1. Update
2. Query TC: N log N, SC: 4N
3. Build
4. TC: log N, (upd, query)

Q. Array of size N. Initially each element is one.
Q queries x y → update value at index x to 0
→ sum of elements from [x, y]

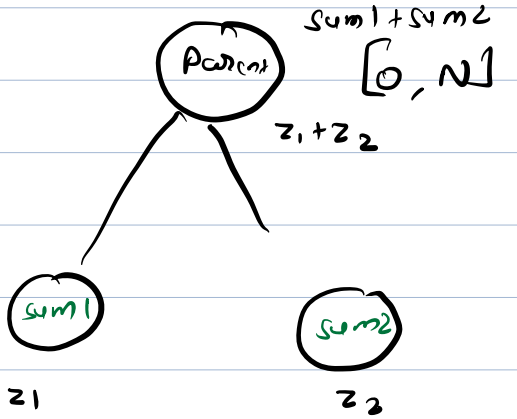
A = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
0 1 2 3 4 5 6 7 8 9

Q = 6

1	A	4		
2	B	0	4	ans = 4
3	A	6		
4	A	0		
5	B	2	6	ans = 3
6	B	1	8	ans = 6

0 1 2 3 4 5 6 7 8 9
[1, 1, 1, 1, 0, 1, 1, 1, 1, 1]

0 1 2 3 4 5 6 7 8 9
[0, 1, 1, 1, 0, 1, 0, 1, 1, 1]



[L R]

```
void build ( idx, start, end)
{
    if ( start == end)
    {
        tree[idx] = arr[start]
        return.
    }
    mid = (start+end)/2

    build ( 2idx+1, start, mid)
    build ( 2idx+2, mid+1, end)

    tree[idx] = tree [ 2*idx + 1 ] + tree (2idx + 2)
}
```

2i+1

0 ↘ 2i+2

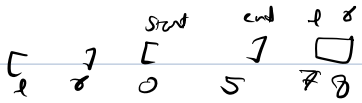
```
void update ( idx, start, end, l, val)
{
    if ( l == start && l == end)
    {
        a[l] = val
        tree[idx] = arr[start]
        return
    }
    mid = (start+end)/2

    if ( l >= start && l <= mid)
        update ( 2idx+1, start, mid, l, val)
    else
        update ( 2idx+2, mid+1, end, l, val)

    tree[idx] = tree [ 2*idx + 1 ] + tree (2idx + 2)
}
```

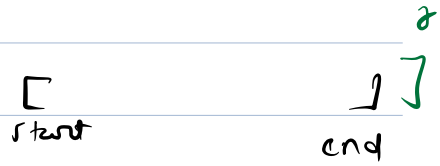
0 10

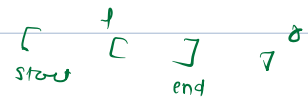
0 5 6 10

8

```
int query ( idx, start, end, l , r)
{
    if ( l > end   ||   r < start)
        return 0
    if ( end ≤ r && start ≥ l)
        return tree [idx]
    mid = (start + end) / 2
    return   query (2idx+1, start, mid , l, r )
           + query ( 2idx+2, mid+1, end , l, r)
}
```








```
int arr[N]
All  arr[i] = 1.
build ( 0, 0, N-1)
for ( i=1 ; i <= q ; i++ )
{
    x  y
    if ( type == 1)
    {
        update ( 0, 0, n-1, x, 0)
    }
    else
    {
```

logN

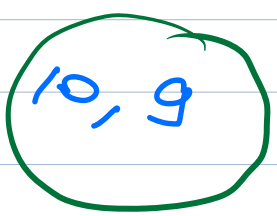
int ans = query(0, 0, n-1, l, r)
cout << ans

return

TC : O(N + Q log N)

SC : O(N)

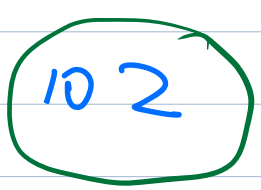
Q. [L R]



10
9
8
2

0 $\frac{N}{2}$

9 8 7, 6



$\frac{N}{2}+1$ N

10 1 2, 0

Q. Array of N. each 1 initially.

Queries → update a index 0

→ find index of x^{th} one in array

```
     0 1 2 3 4 5 6 7 8 9
A:  [1,1,1,1,1,1,1,1,1,1]
```

Q = 6

1	A	4	
2	B	5	ans = 5
3	A	6	
4	A	0	
5	B	3	ans = 3
6	B	4	ans = 5

```
     0 1 2 3 4 5 6 7 8 9
A:  [0,1,1,1,0,1,0,1,1,1]
```

sum [0, K] → 5

(100)

10^{th} one

Sum (0, 99) = 25

Sum (0, 50) = 8

0, 50 100

Bone

25

25 - 8 = 17

```
for( i=1 ; i<=q ; i++ )
    x
    if( type == 1 )
        update( 0, 0, n-1, x, 0 )
    else
        l = 0 , r = n-1                          logN
        while( l <= r )
            mid = (l+r)/2
            val = query( 0, 0, n-1, 0, mid )     logN
            if( val >= x )
                ans = mid
                r = mid-1
            else
                l = mid+1
    cout << ans
```

TC : O(N + Q log N log N)

Q. Array of size N.

Q queries → x^{th} element change to y
A[x] = y.

→ L R

1 × a[L] + 2 × a[L+1] + 3 × a[L+2]
+ ... (R-L+1) * a[R]

[5, 10]

Summation (i - L + 1) * A[i]
i from L to R

Summation(i A[i] - (L-1) A[i])

Array A = A[0] A[1] A[2] A[3] ...
Array B = B[0] B[1] B[2] B[3] ...

B[i] = i A[i]

query B (L,R) - (L-1) query A (L, R)

(L-1) (A[L] + (L-1) A(L+1) + L-1 A(L+2) ~

(L-1) [A[L] + A(L+1) + A(L+2) ~ A(R)

Array B : 0 x A[0] 1 x A[1] 2 x A(2) 3 x A(3)

+ ~ , (n-1) (A n-1)

```
void build ( idx, start, end)
{
    if ( start == end)
    {
        tree1[idx] = A [start]
        tree2 [idx] = B [start]
    }
    return.
    mid = (start + end) / 2
    build ( 2idx+1, start, mid)
    build ( 2idx+2, mid+1, end)
    tree1[ idx] = tree1[ 2*idx + 1] + tree1 ( 2idx + 2)
```

2i+1

0 ↘

2i+2

tree2[idx] = tree2[2*idx+1] + tree2[2*idx+2]

```
void update(idx, start, end, l, val)
{
    if (l == start && l == end)
    {
        a[l] = val
        tree1[idx] = arr[start]
        tree2[idx] = start * arr[start]
        return
    }
    mid = (start + end) / 2
    if (l >= start && l <= mid)
        update(2*idx+1, start, mid, l, val)
    else
        update(2*idx+2, mid+1, end, l, val)
    tree1[idx] = tree1[2*idx+1] + tree1[2*idx+2]
    tree2[idx] = tree2[2*idx+1] + tree2[2*idx+2]
}
```

0 10

0 5 6 10

```
int queryB(idx, start, end, l, r)
{
    if (l > end || r < start)
        return 0
    if (end <= r && start >= l)
        return tree2[idx]
    mid = (start + end) / 2
    return queryB(2*idx+1, start, mid, l, r)
         + queryB(2*idx+2, mid+1, end, l, r)
}
```

start end l r

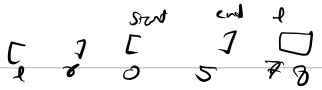
[l r] [0 5] [8]

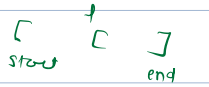
l r

[start] [end]

[start] [end]

```
int queryA( idx, start, end, l, r)
{
    if ( l > end   ||   r < start)
        return 0
    if ( end <= r && start >= l)
        return tree[idx]

    mid = (start + end) / 2

    return   queryA(2idx+1, start, mid, l, r )
           + queryA( 2idx+2, mid+1, end, l, r)
}
```






```
B[N]
for ( i=0 ; i<N ; i++)   B[i] = i * A[i]

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

```
for ( i=1 ; i<=q ; i++ )
{
    x  y

    if ( type == 1)
```

```
{
    update(0, 0, n-1, x, 0)
}
else
{
    int ans = queryB(0, 0, n-1, l, r)
              - (l-1) * queryA(0, 0, n-1, l, r)
}
```

query (l, R)

update (l, R) → lazy propagation.

Sat 9PM.