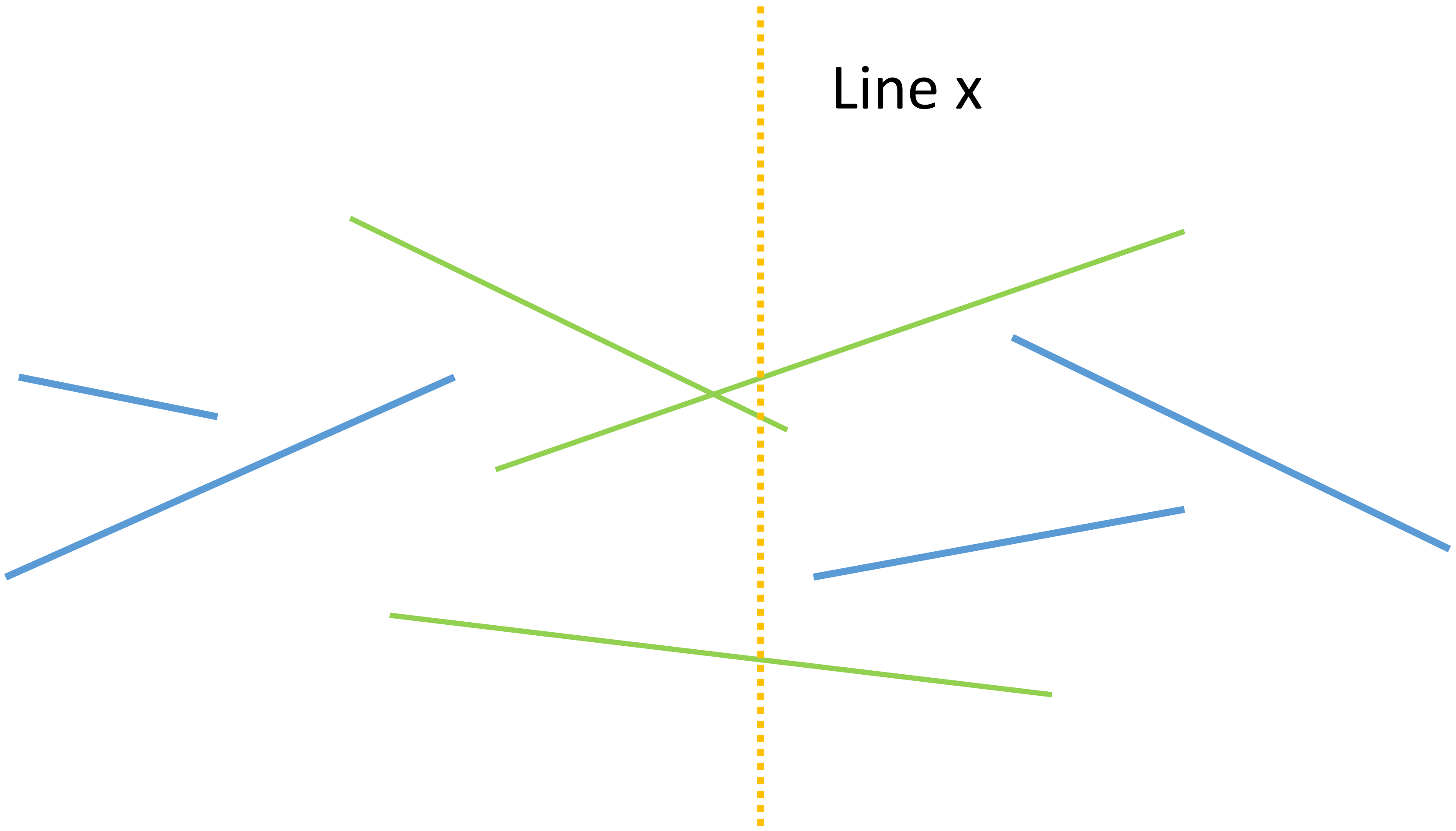
Stabbing Query

Interval Tree & Segment Tree

Nattee Niparnan

Stabbing Query



Green = reported

Stabbing Query

- From stabbing query, the y-coordinates are irrelevant

- Point Search:
  - Given n interval and a point x
  - Which intervals contain the point?

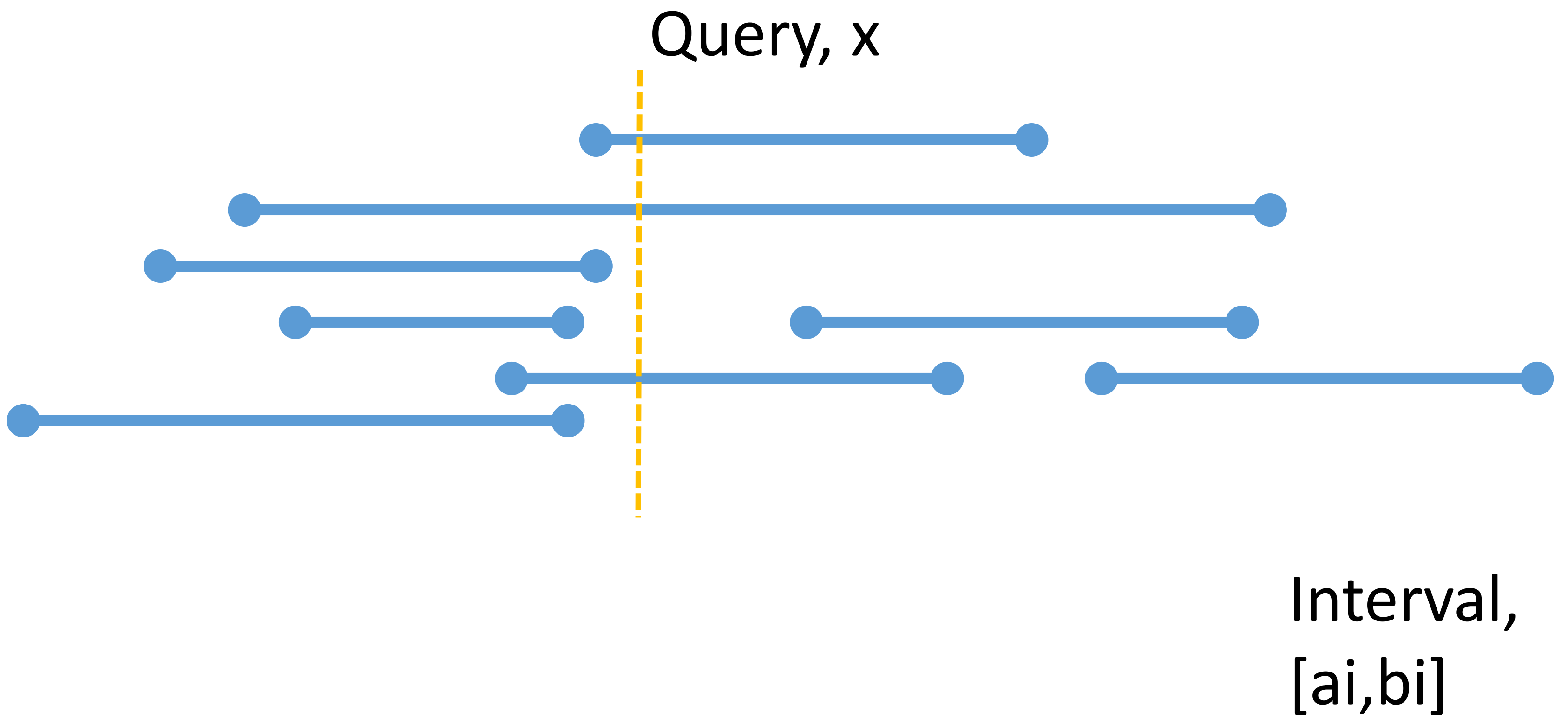
The Problem

- There are several interval
  - $S_i = [a_i,b_i]$
  - Known, static
- We want to ask
  - Give x
  - Find every segment s_i such that $x \in S_i$

  - There are several x that we would like to ask

Interval

- $S = [3,10] \Rightarrow \{x \mid 3 \le x \le 10\}$
  - Closed segment
- $S = (3,10) \Rightarrow \{x \mid 3 < x < 10\}$
  - Opened segment
- $S = [3,3] \Rightarrow \{3\}$
  - Point


Query, x
Interval,
[ai,bi]

Output Sensitive

- What is the size of the output of stabbing query?
  - O(N)?
- So, stabbing query is O(N) ??

We says it’s O(K)
Where K is the number of output

Interval Tree

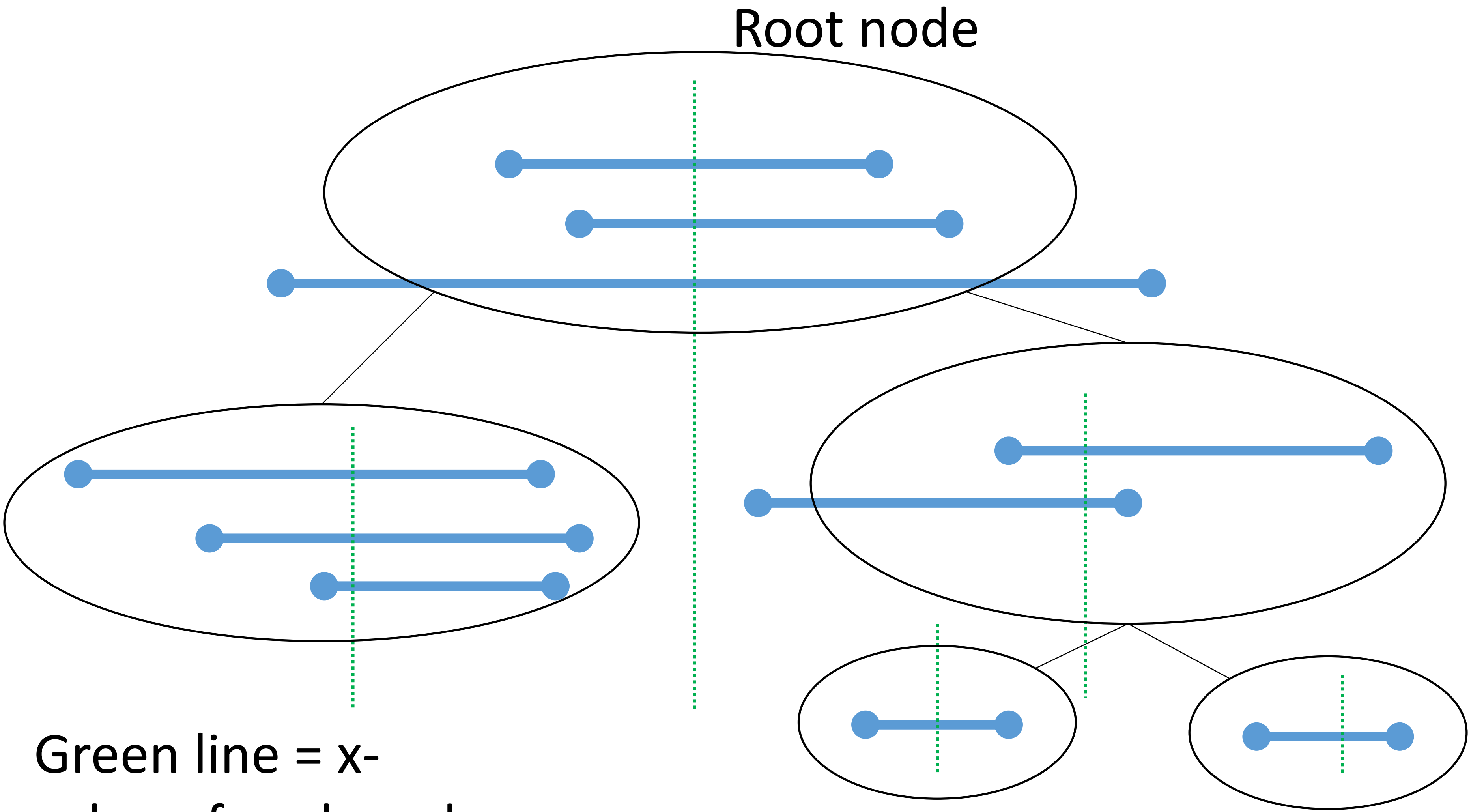
Key Idea

- Using a tree, similar to a binary search tree
  - Each node is identified by x-coordinate
  - Each node stores several intervals
    - In another kind of data structure
- Querying
  - Start at the root node and traverse to a leaf
  - For each visited node, check the stored intervals in the node contains the query x-coordinate
    - This is done efficiently by using the data structure stored in the node

Node

- Each node is associated with an a key X-value
  - The x-value at the root node should be the median of all endpoints
- If any interval intersect with the x-value of the node, that interval should be stored in the node
- If the right-end of an interval is to the left of the x-value, the interval should be stored in the left subtree
- If the left-end of an interval is to the right of the x-value, the interval should be stored in the left subtree

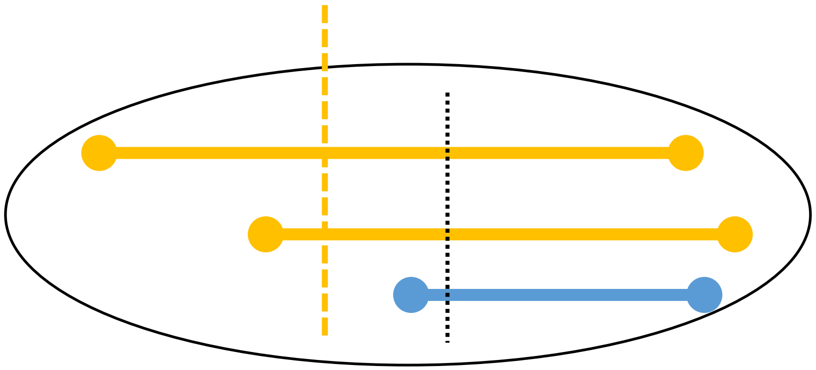
Example of an interval tree



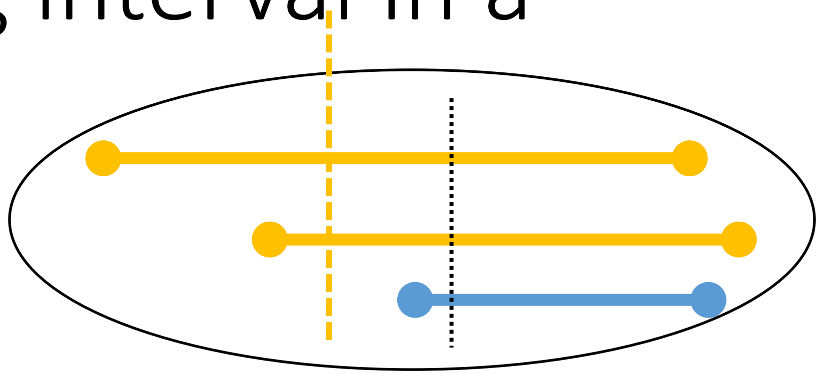
Green line = x-value of each node

Searching

- Start with the root node
- Let the query be x = q
- Report any interval in the node that intersect the query
  - Can be done by storing endpoints of interval (next slide)
- Recursively go to the child node
  - If q < x-value
    - Go to the left child
  - else
    - Go to the right child



Reporting intersecting interval in a node



- keep two lists on each node
  - One for left endpoints L_list
  - The other for right endpoints R_list
  - Both sorted by x
- Reporting intersecting interval is as simply as traverse the list from the one end
  - If q < x-value
    - Start from the leftmost endpoints of the L_list and traverse right until the x of endpoints is more than q, report any interval traverse
  - Else
    - Start from the rightmost endpoints of the R_list and traverse left until the x of endpoints is less than q, report any interval traverse

Conclusion

- Create an interval tree
  - O(n log n)
- Query Time
  - O(K + log n)
- Space
  - There are at most O(N) nodes
  - Each interval is stored in exactly one node
    - Space depends on the data structure that store the interval

Interval Tree

- Construction
  - Given int *a, int *b
- Operation
  - `void create(int n,int *a, int *b);`
    - `O(n log n)`
  - `void query(int x,int *ans,int &n);`
    - `O(log n + K)`

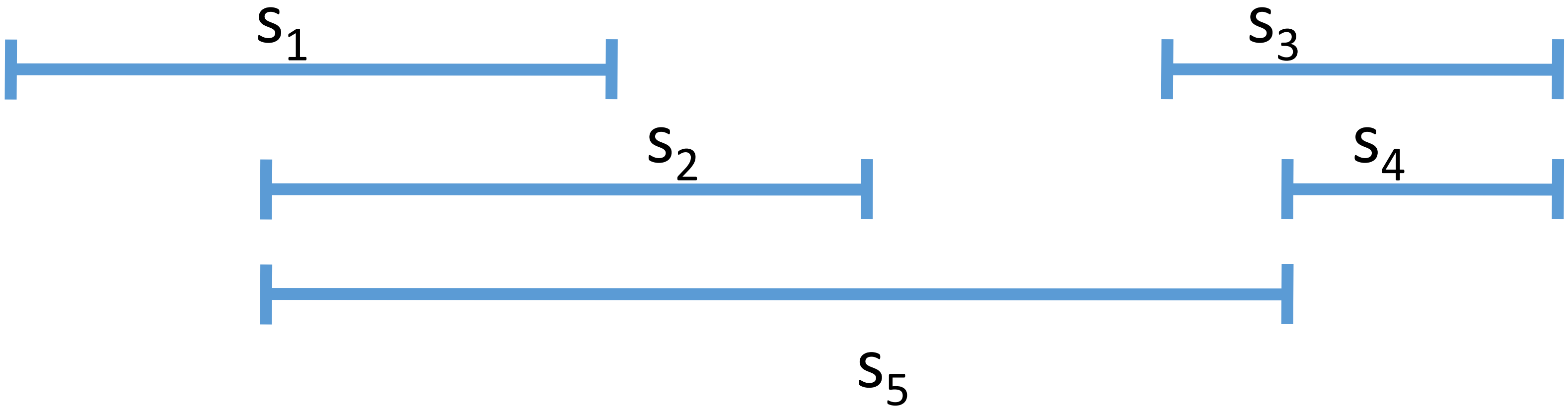
Segment Tree

Storage

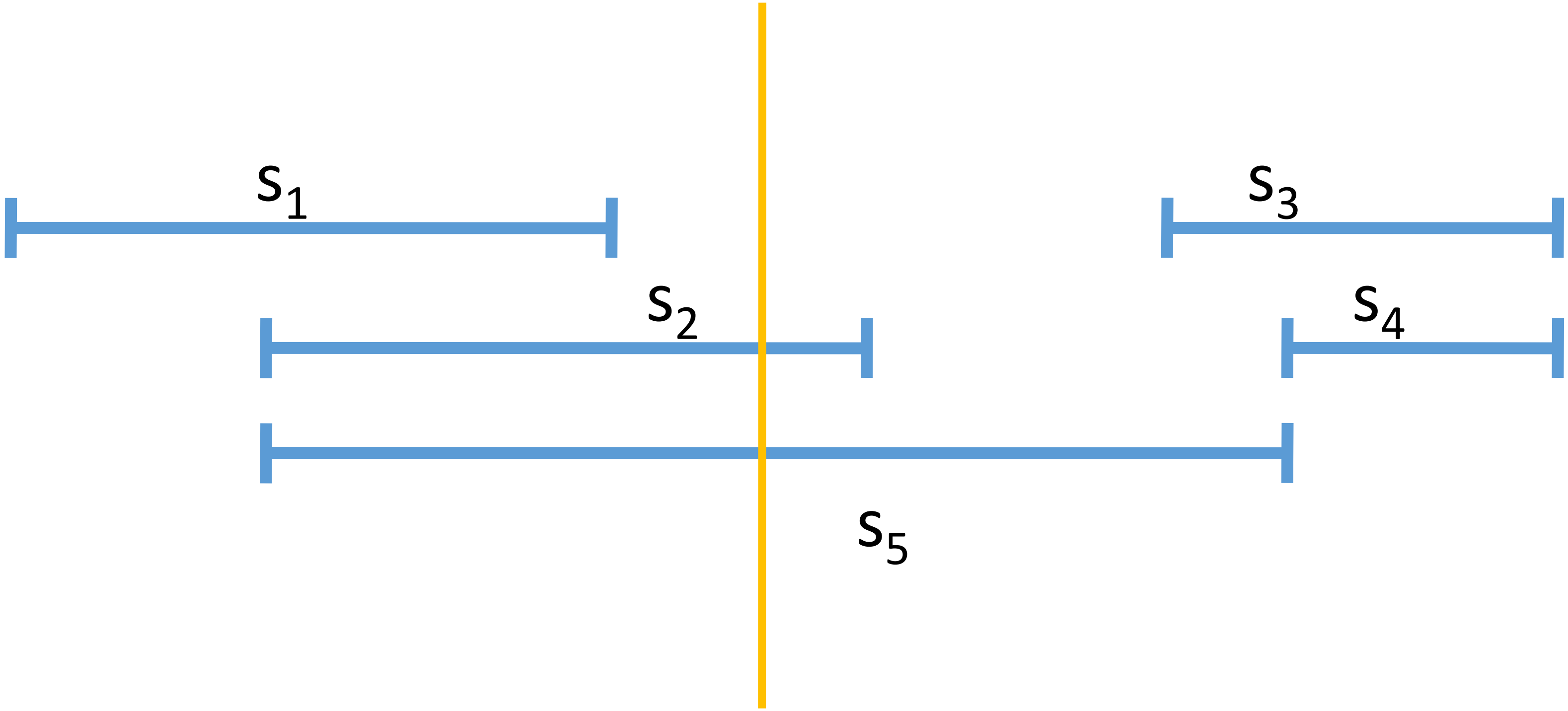
- Data
  - A binary Tree
  - Each node contain a set of intervals

- It uses O(n log n) space

Example of segments

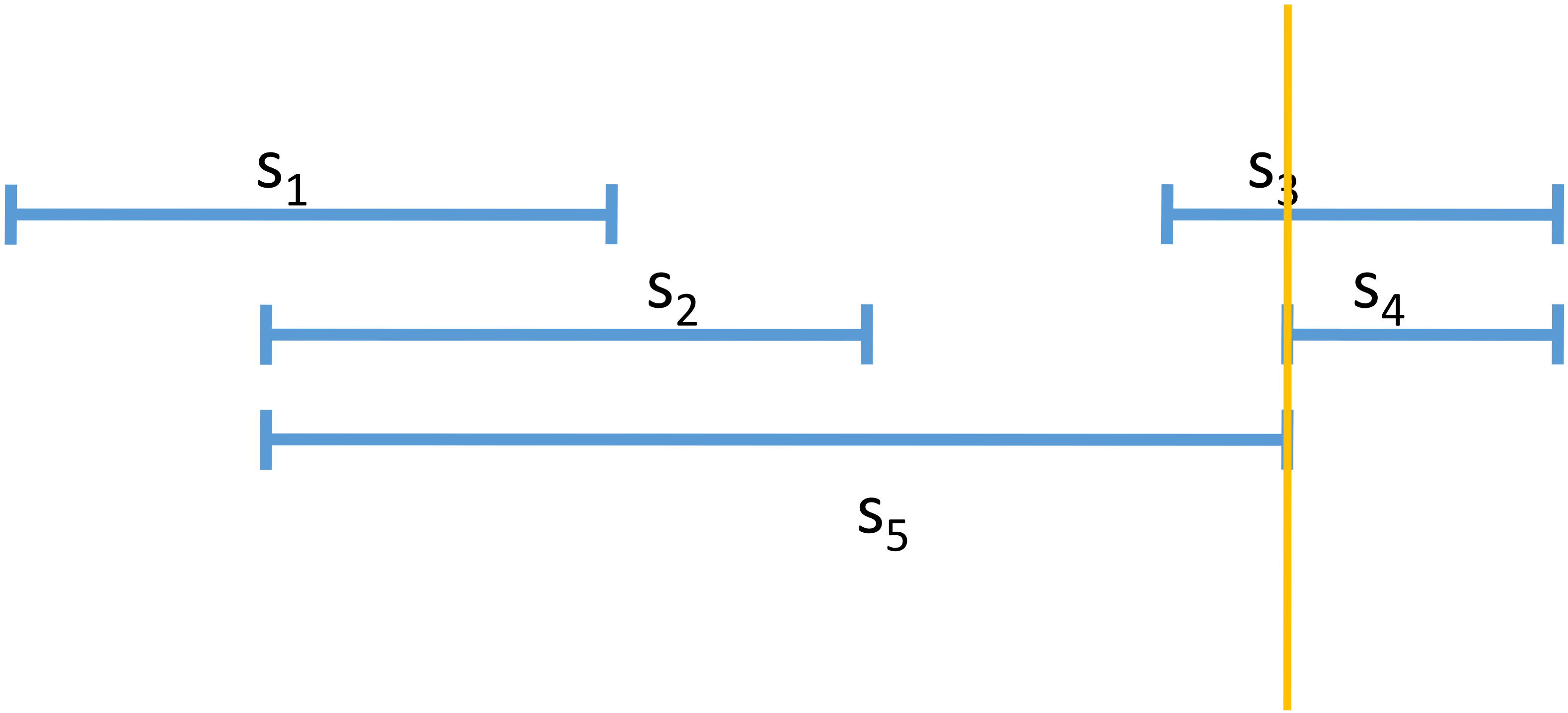


Example of queries



Answer = {2,5}

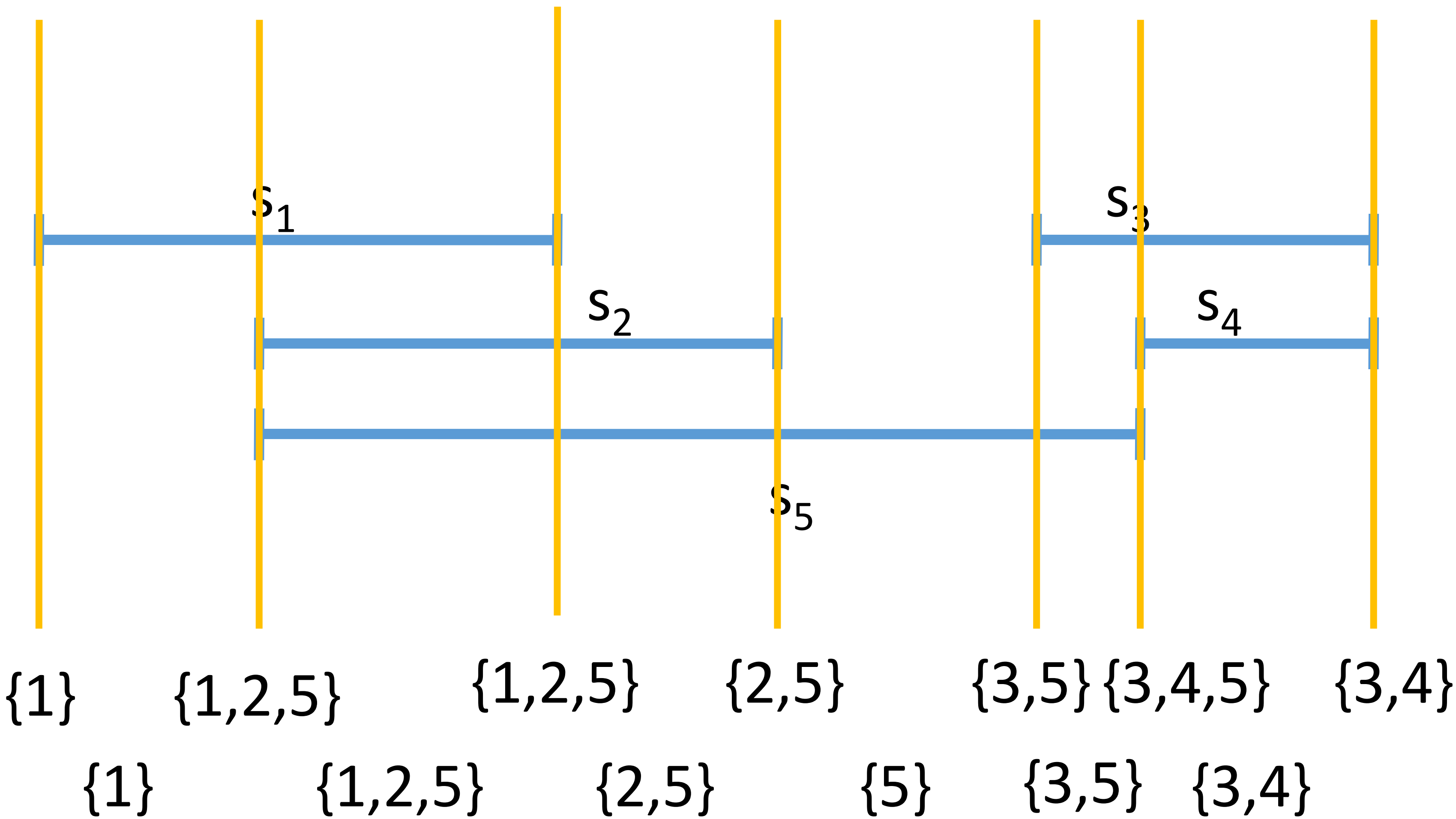
Example of queries



Answer = {3,4,5}

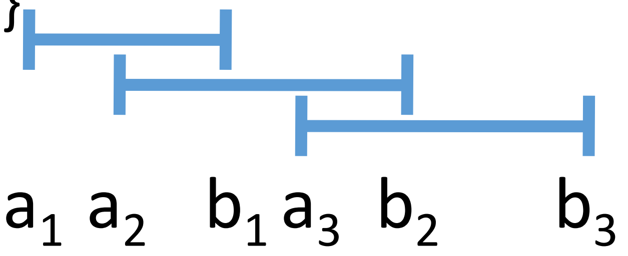
How it works

Answer (may) changes at the endpoints of segment only



Elementary Intervals (EI)

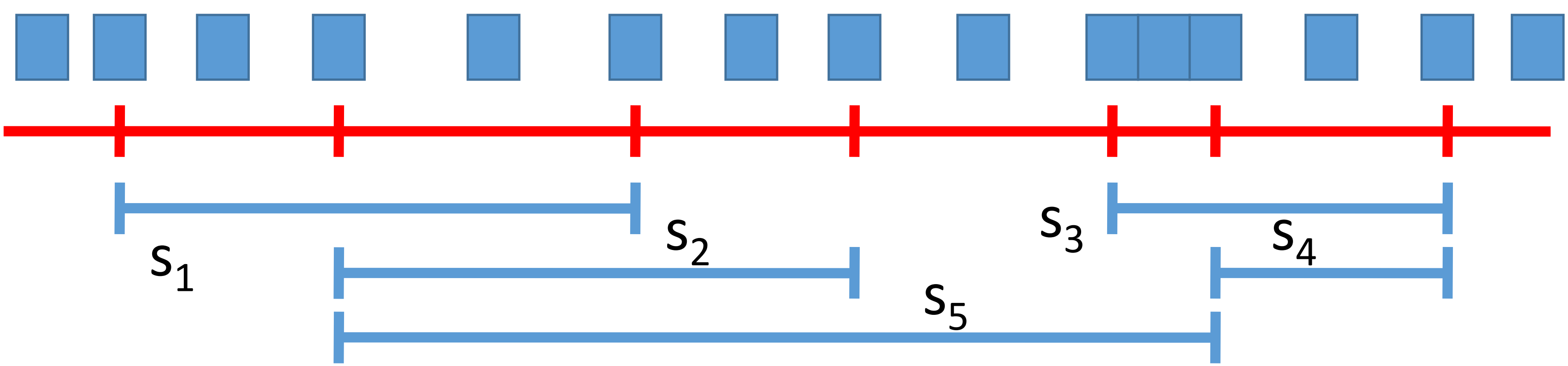
- e = array of 2N endpoints of $\{S_i\}$
  - eg, $S_1 = \{a_1,b_1\}$, $S_2 = \{a_2,b_2\}$, $S_3 = \{a_3,b_3\}$
  - $E = [a_1,b_1,a_2,b_2,a_3,b_3]$
- E* = Sorted E
  - eg, $E^* = [a_1,a_2,b_1,a_3,b_2,b_3]$



- EI = every interval between each element of E*
  - And every point of E* itself (point is also an interval)
  - sorted
- Eg, $EI = (-\infty,a_1), [a_1,a_1], (a_1,a_2), [a_2,a_2], (a_2,b_1), [b_1,b_1]\ldots$

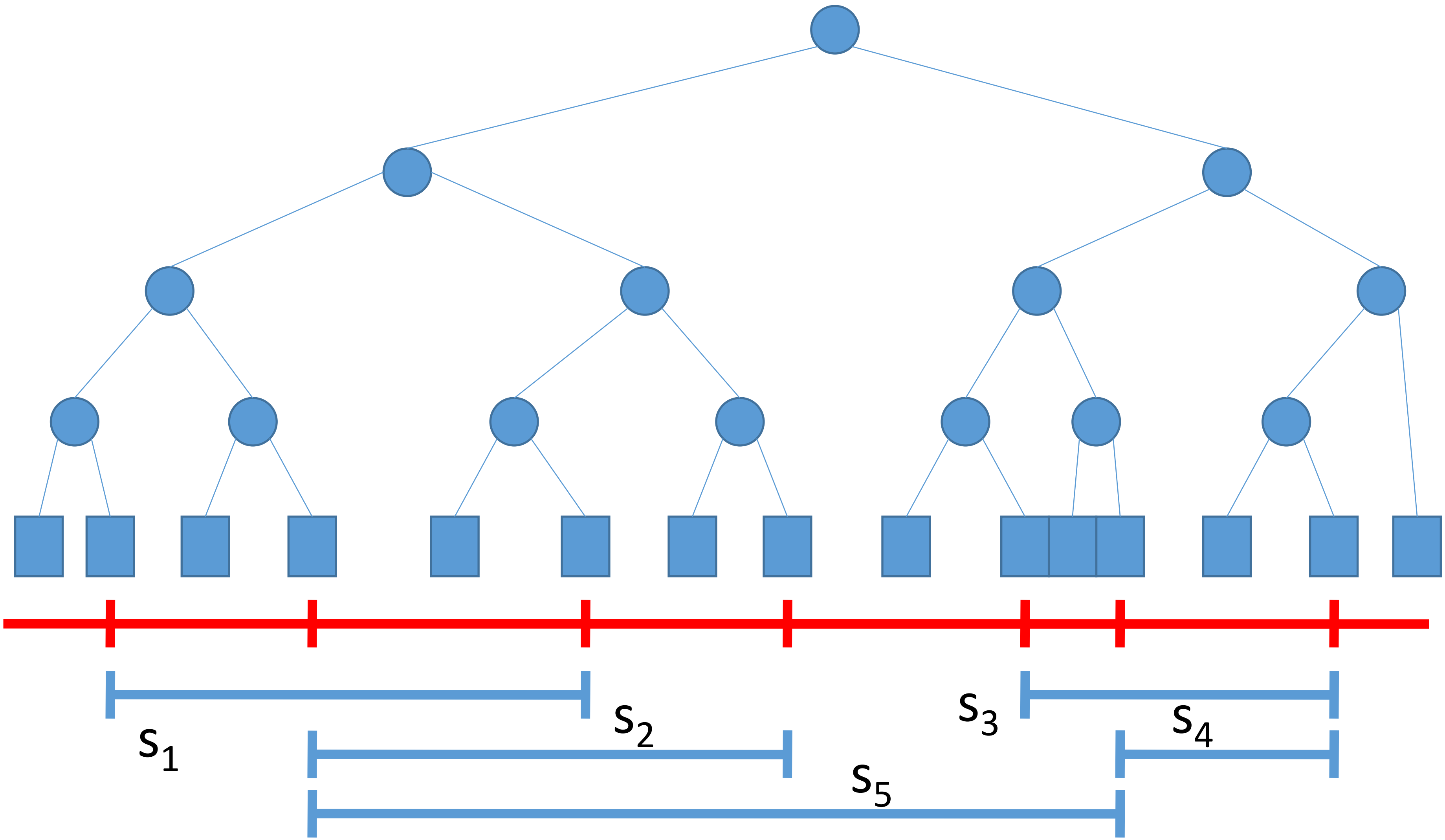
Segment Tree

- Each leaf= elementary interval



Segment Tree

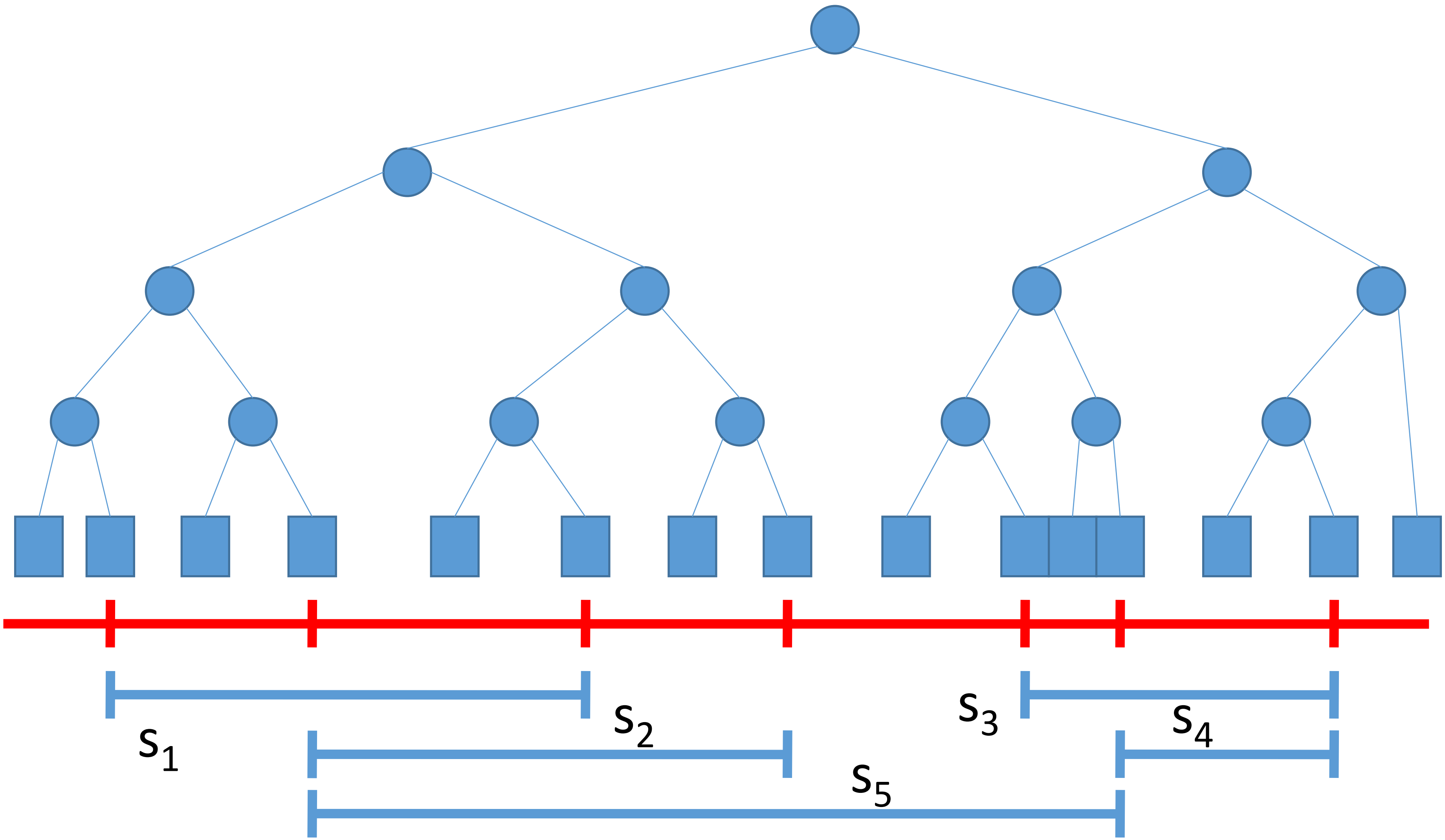
Each internal node
Union of child intervals



Internal Node Construction

- Construct internal node progressively from bottom to top
  - Pairing adjacent child
- Notice that
  - child does not intersect!
  - Interval of child is a subset of the interval of the root
- Hence, Each node in the tree corresponds to an interval
  - Let u be the node I(u) is our the interval associated with the node

Construction

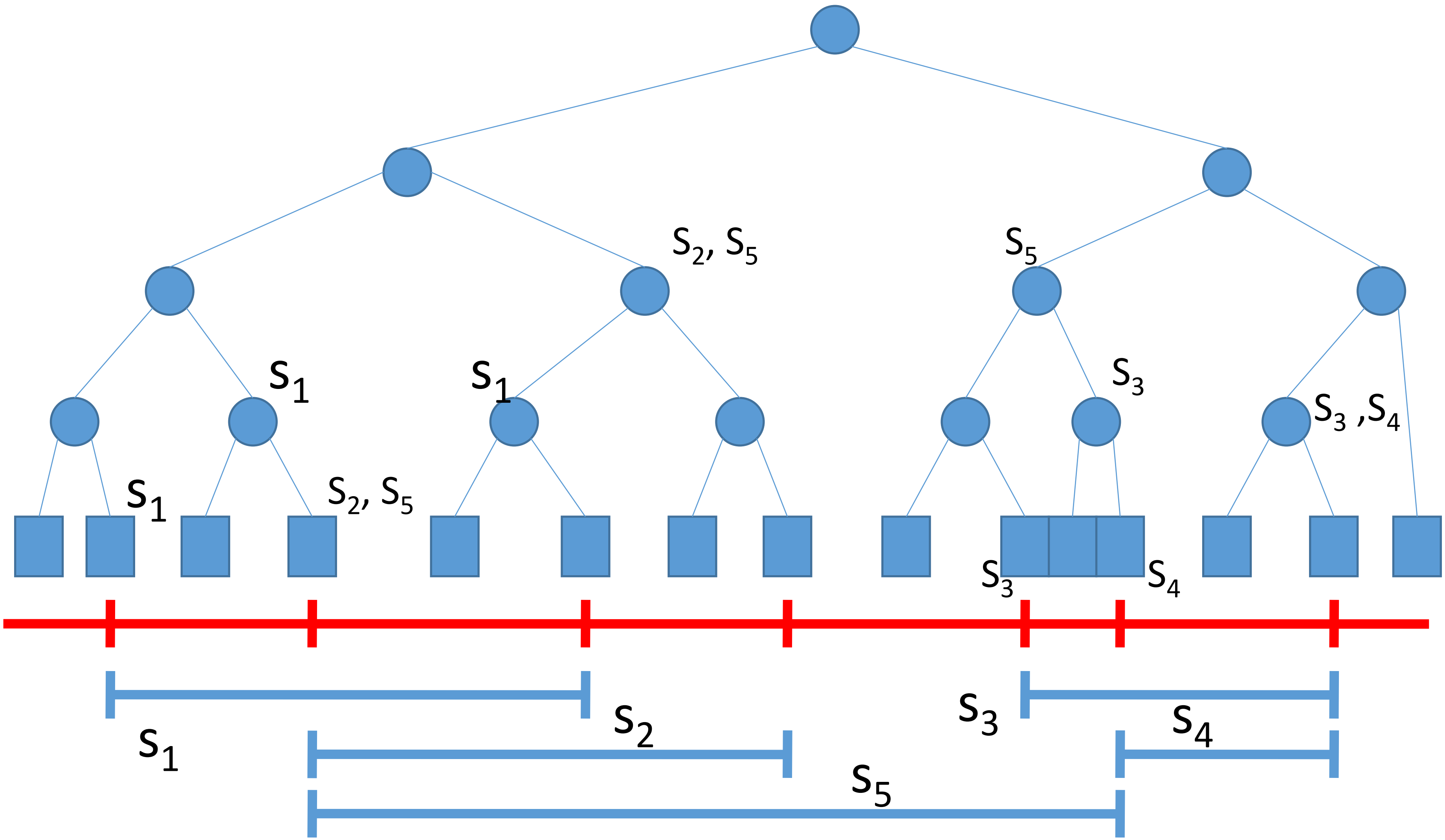


Canonical List

- Each node, additionally, store a list of ***input segment***

- The segment is stored in the canonical list of node u only when I(u) is a subset of the segment
  - Do not store in the children if the parent has it

Canonical List



Tree Summary

- Given segments, we can construct tree
- Each leaves is the elementary interval
- Each internal node is the union of the leaves
  - i.e., each node is associated with an interval

- Each node stores canonical list, a list of segment that cover the interval
  - i.e., the interval is a subset of the segment
  - Do not store in the children

Answering Stabbing Query

- Start at root
- Recursively go through child that intersect the query
  - Report everything in canonical list of node in the path

- O(log n + K)

Construction of the Canonical List

- Construct a tree first, without any canonical list
- Insert every segment into the list

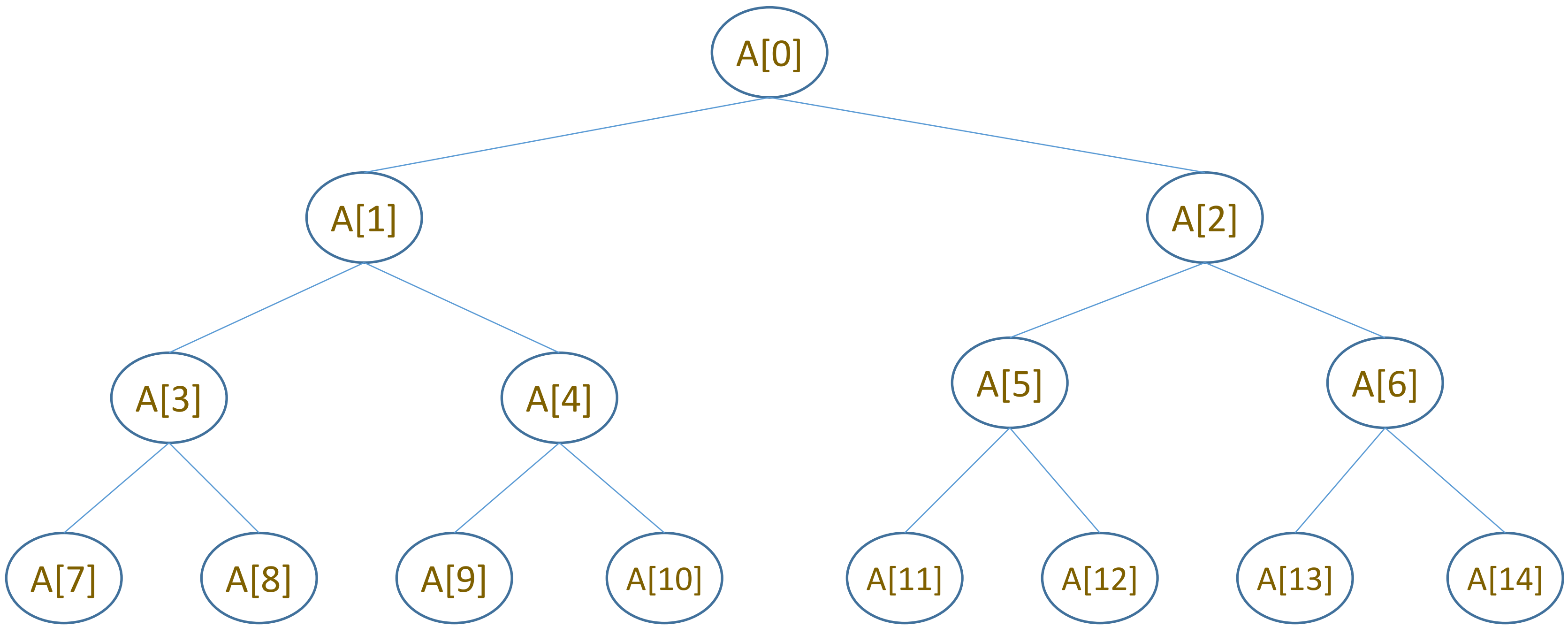
- Insertion each segment
- Start at root node
- Recursively process children that the segment intersect
  - If the interval of each node is a subset of the segment, store the segment into the canonical list

Construction Cost

- Each segment can be insert in O(log N)
- At each level in the tree, at most 4 nodes are considered
  - The considered node will form continuous coverage
  - Only left most and right-most node will recurs, the middle nodes will just store the segment
- Each segment might “appear” in the canonical list of at most 2 nodes per level
- O(N log N) space requirement

Tree Structure

- Use Heap style
- Because the structure of the tree does not change



Query Code

```
void query(node u,float x, int *ans, int &count) {
    append_canonical(u, ans, n);
    if u.isLeaf() == false
        if is_intersect(x, interval( u.left ) )
            query(u.left, x, ans, count);
        else
            query(u.right, x, ans, count);
}
```

Create

```
void create(int n, int *begin, int *end) {
    CreateTree(n, begin, end);
    for (int i = 0;i < n;i++) {
        insert(root, segment(begin[i], end[i] );
    }
}
```

Insert Code

```
void insert(node u, segment s) {
    if is_subset(interval(u), s)
        store_canonical(u, s)
    else {
        if is_intersect(s, interval( u.left ) )
            insert(u.left, s);
        if is_intersect(s, interval( u.right ) )
            insert(u.right, s);
    }
}
```

Comparison with Interval Tree

	Segment Tree	Interval Tree
Key Idea	Node = coverage Node stores “answer”	Node = separation Node stores likely candidate
Pre-process	O(n log n)	O(n log n)
Query	O(log n + K)	O(log n + K)
Space	O(n log n)	O(n)
Higher Dimension?	Fairly simple	Complex

Higher Dimension?

- Segment Tree
  - Node consider only y-value
  - The canonical list of each node is another segment tree
    - In the sub-segment tree, we consider the x-value
  - Just like range tree
- Interval Tree
  - Complex…

Range Minimum Query Problem

RMQ problem

- Given an array A of integer
- What is the minimal value in A[p] … A[q]?

RMQ naïve

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

$$\underbrace{A[2] \ldots A[7]}_{\min(2,7) = a[5]}$$

min(2,7) = a[5]

Preprocess = $O(N^2)$
Query = $O(1)$

RMQ better

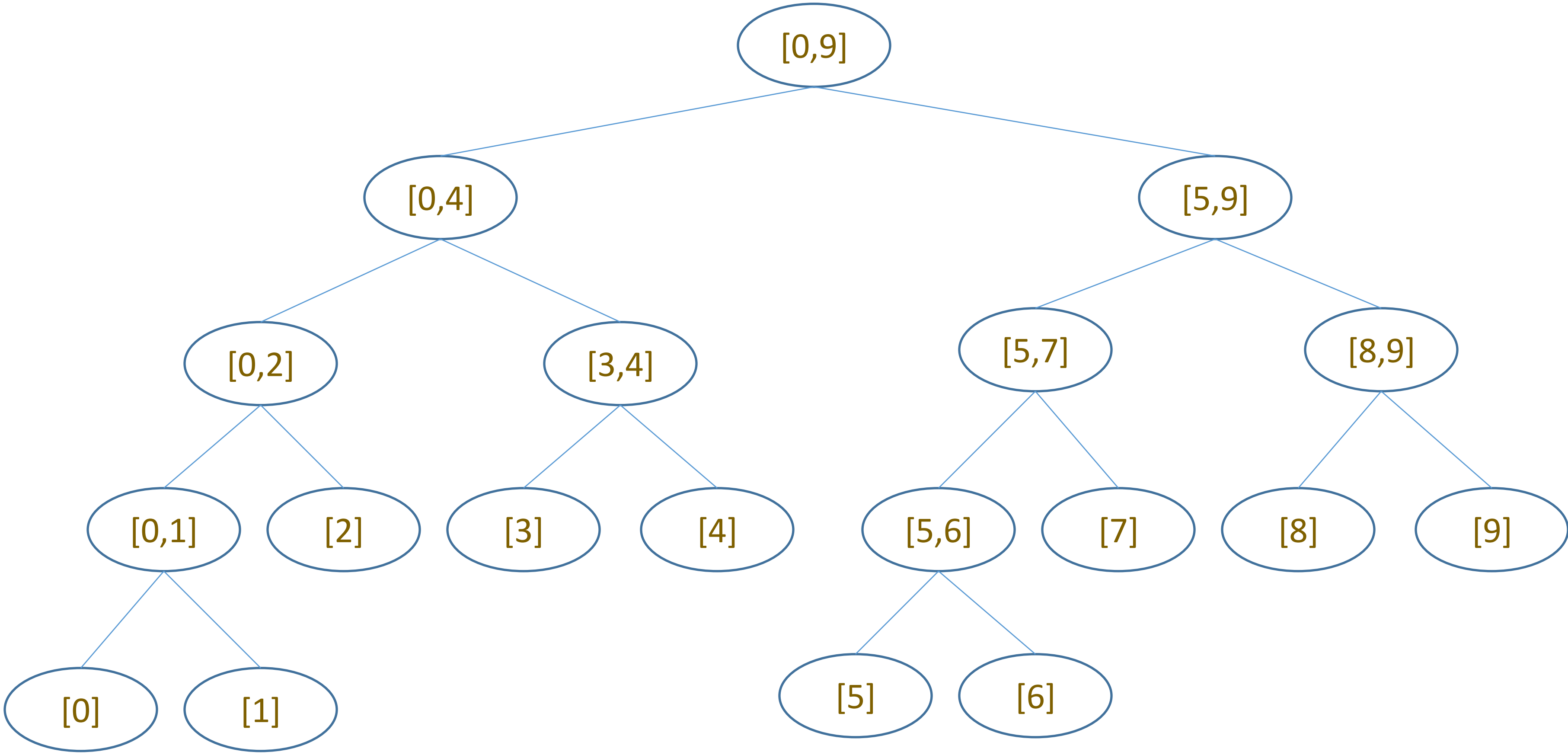
M[0] = A[0]   M[1] = A[5]   M[2] = A[6]   M[3]

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

min(2,7) =
min(M[1],
A[2],
A[6],A[7])

Preprocess = O(N)
Query = $O(N^{0.5})$

RMQ segment Tree



Preprocess = O(N log N)
Query = O(log N)

A segment tree for the interval **[0, 9]**