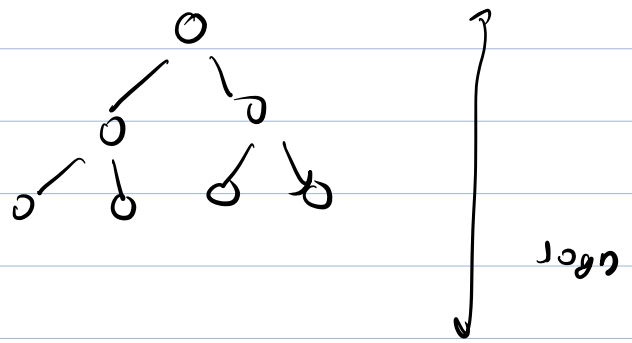
Build — TC : O(N)

query (l, r) — TC : O(log N)

update (i^th, val) — TC : O(log n)




$1 + 2 + 4 + \dots$

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

$= 1\left(\frac{2^{\log n} - 1}{2 - 1}\right)$

$S_n = N - 1$

```
void build ( int idx ; l, r)
{
    if (l == r)
    {
        segtree[idx] = arr[l]
        return
    }
    mid = __
    build on left child
    build on right child
    segtree[idx] = left + right
}
```

$$T(N) = 2 * T\left(\frac{N}{2}\right) + 1$$

$$T(N) = 2\left[2 \times T\left(\frac{N}{2}\right) + 1\right] + 1$$

$$T(N) = 4\,T\left(\frac{N}{4}\right) + 3$$

$$T(N) = N\,T\left(\frac{N}{N}\right) + (N-1)$$

$$T(N) = N + N - 1$$

$$T(N) = O(N)$$

sum of intervals.

update $[l, r]$ $+1$

```
for( i=l ; i≤r ; i++)
{
    update ith element = increase it by 1
    update( ith dimension, val)
}
```

$(r-l) \times \log N$

$\downarrow$

$N \log N$

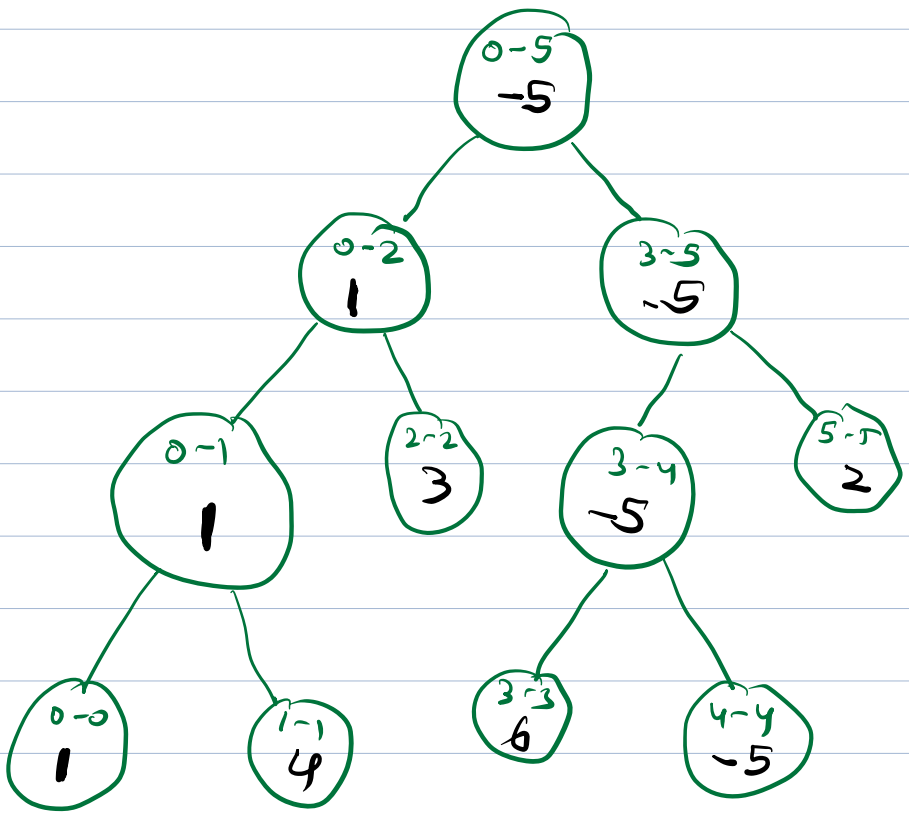
Build $\rightarrow$ $O(N)$

query(l, r) → O(log N)

update(l, r, val) → O(N log N)

0 1 2 3 4 5

{ 1, 4, 3, 6, -5, 2 }




0 - 4 +1

Build → O(N)

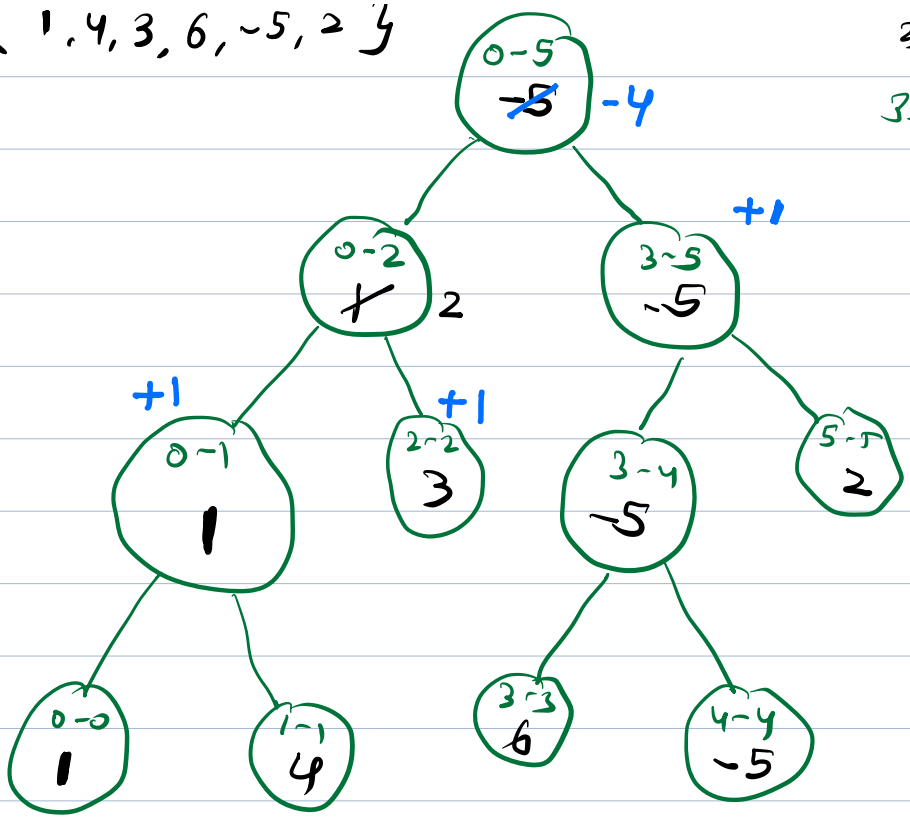
query(l, r) → O(log N)

update(l, r, val) → O(N)

0 1 2 3 4 5

1) update(0-5) +1

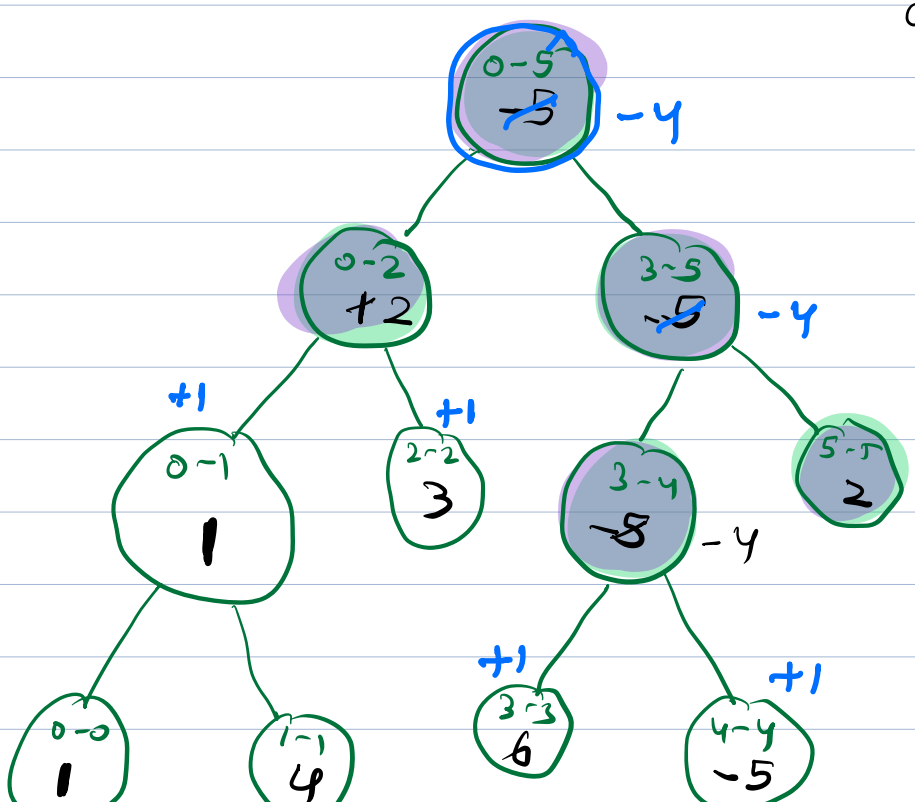
{1, 4, 3, 6, -5, 2}




2) query (0-5) = -4

3) query (0-2)

= 2




query (1-4)

↓

log N.

0-4

update (0-4)

↓

log N.

| | | |
|---|---|---|
| Build | → | O(N) |
| query(l, r) | → | O(log N) |
| update(l, r, val) | → | O(log N) |

Idea is update node where complete range is overlapping. and store somewhere that its children needed to be updated later.

**Lazy propogation**

delaying your updates, until they are actually required.

```
int segtree[4N]
int lazy[4N] = {0}
```

Build function ?

minimum element

```
void update ( idx, start, end, l, r, val )
{
    if ( lazy[idx] != 0 )
    {
        segtree[idx] += lazy[idx]
        if ( start != end )
        {
            lazy[2idx+1] += lazy[idx]
            lazy[2idx+2] += lazy[idx]
        }
        lazy[idx] = 0
    }

    if ( l > end || r < start ) return

    if ( start >= l && end <= r )
    {
        segtree[idx] += val
        if ( start != end )
        {
            lazy[2idx+1] += val
            lazy[2idx+2] += val
        }
    }

    int mid = (start + end) / 2

    update ( 2idx+1, start, mid, l, r, val )
    update ( 2idx+2, mid+1, end, l, r, val )

    segtree[idx] = min ( segtree[2idx+1], segtree[2idx+2] )
}
```

```
int query ( idx, start, end, l, r )
{
    if ( lazy[idx] != 0)
    {
        segtree[idx] += lazy[idx]
        if ( start != end)
        {
            lazy[2*idx+1] += lazy[idx]
            lazy[2*idx+2] += lazy[idx]
        }
        lazy[idx] = 0
    }
    if ( r < start || l > end ) return INT_MAX
    if ( start >= l || end <= r ) return segtree[idx]
    mid = (start+end)/2
    {
        query left
        query right
        return min( query left, query right )
    }
}
```

Q. Given a array of N integers.

q queries
- Type1 → l, r (flip sign)
- Type2 → (l, r) sum of elements

1 2 3

A = {2, 3, 4}

| q=2 | | | | |
|---|---|---|---|---|
| | T1 | 1 | 2 | {-2 -3 4} |
| | T2 | 1 | 3 | -1 |
| | T1 | 2 | 3 | {-2 3 -4} |
| | T0 | 2 | 2 | {-2 -3 -4} |
| | T2 | 2 | 3 | -7 |

A[N] given

build ( 0, 0, N-1)

for( i=1 ; i≤q ; i++)

{

if ( Type1)

{ update ( 0, 0, N-1, l, r)

else

{ print ( query (0, 0, N-1, l, r) )

}

void build ( idx, start, end)

{

if( start == end)

{ segtree[idx] = A[start]

return

int mid = (start + end) /2

```
build ( 2idx+1, start, mid)
build ( 2idx+2, mid+1, end)
segtree[idx] = segtree[2idx+1] + segtree[2idx+2]
```

```
void update (idx, start, end, l, r)
```

6 = 4 5 -3
-6 = -4 -5 3

```
if ( lazy[idx] != 0)
{
    segtree[idx] = segtree[idx] * -1
    if ( start != end)
    {
        lazy[2idx+1] = ! lazy[2idx+1]
        lazy[2idx+2] = ! lazy[2idx+2]
    }
    lazy[idx] = 0
}

if ( l > end || r < start)   return
if ( start >= l && end <= r)
{
    segtree[idx] = segtree[idx] * -1
    if ( start != end)
    {
        lazy[2idx+1] = ! lazy[2idx+1]
        lazy[2idx+2] = ! lazy[2idx+2]
    }
}

int mid = start + end
```

int mid = start + end / 2

update ( 2idx+1, start, mid, l, r, val )

update ( 2idx+2, mid+1, end, l, r, val )

segtree [idx] = segtree [2idx+1] + segtree [2idx+2]

int query ( idx, start, end, l, r )

{

if ( lazy [idx] != 0 )

{

segtree [idx] = segtreee [idx] * -1

if ( start != end )

{ lazy [2idx+1] = ! lazy [2idx+1]

lazy [2idx+2] = ! lazy [2idx+2]

}

lazy [idx] = 0

}

if ( r < start || l > end ) return 0

if ( start ≥ l || end ≤ r ) return segtree [idx]

mid = start + end / 2

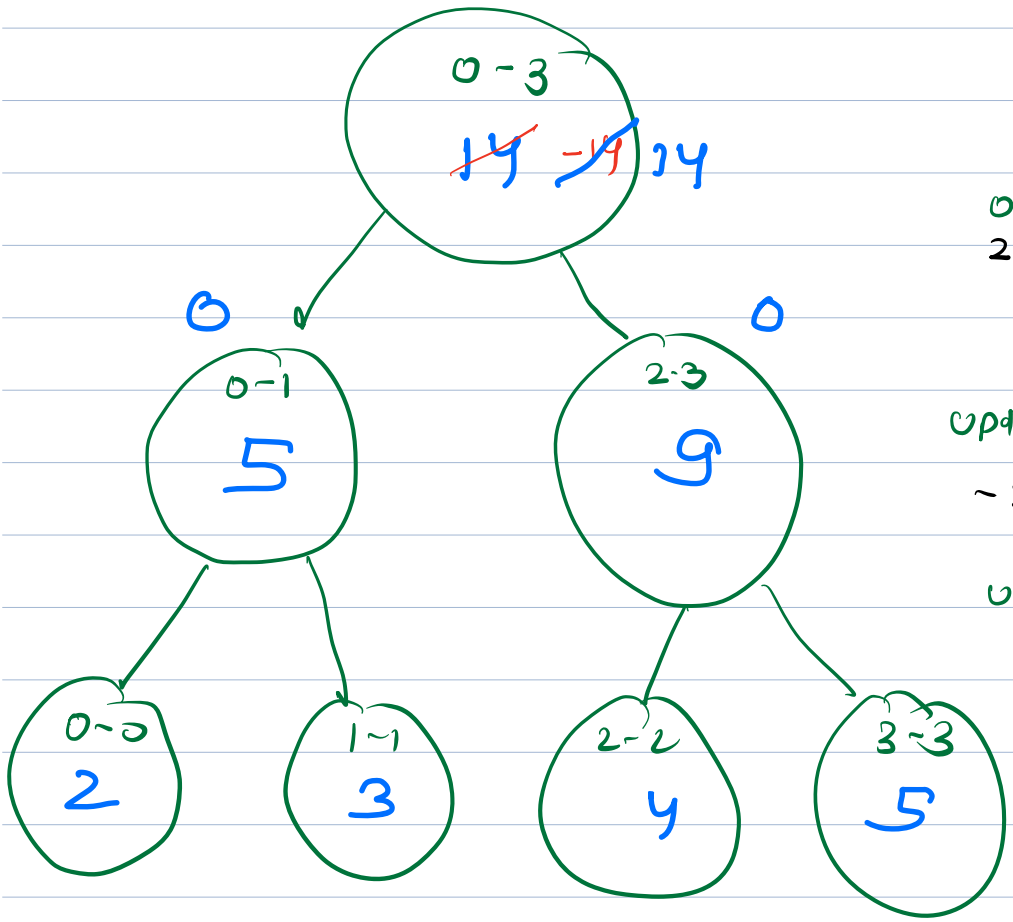
query left

query right

return query left + query right

}

Doubts




0 1 2 3
2, 3, 4, 5

update (0-3)
-2, -3, -4 -5

update (0-3)