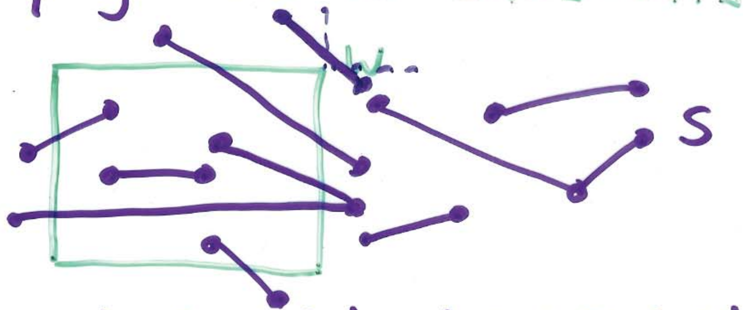
Windowing Problem

Given: A set S of n (non-intersecting) line segments in the plane

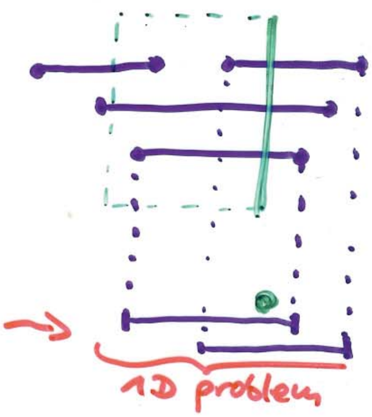
Task: Process S into a data structure such that the following windowing query can be answered efficiently: Report all segments in S that intersect a given query window $W := [x, x'] \times [y, y']$



Segments having at least one endpoint in W can be found by range queries in range trees
→ $O(\log n + k)$ time (with fractional cascading)

Subproblem I (for horizontal segments)

Process a set of horizontal line segments s.t. segments intersecting a vertical query segment can be reported efficiently.
⇝ Consider query line instead of segment →



Subproblem II (1 dimensional):

Given: A set $I = \{[x_1, x_1'], \ldots, [x_n, x_n']\}$ of intervals in $\mathbb{R}$

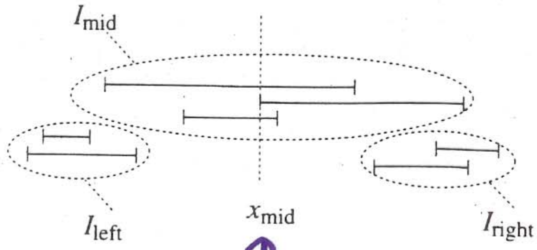
Task: Process I into a data structure which supports queries of the type: Report all intervals that contain a query point

→ Interval trees

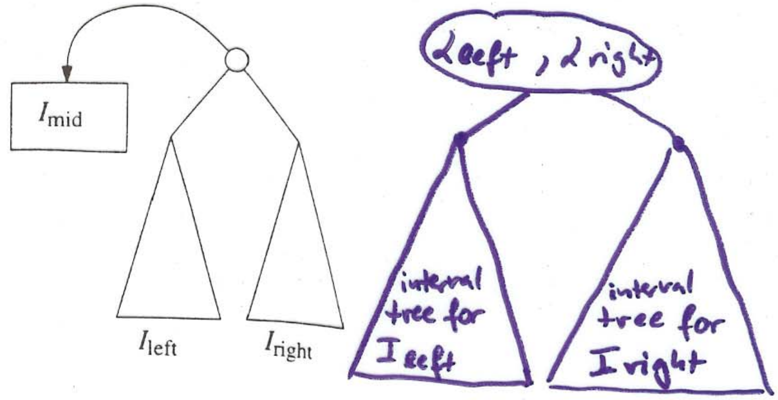
→ Segment trees

Interval Trees

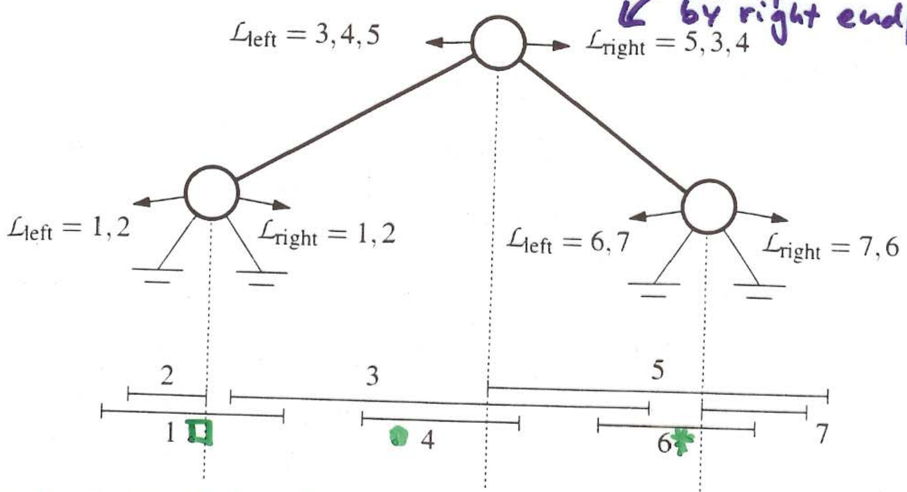
$I = I_{left} \,\dot\cup\, I_{mid} \,\dot\cup\, I_{right}$



median of 2n interval endpoints



List of intervals in I_{mid} sorted by right endpoint



Interval Tree

Lemma: An interval tree on a set of n intervals uses $O(n)$ storage and has depth $O(\log n)$.

Proof: Each interval is stored in a set I_{mid} only once $\rightarrow O(n)$ storage.

Algorithm CONSTRUCTINTERVALTREE(I)
Input. A set I of intervals on the real line.
Output. The root of an interval tree for I.

1. **if** $I = \emptyset$
2.   **then return** an empty leaf
3.   **else** Create a node ν. Compute x_{mid}, the median of the set of interval endpoints, and store x_{mid} with ν.
4.   Compute I_{mid} and construct two sorted lists for I_{mid}: a list $\mathcal{L}_{left}(\nu)$ sorted on left endpoint and a list $\mathcal{L}_{right}(\nu)$ sorted on right endpoint. Store these two lists at ν.
5. $lc(\nu) \leftarrow$ CONSTRUCTINTERVALTREE(I_{left})
6. $rc(\nu) \leftarrow$ CONSTRUCTINTERVALTREE(I_{right})
7.   **return** ν

Time analysis: $O(\cancel{n} + |I_{mid}| \cdot \log |I_{mid}|)$ per vertex $\Rightarrow O(n \log n)$
$|I|$

Algorithm QUERYINTERVALTREE(ν, q_x)

Input. The root ν of an interval tree and a query point q_x.

Output. All intervals that contain q_x.

```
if ν is not a leaf
   then if q_x < x_mid(ν)
           then Walk along the list L_left(ν), starting at the interval with the
                leftmost endpoint, reporting all the intervals that contain
                q_x. Stop as soon as an interval does not contain q_x.
                QUERYINTERVALTREE(lc(ν), q_x)
           else Walk along the list L_right(ν), starting at the interval with the
                rightmost endpoint, reporting all the intervals that contain
                q_x. Stop as soon as an interval does not contain q_x.
                QUERYINTERVALTREE(rc(ν), q_x)
```

Time analysis:

- $O(1+k_v)$ time at vertex v; $k_v :=$ #intervals reported at v
- visit ≤ 1 node at any depth

$\rightsquigarrow O(\log n + k)$

Theorem: An interval tree for a set of n intervals ~~in the plane~~ can be constructed in $O(n \log n)$ time and uses $O(n)$ storage. All intervals that contain a query point can be reported in $O(\log n + k)$ time; $k =$ #reported intervals.

Segment Trees

Let $p_1, p_2, \cdots, p_m$ the sorted list of distinct interval endpoints of I

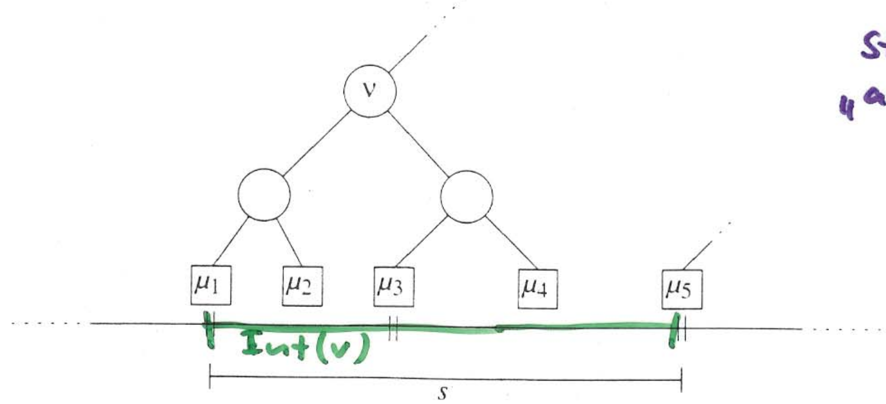
$\rightsquigarrow$ Consider partitioning into elementary intervals

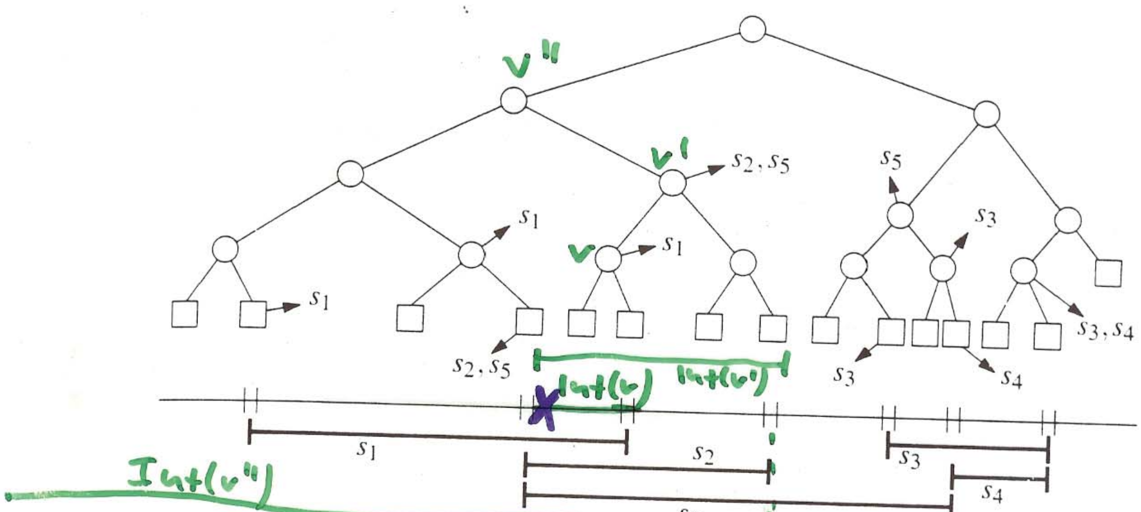
$(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, (p_{m-1}, p_m), [p_m, p_m], (p_m, \infty)$

- Balanced binary search tree $\mathcal{T}$ with leaves corresponding to elementary intervals
- $\mathrm{Int}(\mu) :=$ elementary interval corresponding to leaf μ
  $\mathrm{Int}(v) :=$ union of $\mathrm{Int}(\mu)$ of all leaves in subtree rooted at v
- Each node or leaf v stores
  - $\mathrm{Int}(v)$
  - the canonical subset $I(v) \subseteq I$:

$$I(v) := \{ [x, x'] \in I \mid \mathrm{Int}(v) \subseteq [x, x'] \text{ and } \mathrm{Int}(\mathrm{parent}(v)) \nsubseteq [x, x'] \}$$

Store intervals „as high as possible".





Lemma: A segment tree on n intervals uses $O(n \log n)$ storage

Proof idea: Any interval is stored in a set $I(v)$ for at most two nodes at the same level of $\mathcal{T}$.

Algorithm QUERYSEGMENTTREE(ν, q_x)
Input. The root of a (subtree of a) segment tree and a query point q_x.
Output. All intervals in the tree containing q_x.

Report all the intervals in $I(\nu)$.
if ν is not a leaf
  then if $q_x \in \mathrm{Int}(lc(\nu))$
      then QUERYSEGMENTTREE($lc(\nu), q_x$)
      else QUERYSEGMENTTREE($rc(\nu), q_x$)

Time analysis: • Visit one node per level
• Spend $O(1+k_\nu)$ per node ν
$\}$ $O(\log n + k)$ time

Segment_Tree_Construction:

1) Sort interval endpoints of I $\rightsquigarrow$ elementary intervals
2) Construct balanced bin. search tree on elem. intervals $\}$ $O(n \log n)$
3) Determine $\mathrm{Int}(v)$ bottom-up $O(n)$
4) Compute canonical subsets by incrementally inserting the intervals $[x, x'] \in I$ into $\mathcal{T}$, using Insert Segment Tree

Algorithm INSERTSEGMENTTREE($\nu, [x : x']$)
Input. The root of a (subtree of a) segment tree and an interval.
Output. The interval will be stored in the subtree.

if $\mathrm{Int}(\nu) \subseteq [x : x']$
  then store $[x : x']$ at ν
  else if $\mathrm{Int}(lc(\nu)) \cap [x : x'] \neq \emptyset$
      then INSERTSEGMENTTREE($lc(\nu), [x : x']$)
    if $\mathrm{Int}(rc(\nu)) \cap [x : x'] \neq \emptyset$
      then INSERTSEGMENTTREE($rc(\nu), [x : x']$)

Time analysis: • Spend constant time per node
• If we don't store $[x, x']$ at v, then $x \in \mathrm{Int}(v)$ or $x' \in \mathrm{Int}(v)$
• Each interval stored $\leq$ twice at each level.
At most one node per level whose interval contains x (similar for x')
$\rightsquigarrow$ Visit ≤ 4 nodes per level $\Rightarrow O(\log n) \Rightarrow O(n \log n)$ together

Theorem: A segment tree for a set of n intervals can be built in $O(n \log n)$ time and uses $O(n \log n)$ storage.
All intervals that contain a query point can be reported in $O(\log n + k)$ time.

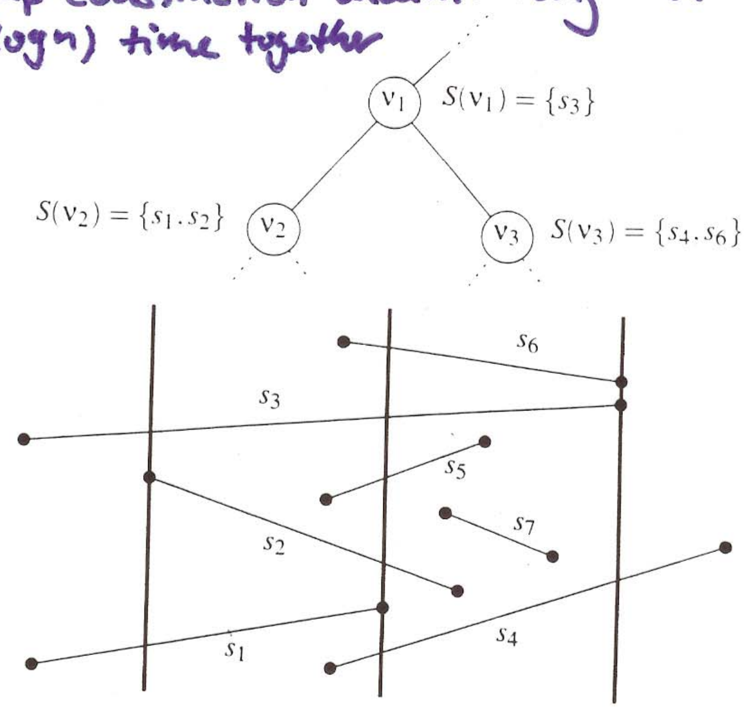
2D Windowing Revisited

Given: A set S of n disjoint segments in the plane

Task: Process S into a data structure such that all segments intersecting a vertical query segment $q := q_x \times [q_y, q_y']$ can be reported efficiently

- Build segment tree T based on x-intervals of segments in S
  $\rightsquigarrow$ each $Int(v) \cong Int(v) \times (-\infty, \infty)$ vertical slab
- $I(v) \cong S(v)$ canonical subset of segments spanning vertical slab at v
- Store $S(v)$ in binary search tree $T(v)$ based on vertical order of segments

Analysis:
- Storage $O(n \log n)$
- Bottom-up construction maintaining vertical order of segments
  $\rightsquigarrow O(n \log n)$ time together



Query algorithm:
- Search regularly for q_x in T
- In every visited node v report segments in $T(v)$ between q_y and q_y' (1D range query)
  $\rightarrow O(\log n + k_v)$ time for $T(v)$ $\rightsquigarrow O(\log^2 n + k)$ altogether

Theorem: Let S be a set of (interior-) disjoint segments in $\mathbb{R}^2$. The segments intersecting a vertical query segment (or an axis-parallel rectangular query window) can be reported in $O(\log^2 n + k)$ time, with $O(n \log n)$ preprocessing time and $O(n \log n)$ storage.