09-XC

BST Application

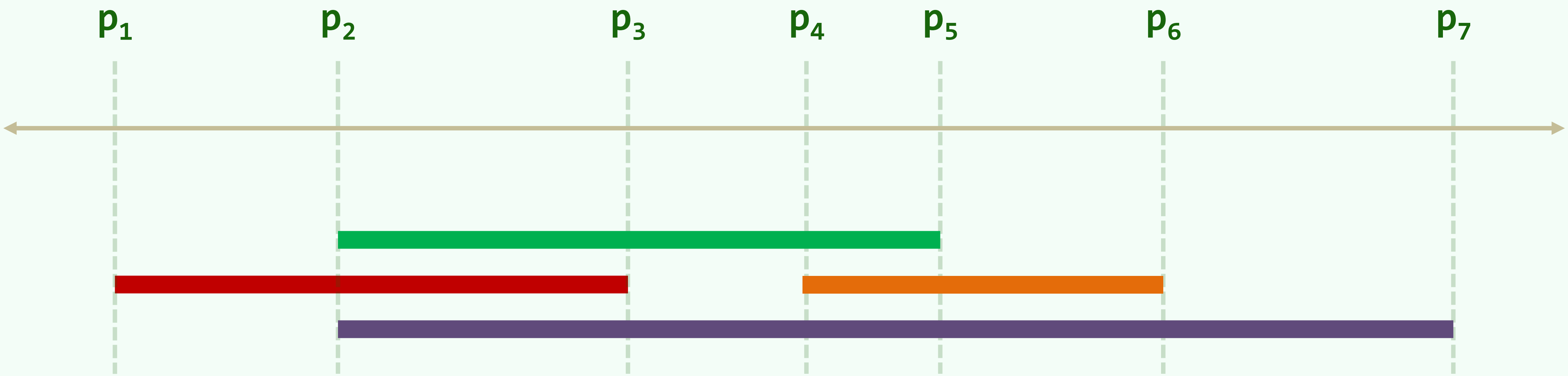
Segment Tree

邓俊辉
deng@tsinghua.edu.cn

Elementary Intervals

- Let $I = \{ [x_i, x_i'] \mid i = 1, 2, 3, \ldots, n \}$ be n intervals on the x-axis

- Sort all the endpoints into $\{ p_1,\ p_2,\ p_3,\ \ldots,\ p_m \},\ m \leq 2n$



- m+1 elementary intervals are hence defined as:

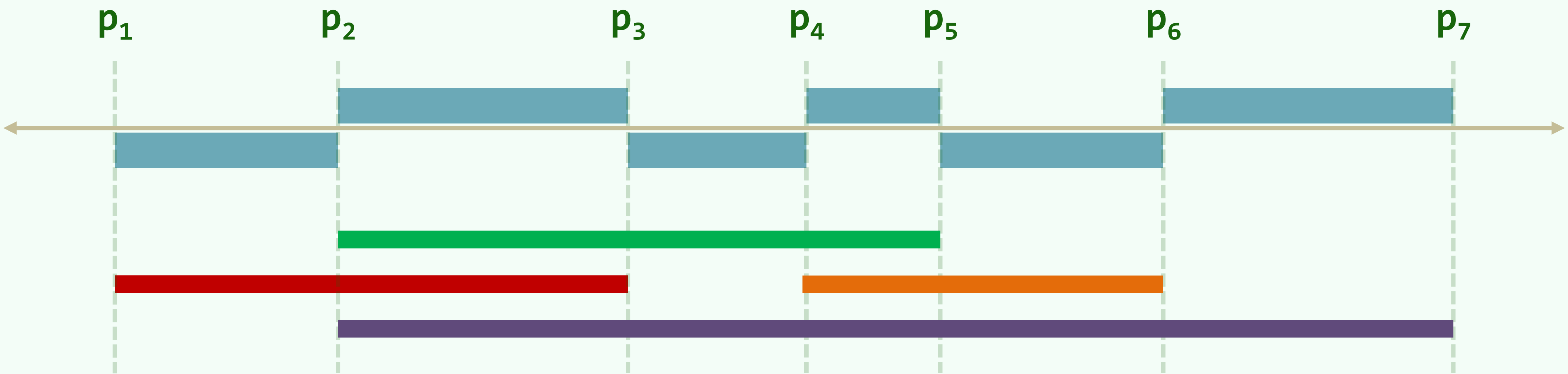
$$(-\infty, p_1],\ (p_1, p_2],\ (p_2, p_3],\ \ldots,\ (p_{m-1}, p_m],\ (p_m, +\infty]$$

Discretization

👁Within each EI, all stabbing queries share a same output

∴ If we **sort** all EI's into a vector and

store the corresponding **output** with each EI, then ...



∴ Once a query position is determined, //by an $\mathcal{O}(\log n)$ time binary search

the output can then be returned directly //$\mathcal{O}(r)$

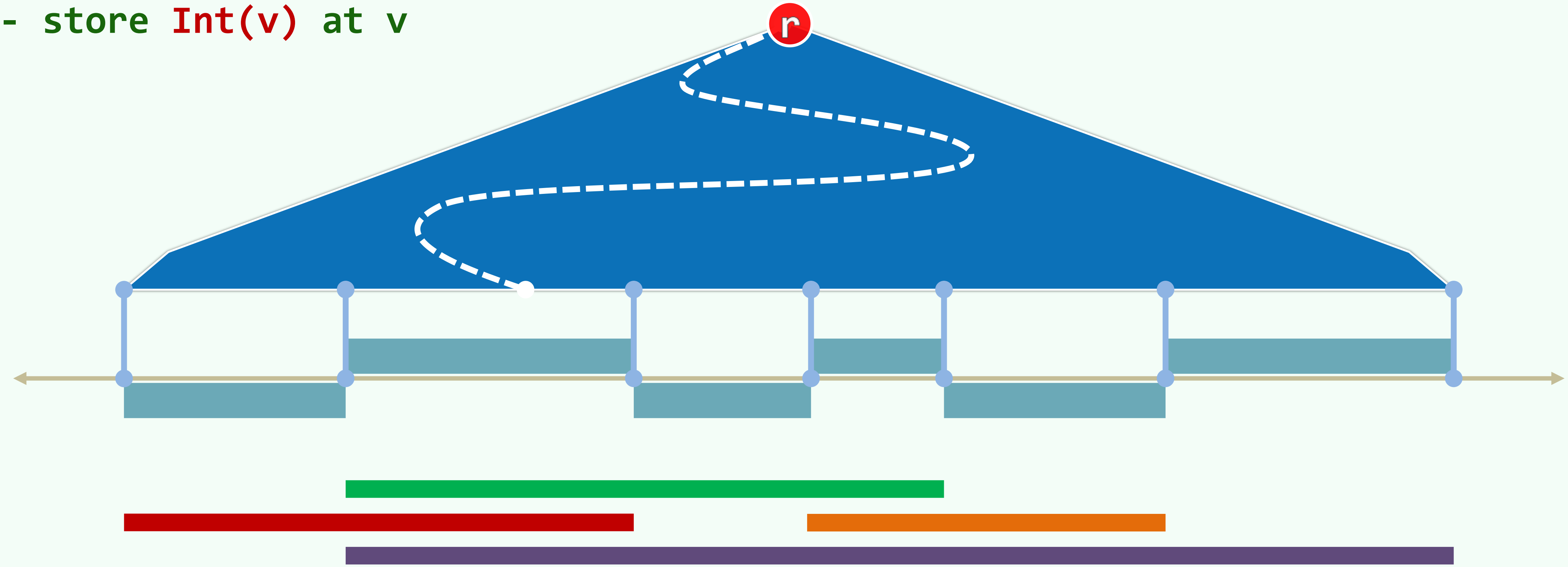
Worst Case

❖ Every interval spans $\Omega(n)$ EI's and a total space of $\Omega(n^2)$ is required

Sorted Vector --> BBST

❖ For each leaf v,

- denote the corresponding elementary interval as EI(v)
- denote the subset of intervals containing EI(v) as Int(v) and
- store Int(v) at v

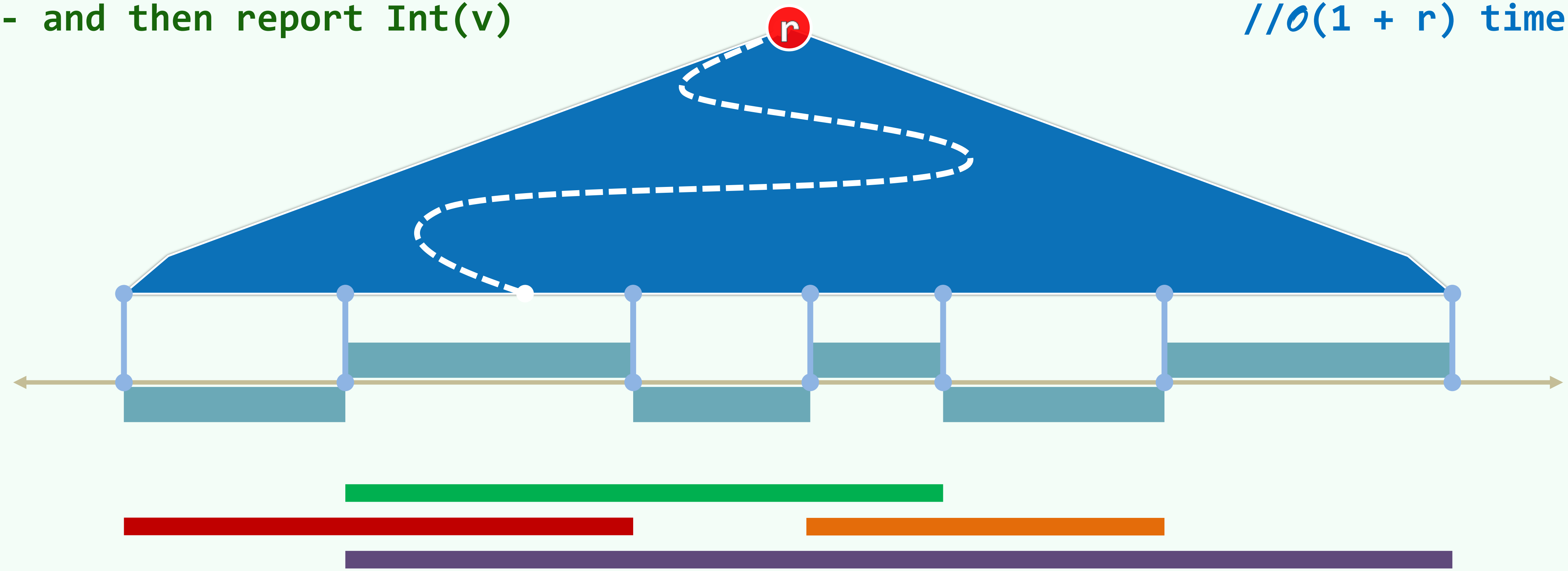


1D Stabbing Query with BBST

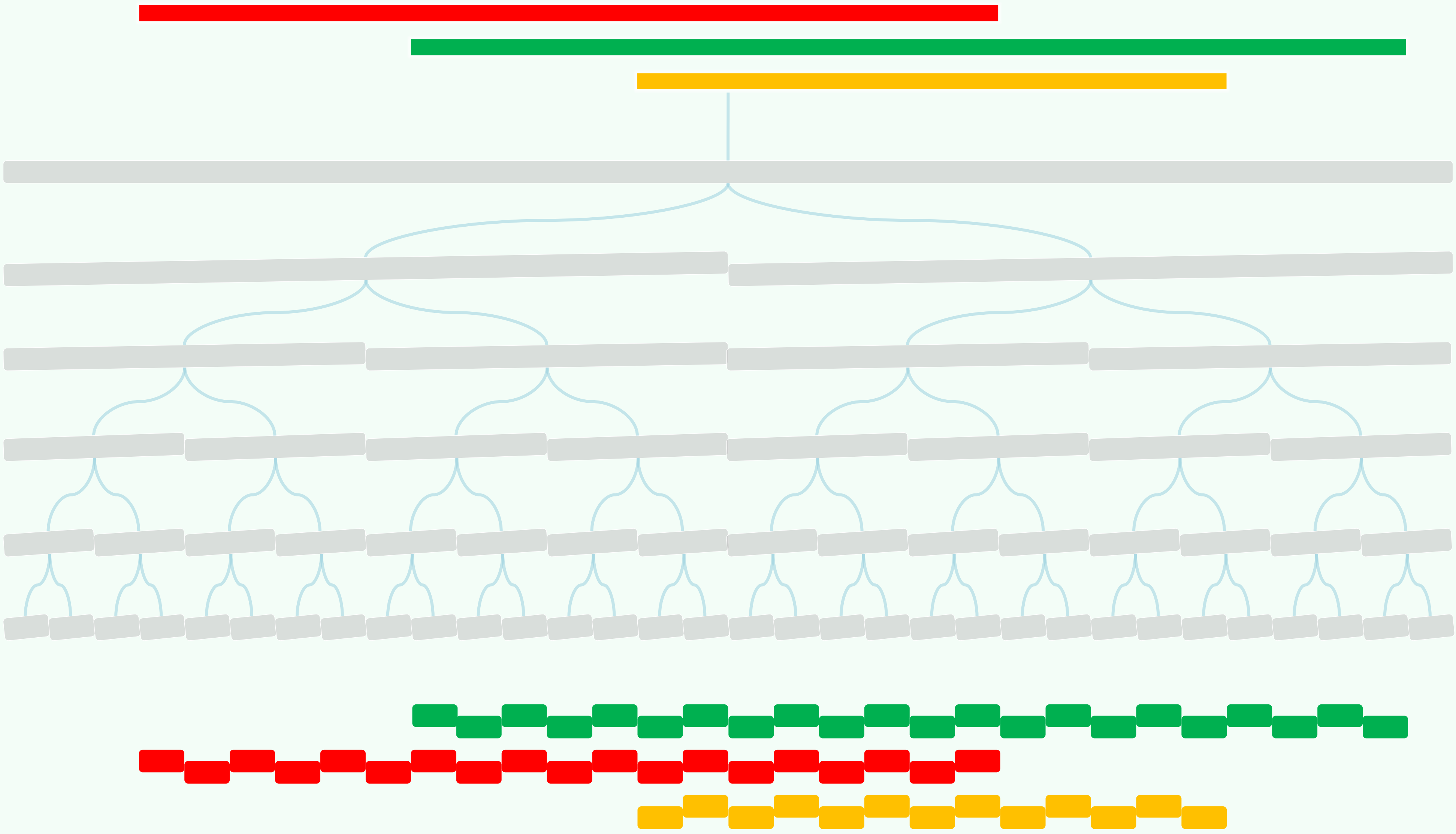
❖ To find all intervals containing q_x, we can

- find the EI(v) containing q_x //$\mathcal{O}$(logn) time for a BBST

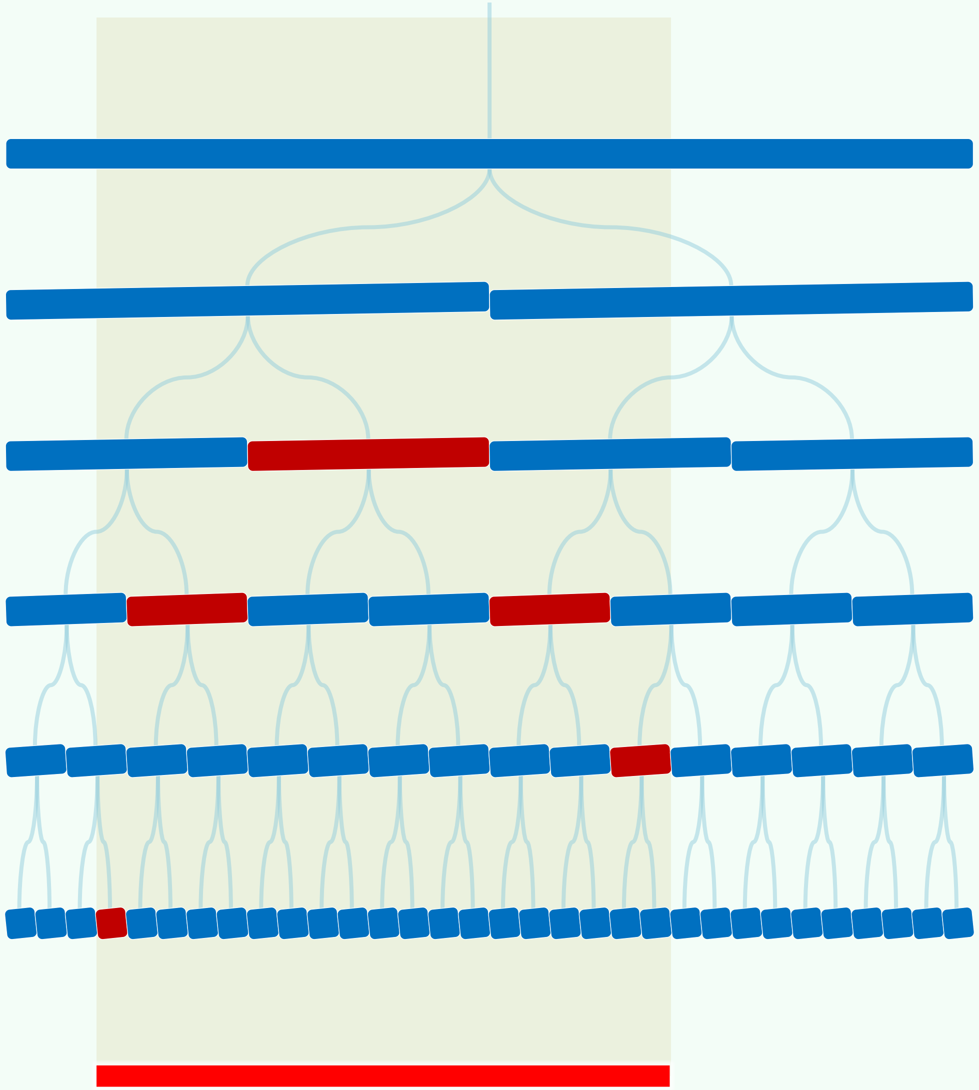
- and then report Int(v) //$\mathcal{O}$(1 + r) time

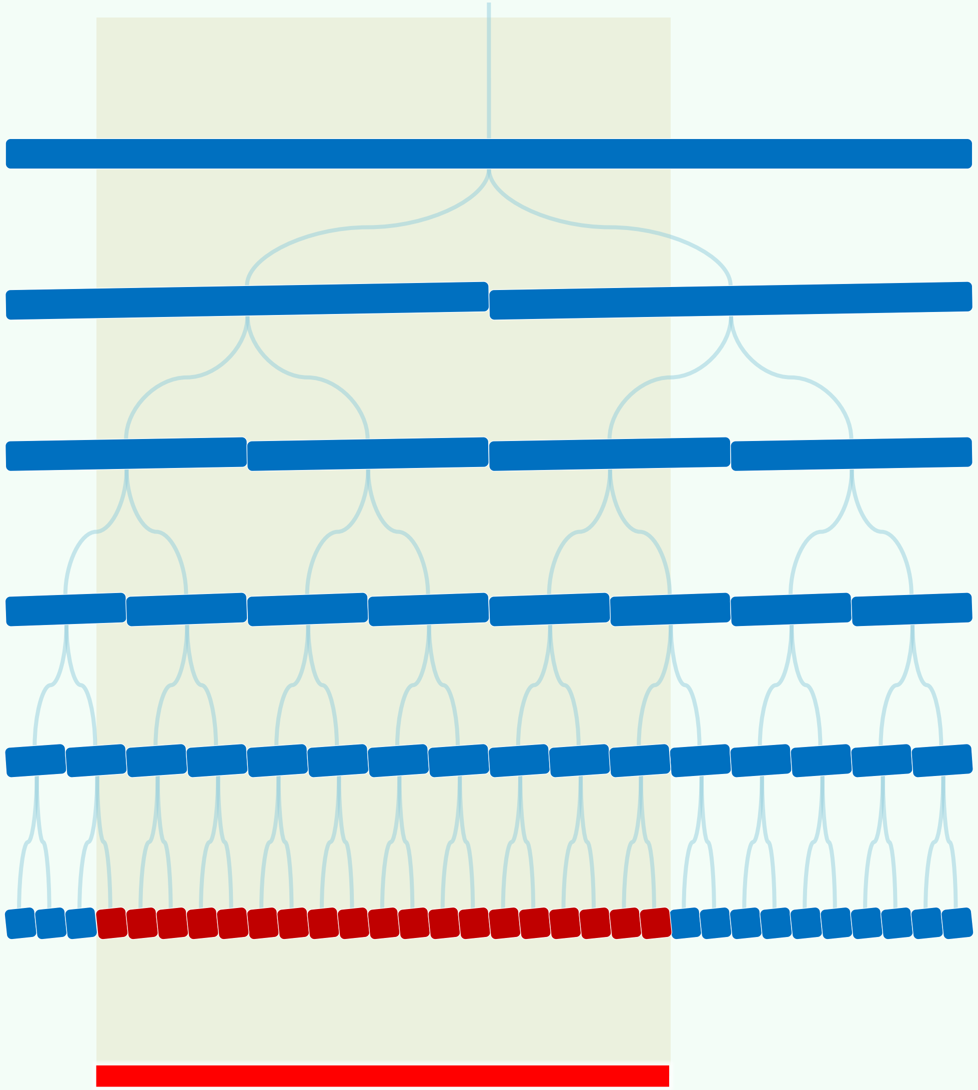


$\Omega(n^2)$ Total Space In The Worst Cases

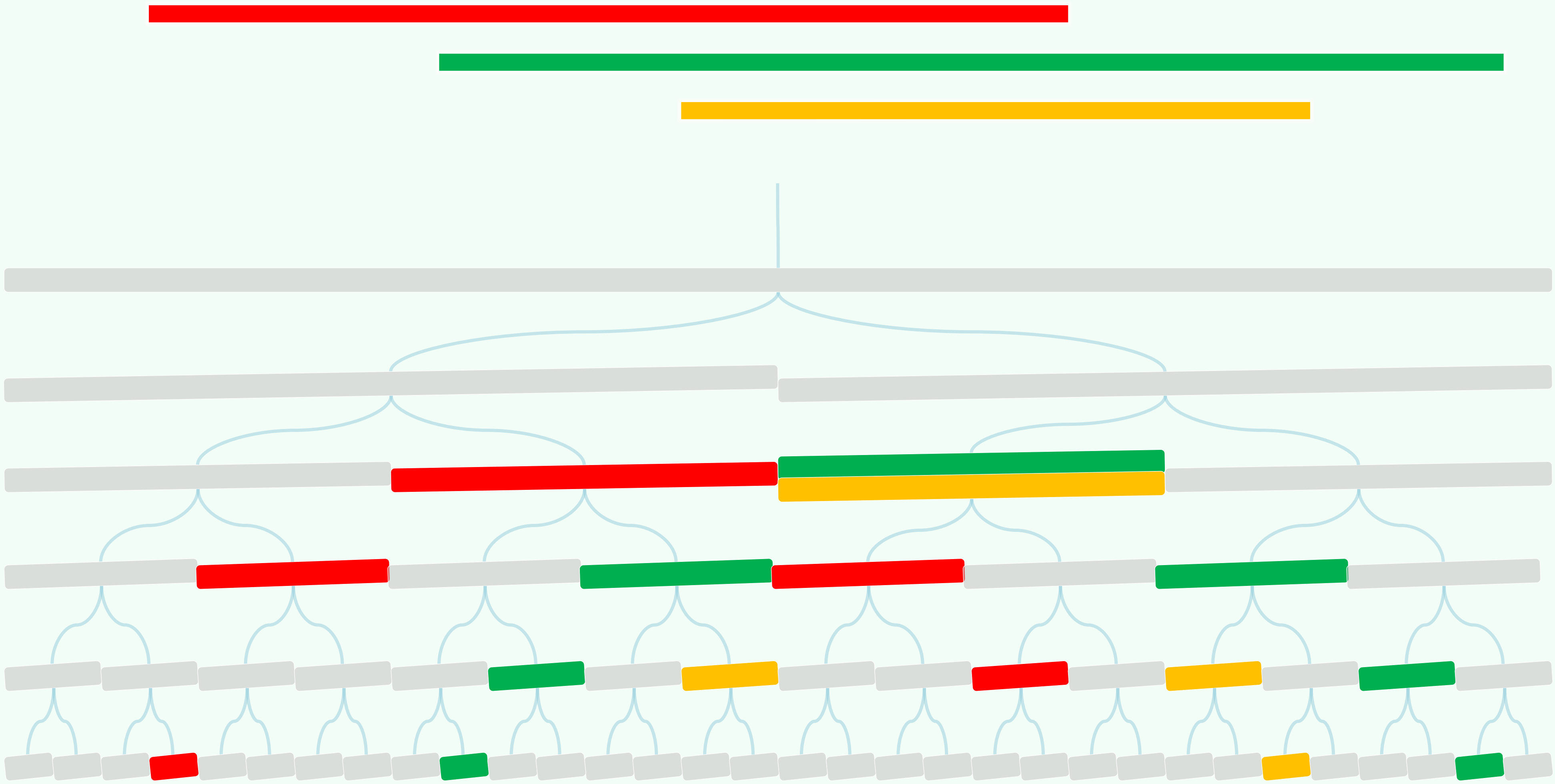


Merge At Common Ancestors



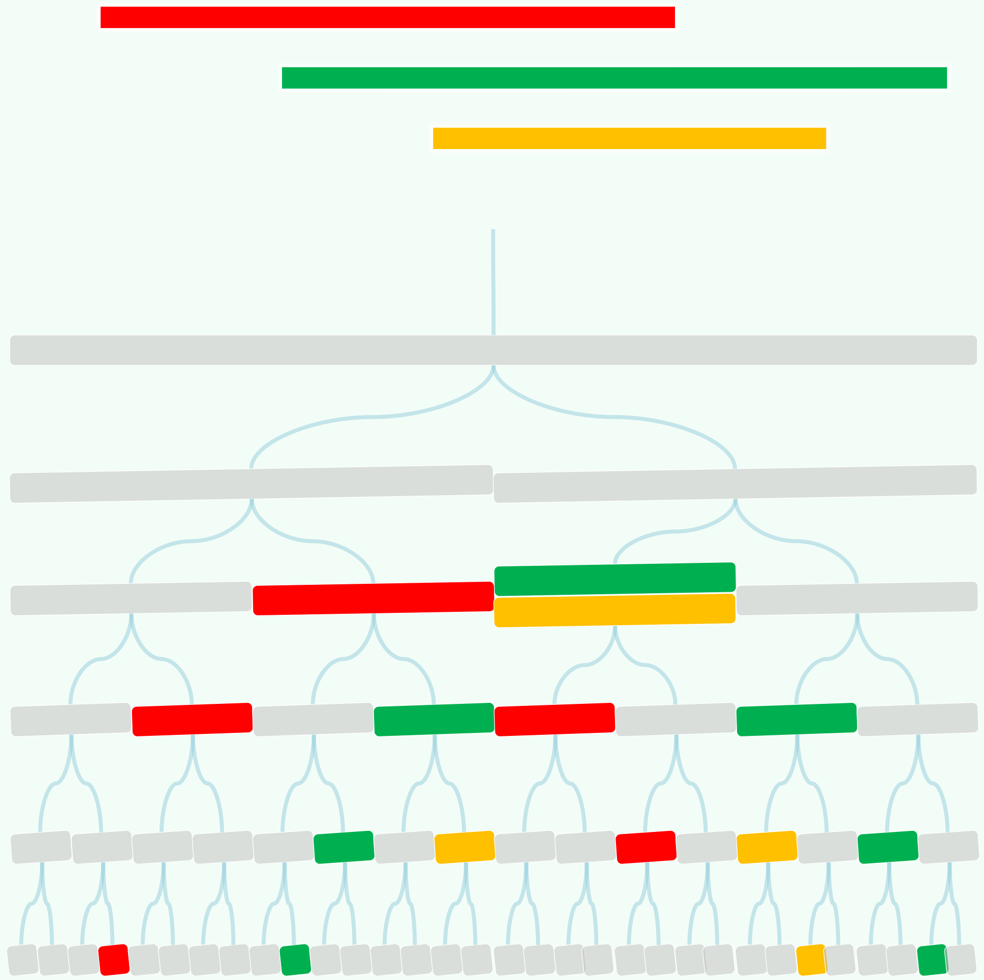


Canonical Subsets with $\mathcal{O}(n\log n)$ Space



BuildSegmentTree(I)

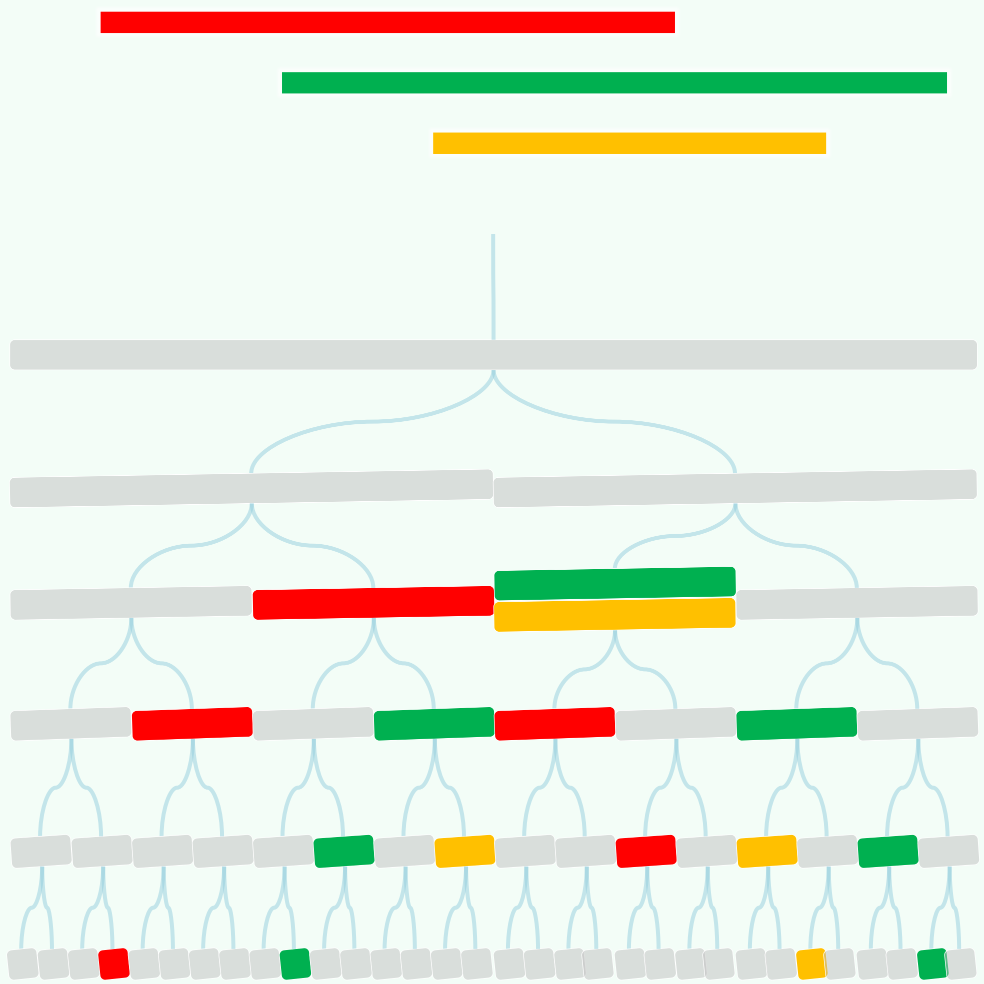
```
❖ // Construct a segment tree on
  // a set I of n intervals
  Sort all endpoints in I before
     determining all EI's //O(nlogn)
  Create T a BBST on all the EI's //O(n)
  Determine Int(v) for each node v
     //O(n) if done in a bottom-up manner
  For each s of I
     call InsertSegmentTree( T.root , s )
```



InsertSegmentTree(v , s)

❖ // Insert an interval s into

```
// a segment (sub)tree rooted at v
if ( Int(v) ⊆ s )
   store s at v and return;
if ( Int( lc(v) ) ∩ s ≠ ∅ ) //recurse
   InsertSegmentTree( lc(v), s );
if ( Int( rc(v) ) ∩ s ≠ ∅ ) //recurse
   InsertSegmentTree( rc(v), s );
```



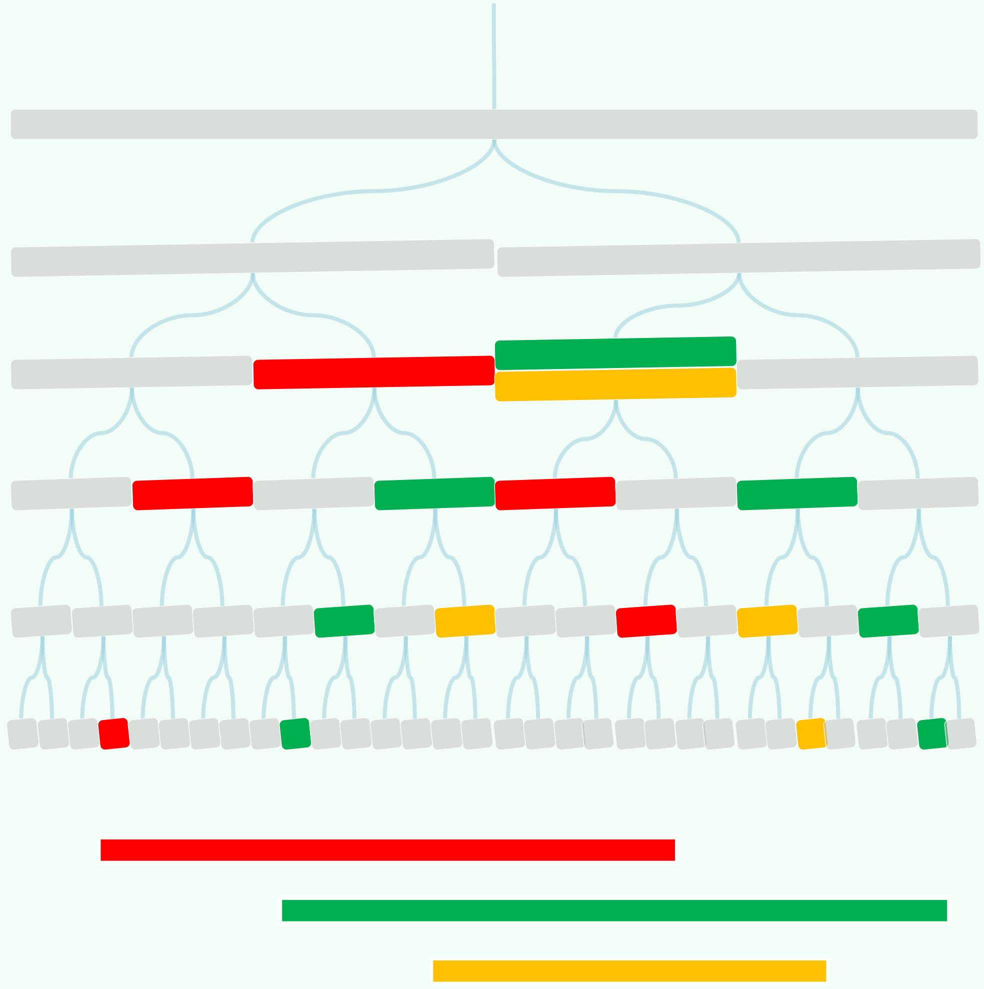
👁 At each level, ≤4 nodes are visited

(2 stores + 2 recursions)

∴ $\mathcal{O}(\log n)$ time

QuerySegmentTree(v , q_x)

```
// Find all intervals
// in the (sub)tree rooted at v
// that contain qx
report all the intervals in Int(v)
if ( v is a leaf )
   return
if ( qx ∈ Int( lc(v) ) )
   QuerySegmentTree( lc(v), qx )
else
   QuerySegmentTree( rc(v), qx )
```



$\mathcal{O}(r + \log n)$

- Only one node is visited per level,

  altogether $\mathcal{O}(\log n)$ nodes

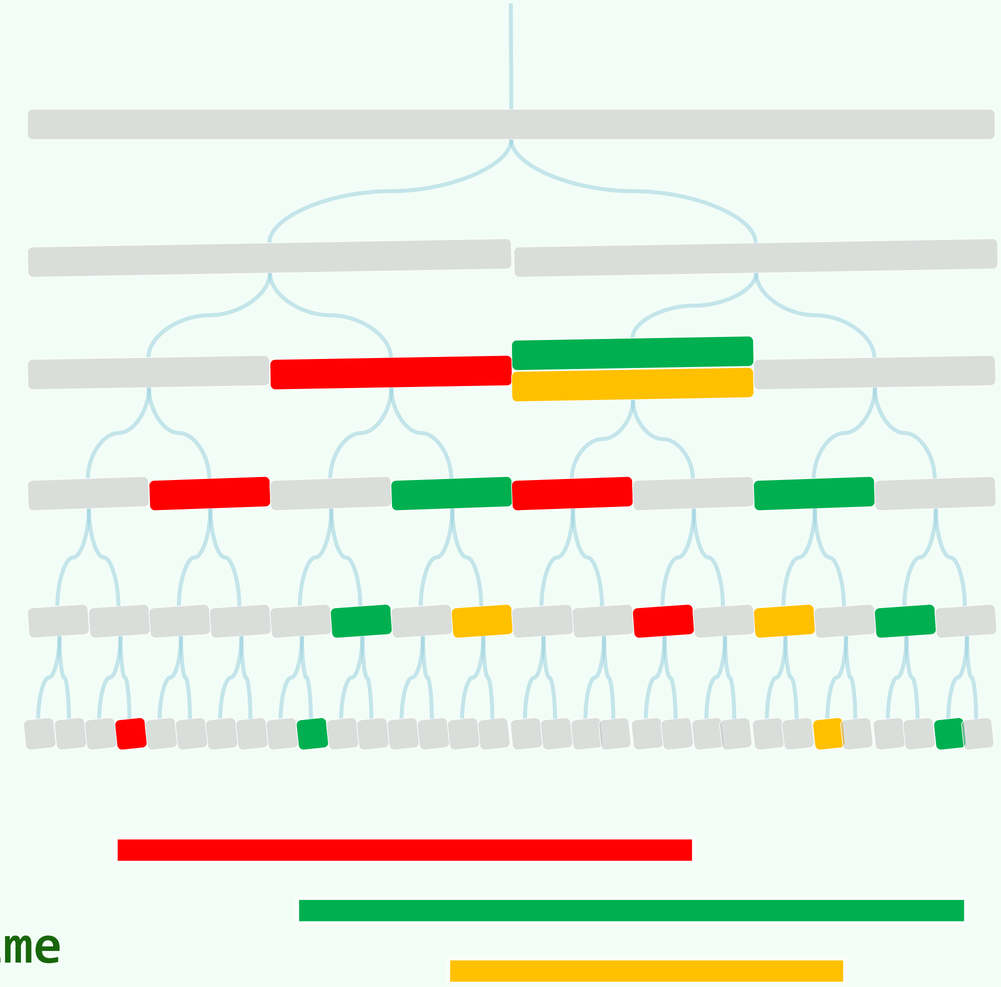
- At each node v

  - the CS Int(v) is reported

  - in time

    $$1 + |\mathrm{Int}(v)| = \mathcal{O}(1 + r_v)$$

∴ Reporting all the intervals costs $\mathcal{O}(r)$ time



Conclusion

❖ For a set of n intervals,

- a segment tree of size $\mathcal{O}(n\log n)$
- can be built in $\mathcal{O}(n\log n)$ time
- which reports all intervals

  containing a query point

  in $\mathcal{O}(r + \log n)$ time