Segment Trees 1
Monday, 3 August 2020 3:06 PM

	0	1	2	3	4	5	6	7	8	9	10
arr[i] =	3	2	-1	5	7	10	8	2	4	-7	9

Q: Sum Range Queries

Type	l	r	
1	2	6	: 25
1	1	4	: 13

Prefix sum + other queries: $O(N)$ $O(1)$

for 2 $\le$ 10:
- update array $O(1)$
- update pf[] : $O(N)$

1 1 7 : —

Overall TC:
Type 1 query : $O(1)$
Type 2 query : update/pf[] → N
for a window $\emptyset$:
TC: $O(Q*N)$

Q: Max range query / Assume no updates

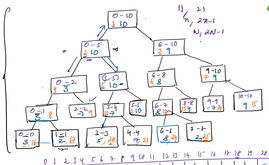
Type	l	r	
1	2	6	: 10
1	3	7	: 10

$\boxed{\text{Max}(l-r) = \text{Max}(l-k) = \text{Max}(k-r)}$ *

For every query simply iterate & get max. TC: $O(Q*N)$

Segment Trees (SG) Range Max Query

	0	1	2	3	4	5	6	7	8	9	10
	3	2	-1	5	7	10	8	2	4	-7	9



Tree[] = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
10 10 5 3 10 8 3 3 -1 7 10 8 4 7 -1 9 3 8 - - 5 7 - - 8 2 - -

Prerequisite: Bottom up

Eg: Fix Balanced Binary Tree

Note: If l==r, it will be a leaf node

NOTE 1: For N Nodes Tree[N+1] MUST should be our Tree[]?

→ lot of memory wastage?

If (N) elements then many nodes we have to create (2N)

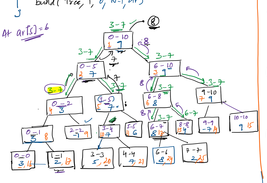
Parameter: type, parent, array → $2N$ vs $4N$ & we prefer arrays for segment Tree

Define N Elements
Declare Tree[4*N+1]

```
void build( int Tree[], int i, int l, int r, int arr[]){
    if( l==r) { Tree[i] = arr[l]; return;}
    int m = (l+r)/2
    build( Tree, 2i, l, m, arr)      Tree[2i]
    build( Tree, 2i+1, m+1, r, arr)  Tree[2i+1]
    Tree[i] = max(Tree[2i], Tree[2i+1])
}

main() {
    build( Tree, 1, 0, N-1, arr)
}
```

At arr[i] = 6




```
int Query( int tree[], int i, int l, int r, int qs, int qe){
    if( qs <= l && r <= qe){
        return tree[i]
    }
    if (r < qs || qe < l){
        return int_MIN
    }
    int m = (l+r)/2
    int lm = Query( tree, 2*i, l, m, qs, qe)
    int rm = Query( tree, 2*i+1, m+1, r, qs, qe)
    return max(lm, rm)
}
```

TC: $\log N$

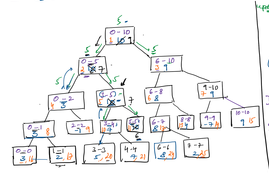
Query

qs qe — Query range
l r — Segment range

Case 1: Complete Overlap
qs <= l && r <= qe
return current node value

Case 2: No Overlap
l r qs — qe l r
(r < qs || qe < l)
return INT_MIN

Case 3: Partial Overlap
→ Call Query for both left & right
→ return max / both



Update:
arr[i] = 6 — qfind
find_i = 6 — find we are trying to update

(l r)
segment range = l r

1) No overlap: (qfind < l || qfind > r)
2) Full overlap: (qfind == l == r)
3) Partial ovl:
↳ Call left
↳ Call right
update current node

```
void update( int tree[], int i, int l, int r, int qfind, int val){
    if( qfind < l || qfind > r){ return } // No
    if( qfind == l && l == r){
        tree[i] = val // updating to new val
        return;
    }
    int mid = (l+r)/2
    update( tree, 2*i, l, m, qfind, val)
    update( tree, 2*i+1, m+1, r, qfind, val)
    tree[i] = max( tree[2i], tree[2i+1])
}
```

TC: $O(\log N)$

```
void build( int Tree[], int i, int l, int r, int arr[]){
    if( l==r) { Tree[i] = arr[l]; return;}
    int m = (l+r)/2
    build( Tree, 2i, l, m, arr)      Tree[2i]
    build( Tree, 2i+1, m+1, r, arr)  Tree[2i+1]
    Tree[i] = max( Tree[2i], Tree[2i+1])
}
```

4 Points
1) What are we storing in Tree[i]?
2) How to compute Tree[i], using its children
3) Using query vals of left & right how to get query of overall
4) In query if No overlap, what to return?

Playing Sum:
Tree[i] : sum of all elements of segment range
Tree[i] = Tree[2i] + Tree[2i+1]

5PM
7PM - 10PM, 3 Problems

	0	1	2	3	4	5	6	7	8
arr[ ]:	3	2	7	7	101	17	25	21	40

Q:
1 l r : Number of prime from l r
2 i val : arr[i] = val

```
int Query( int tree[], int i, int l, int r, int qs, int qe){
    if( qs <= l && r <= qe){
        return tree[i]
    }
    if (r < qs || qe < l){
        return INT_MIN 0
    }
    int m = (l+r)/2
    int lm = Query( tree, 2*i, l, m, qs, qe)
    int rm = Query( tree, 2*i+1, m+1, r, qs, qe)
    return lm+rm;
}
```

TC: $\log N$

```
void update( int tree[], int i, int l, int r, int qfind, int val){
    if( qfind < l || qfind > r){ return } // No
    if( qfind == l && l == r){
        tree[i] = IsPrime(val)
        return;
    }
    int mid = (l+r)/2
    update( tree, 2*i, l, m, qfind, val)
    update( tree, 2*i+1, m+1, r, qfind, val)
    tree[i] = tree[2*i] + tree[2*i+1]
}
```

TC: $O(\log N)$

Tree[i] = Number of prime in given range

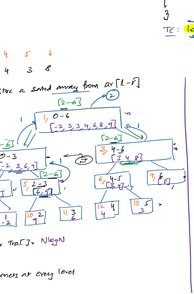
for IsPrime()/ Use Sieve

Q:

	0	1	2	3	4	5	6
arr[i] :	3	-2	7	6	4	3	8

Query:
l r
In given range l to r how many odd value are 2 → 4
2 6 → 3
3 5 8 → 3

Tree[i] : will store a sorted array from arr[l-r]



TC: To construct Tree[] → $N\log N$
SC: $N\log N$
We store N elements at every level

```
vector<int> Tree[N+1];
void build( vector<int> Tree[], int i, int l, int r, int arr[]){
    if( l==r) { Tree[i].push_back(arr[l]); return;}
    int m = (l+r)/2
    build( Tree, 2i, l, m, arr)      Tree[2i]
    build( Tree, 2i+1, m+1, r, arr)  Tree[2i+1]
    Tree[i] = merge( Tree[2i], Tree[2i+1])
}
```

```
int Query( vector<int> tree[], int i, int l, int r, int qs, int qe, int x){
    if( qs <= l && r <= qe){
        return (Apply && on tree[i] & calculate no. of elements > x);
    }
    if (r < qs || qe < l){
        return 0
    }
    int m = (l+r)/2
    int lm = Query( tree, 2*i, l, m, qs, qe)
    int rm = Query( tree, 2*i+1, m+1, r, qs, qe)
    return lm+rm;
}
```

TC: $(\log N) + \log(R-L)$