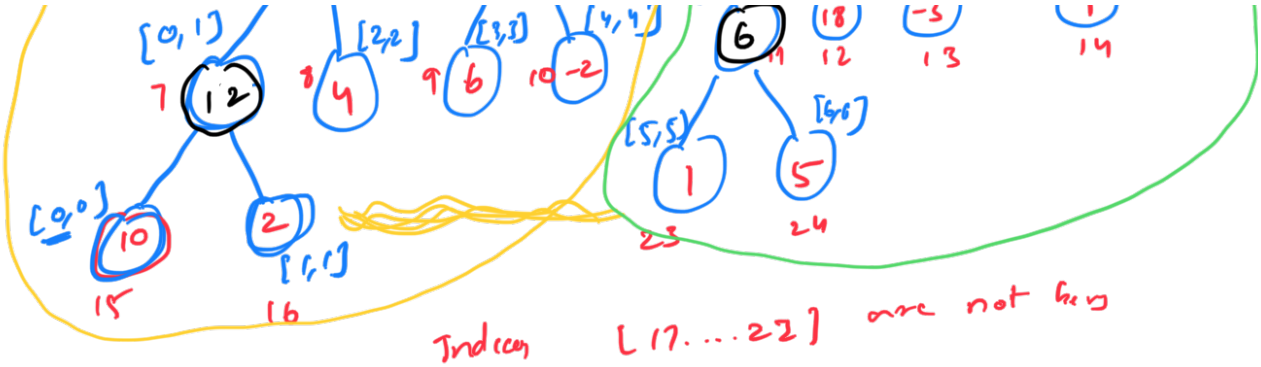
child = i

parent = (i-1)/2

Segment Tree

N = 10





Indices [17....22] are not leaf nodes of segment tree are array elements

→ Leaf Node

→ [Use the existing ranges to efficiently compute answers of all possible ranges.]

Recursion: → [Segment Tree Array]

1) build(i, low, high) builds a segment tree from the array [low..... high] and stores the sum of [l....r] at index 'i'

[l,r]

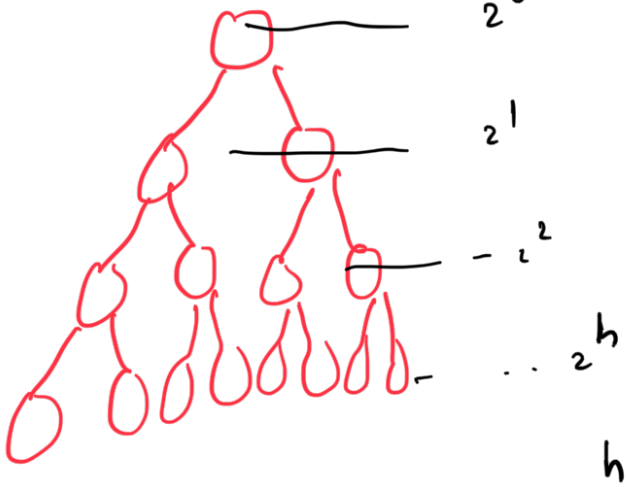
```
int seg[];
void build ( ind, low, high) {
    if(low == high){
        seg[ind] = A[low];
    }
    mid = (low + high)/2;
    build ( 2*ind+1, low, mid);
    build ( 2*ind+2, mid+1, high);
    seg[ind] = (seg[2*ind+1] + seg[2*ind+2]);
                min( , ) .
                max( , )
    return;
}
build ( 0, 0, N-1 );
```

Min
2

Time Complexity:

T.C: O(No. of Nodes in Segment Tree)

What is No. of nodes! N?



Perfect BT

(2N-1) nodes

N leaf

$2^0 + 2^1 + 2^2 \ldots 2^h$

$a\frac{(r^n-1)}{r-1} = 1\frac{(2^{h+1}-1)}{2-1}$

No of leaf Nodes = $2^h = N$

Total Node = $2^0 + 2^1 + 2^2 \ldots 2^h = 2^{h+1} - 1$

$= 2^h \cdot 2 - 1$

$= 2N - 1$

No. of nodes in Seg Tree = (2N-1)

T.C: O(N)

Space Complexity

Size of Segment Tree array? 2

seg [2N-1]

Left, Right

Update flag

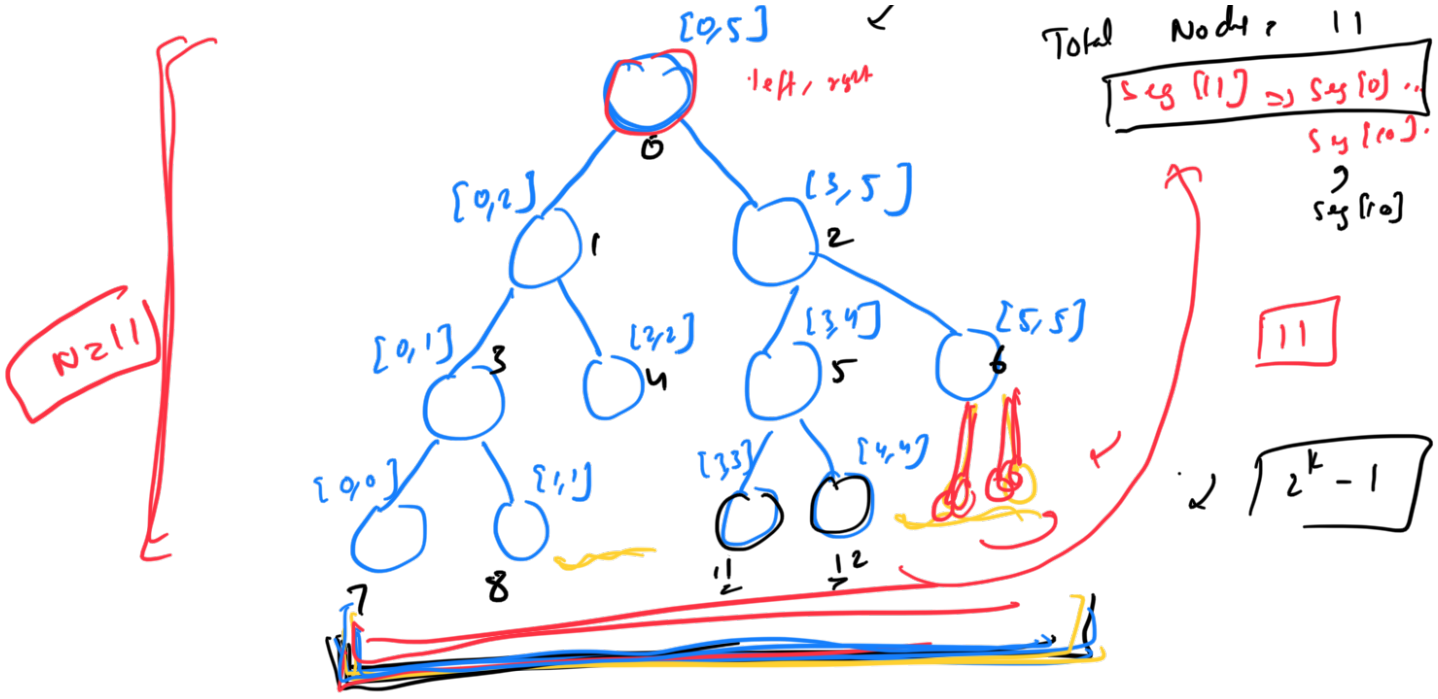
2i+1 2i+2

N=6 → $2^6 - 1 = 63$

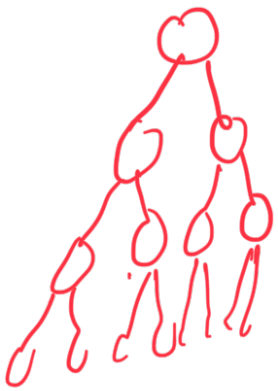
N = 6

16

?



$(2^k - 1)$



N = 7 is $2^3 - 1$

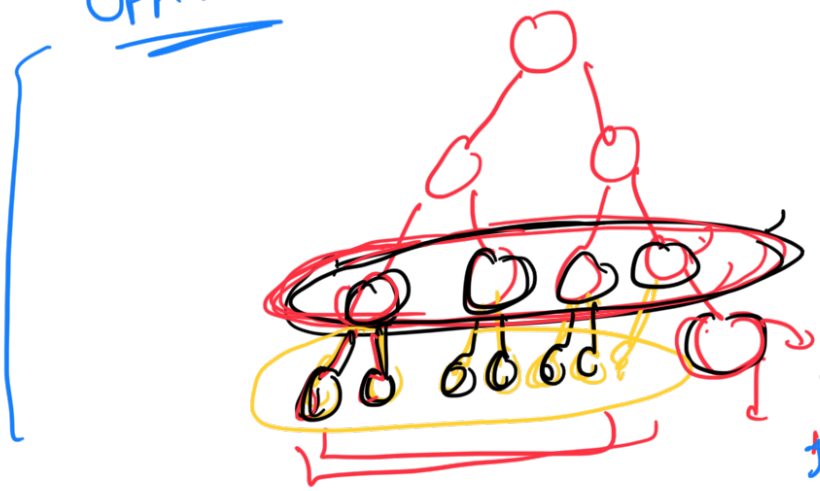
2 $2^h - 1$

→ We should declare an array size which is the nearest power of 2

P.N

$\frac{n}{2}$, $\frac{n}{4}$

Upper bound.



→ N nodes

↓ need space for this.

Array Elements => Leafs

=> N leafs

=> Wasting space - 2·N children

Leafs = N

(2N-1) is the total Nodes

Array: (2N-1) + 2N

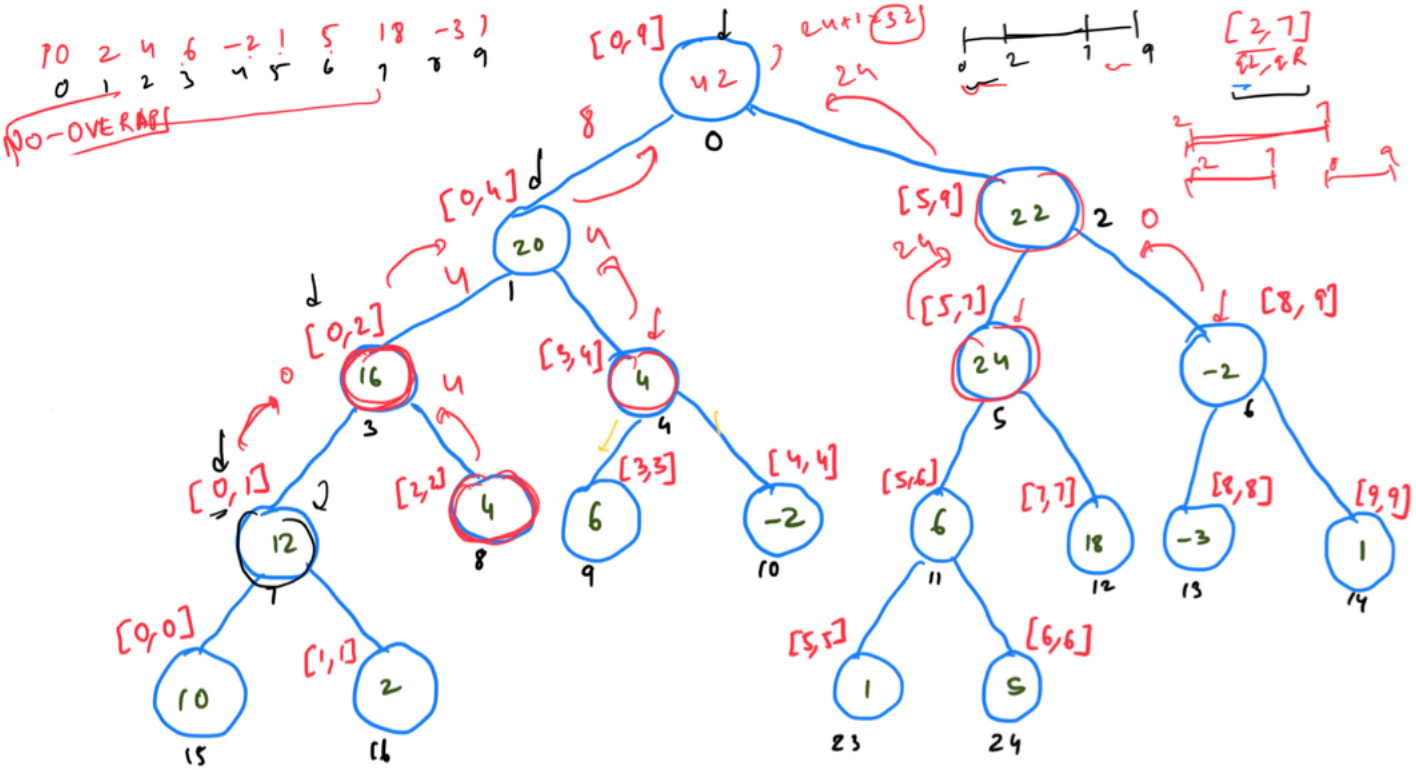
= 4N

N = 10^5 - 10^6

4×10^5, 4×10^6

S.C: O(4N) ≈ O(N)

Range Queries

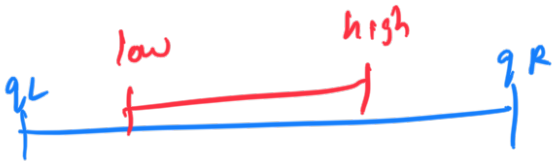

10 2 4 6 -2 1 5 18 -3 7
0 1 2 3 4 5 6 7 8 9
NO-OVERLAP
[0,9]
42
0
8
24
[0,4]
20
1
[5,9]
22
2
0
[0,2]
16
3
[3,4]
4
4
[5,7]
24
5
[8,9]
-2
6
[0,1]
12
7
[2,2]
4
8
[3,3]
6
9
[4,4]
-2
10
[5,6]
6
11
[7,7]
18
12
[8,8]
-3
13
[9,9]
1
14
[0,0]
10
15
[1,1]
2
16
[5,5]
1
23
[6,6]
5
24

Case 1: No Overlap

if(qL > high || qR < low)

return 0;

Case 2: Complete Overlap



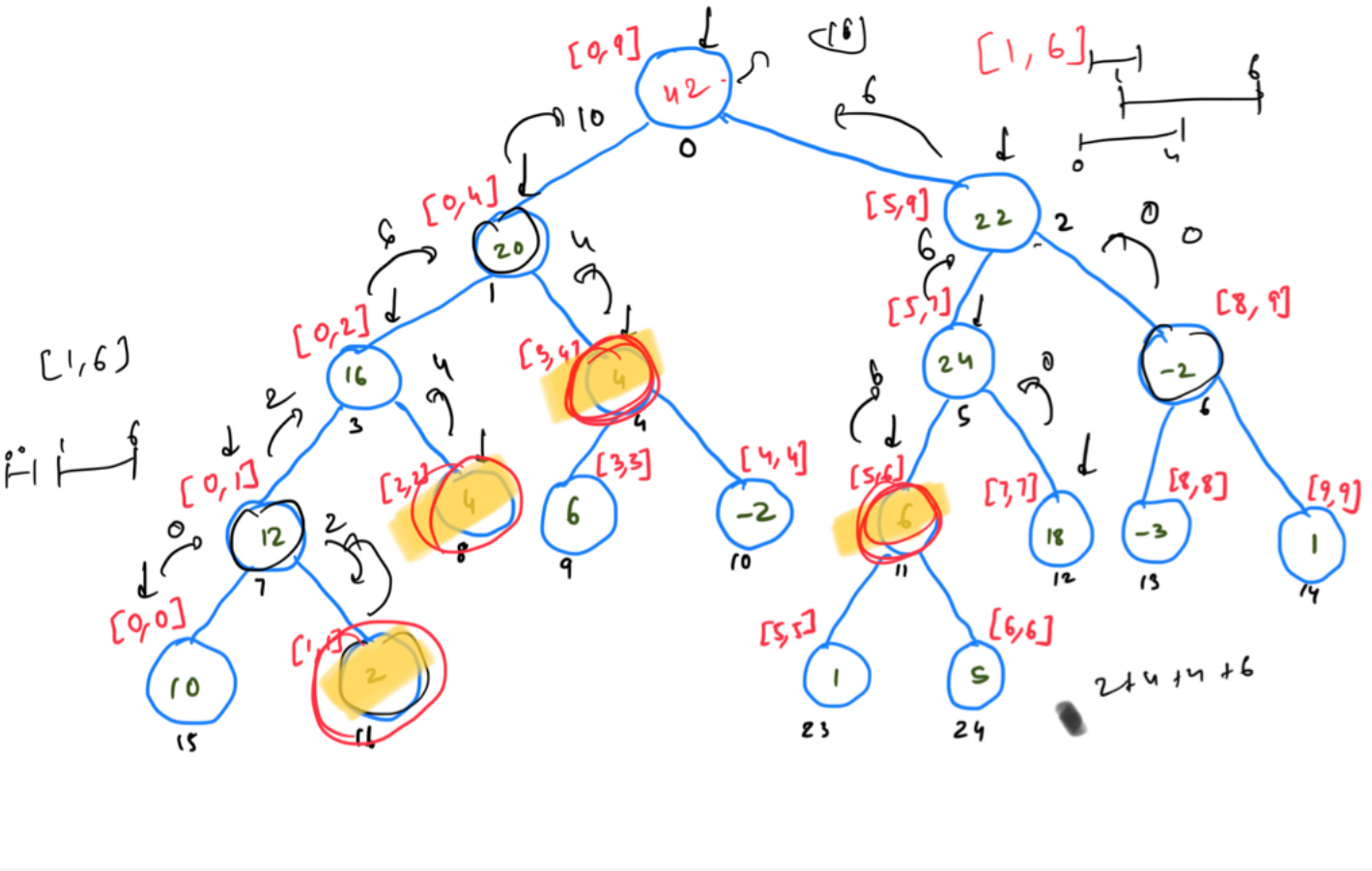
qL,

if(qL ≤ low && high ≤ qR)

return value of current Node

Case 3: Partial overlap

Recurring for Left child & Right child


[0,9]
42
0
[1,6]
[0,4]
20
1
[5,9]
22
2
[0,2]
16
3
[3,4]
4
4
[5,7]
24
5
[8,9]
-2
6
[0,1]
12
7
[2,2]
4
8
[3,3]
6
9
[4,4]
-2
10
[5,6]
6
11
[7,7]
18
12
[8,8]
-3
13
[9,9]
1
14
[0,0]
10
15
[1,1]
2
[5,5]
1
23
[6,6]
5
24
[1,6]
2+4+14+6

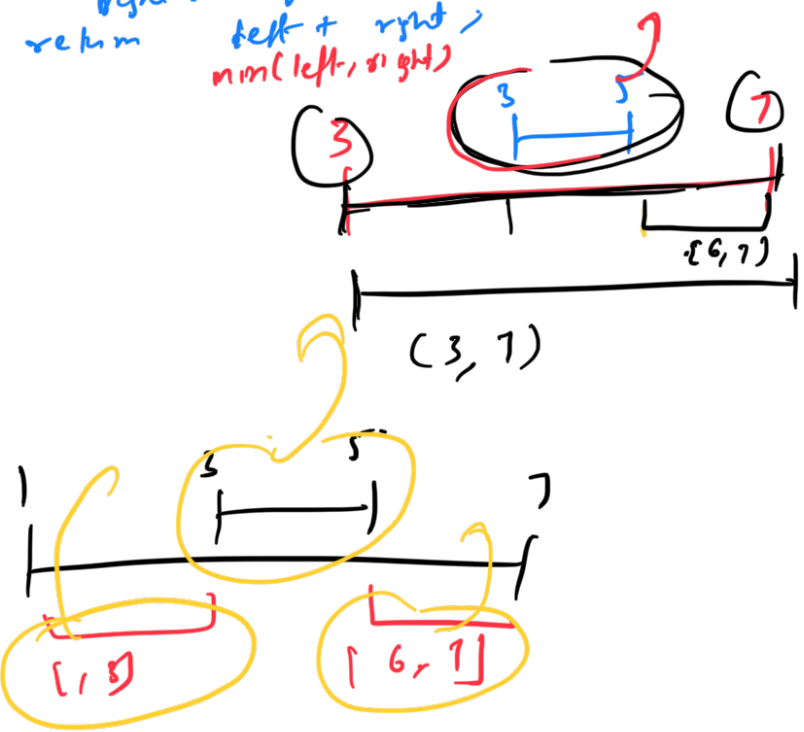
```
seg[4*N];
int query(ind, low, high, qL, qR) {
    // No overlap
    if( qL > high || qR < low)
        return 0;

    // Full overlap
    if( qL <= low && high <= qR)
        return seg[ind];

    // Partial overlap
    left = query(2*ind+1, low, mid, qL, qR);
    right = query(2*ind+2, mid+1, high, qL, qR);
    return left + right;
    min(left, right)
}
```

O(1)

MIN → INT_MAX
MAX → INT_MIN



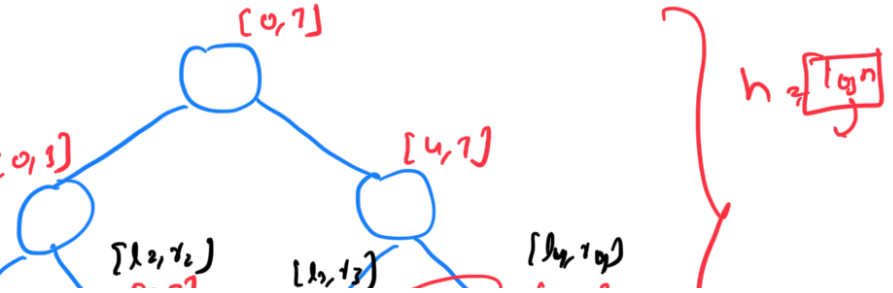
1) No overlap ⇒ Simply Return

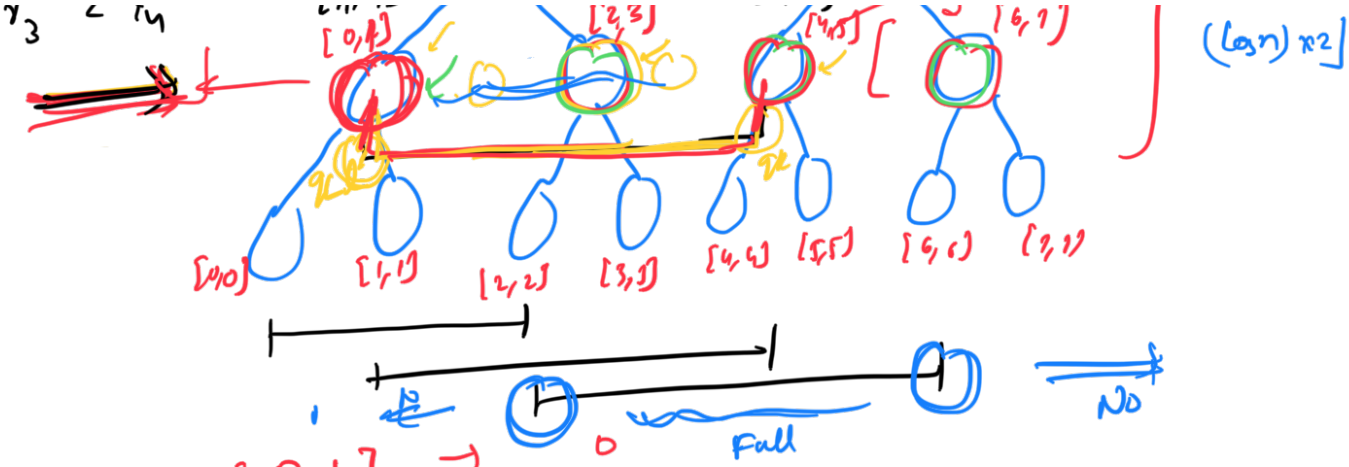
2) Complete overlap ⇒ Simply Return

3) Partial Overlap ⇒ Recur for Left & Right children

$r_1 < l_2 \Rightarrow [r_1 + 1 = l_2]$

$r_2 < l_3$





Range [0, 1] → 0

[1, 3] → 1

[1, 6] → 1 + 1 = 2.

→ logn Level

→ At every level → almost 2 partial overlap

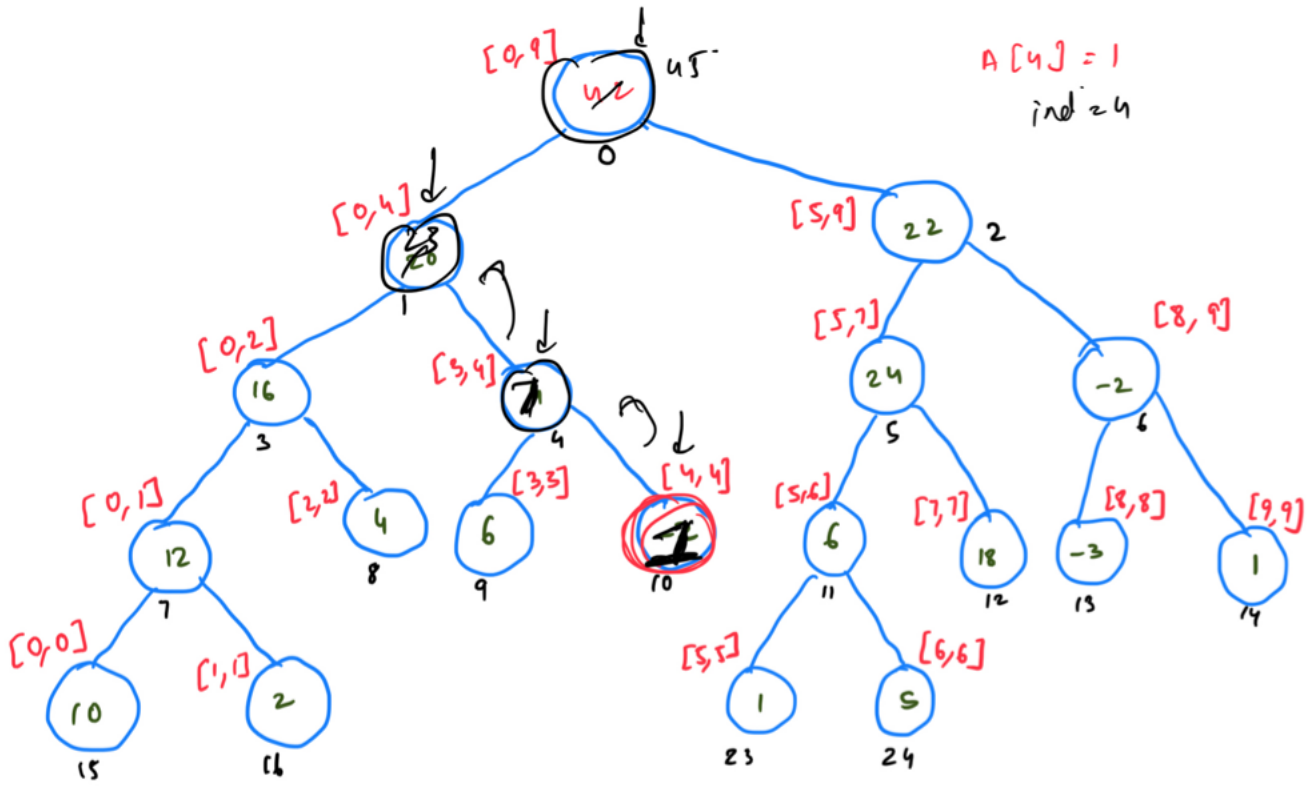
$O(2 \times \log n) = O(\log n)$

T.C: [# Function calls × T.C Per Function]

Function calls → 2 × logn.

T.C Per Function → O(1)

Update Query:


[0,9]
45
0
A[4] = 1
index 4
[0,4]
28
1
[5,9]
22
2
[0,2]
16
3
[3,4]
7
4
[5,7]
24
5
[8,9]
-2
6
[0,1]
12
7
[2,2]
4
8
[3,3]
6
9
[4,4]
1
10
[5,6]
6
11
[7,7]
18
12
[8,8]
-3
13
[9,9]
1
14
[0,0]
10
15
[1,1]
2
16
[5,5]
1
23
[6,6]
5
24

```
void update(ind, val, low, high) {
    if(low == high){
        seg[ind] = val;
        return;
    }
    mid = (high + low)/2;
    if(ind <= mid){
        update(2*ind+1, val, low, mid);
    }
    else{
        update(2*ind+2, val, mid+1, high);
    }
    seg[ind] = seg[2*ind+1] + seg[2*ind+2];
    return;
}
```

T.C: $O(\log n)$

Variations:

1) Sum → 0
2) Minimum → INT_MAX
3) Maximum → INT_MIN
4) Product → 1
5) gcd
6) XOR
7) AND
8) OR

Sqroot Decomp
Segment Trees

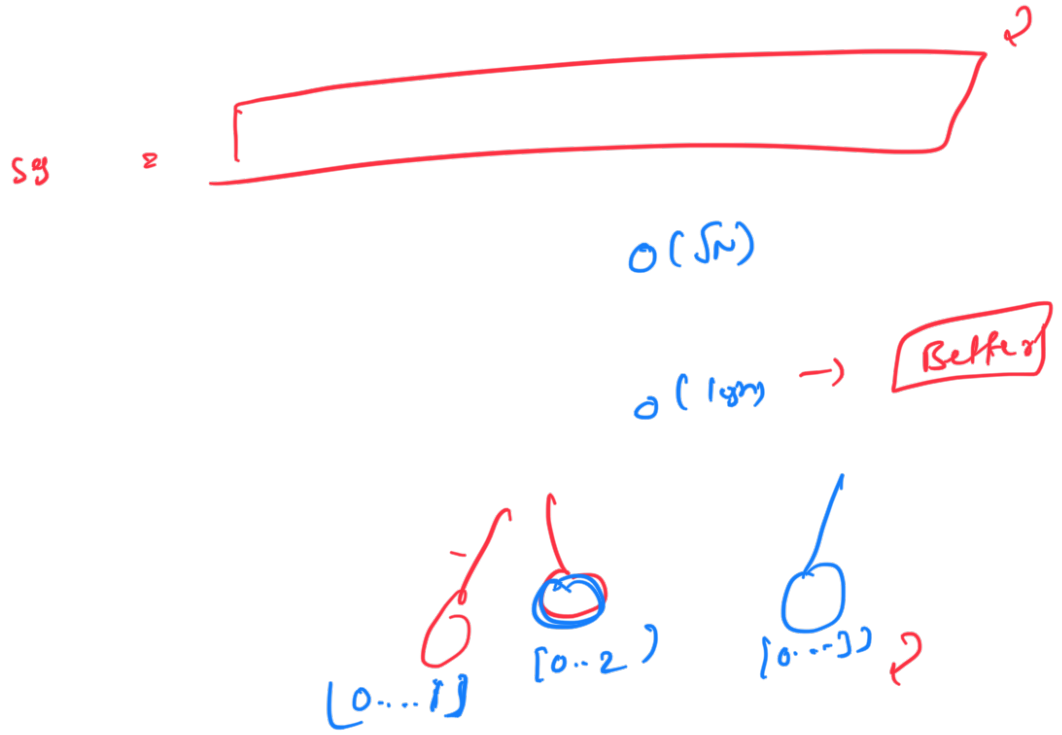
$a*b = b*a$
$a*b = b*a$

Steps

stored at any Node ?

1) What

2) How to compute value of ... using ... its children?




```
void build(ind, low, high){
      if(low == high) {
            tree[ind] = A[low];
      }
      mid = (low + high)/2;
      lc = 2*ind + 1;
      rc = 2*ind + 2;
      build(lc, low, mid);
      build(rc, mid + 1, high);
      tree[idx] = tree[lc] tree[rc];
}

build(0, 0, n-1)
```

```
int query(ind, low, high, qL, qR){
      // No overlap
      if(high < qL || low > qR) return 0;

      // Complete overlap
      if(low >= qL && high <=qR) return tree[ind];

      // Partial Overlap
      mid = (low + high)/2;
      int ans_l = query(2*idx+1, low , mid, qL, qR);
      int ans_r = query(2*idx+2, mid+1 , high, qL, qR);

return ans_l + ans_r;
}
```

```
void update(ind, low, high, i, val){
      if(low == high){
            tree[i] = val;
            // a[i] = val;
            return;
      }
      mid = (low + high)/2;
      if(i <= mid)
            update(lc, low, mid, i, val);
      else
            update(rc, mid+1, high, i, val);
      tree[idx] = (tree[lc]+ tree[rc]);
}
```