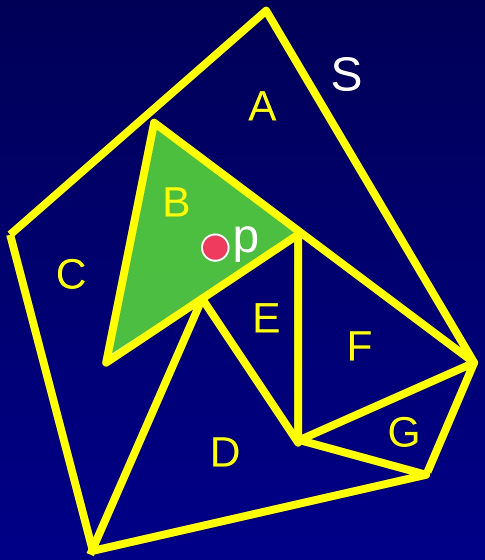
Computational Geometry

Chapter 6

Point Location

Problem Definition

- Preprocess a planar map *S*. Given a query point *p*, report the face of *S* containing *p.*



- **Goal:** O(*n*)-size data structure that enables O(log *n*) query time.

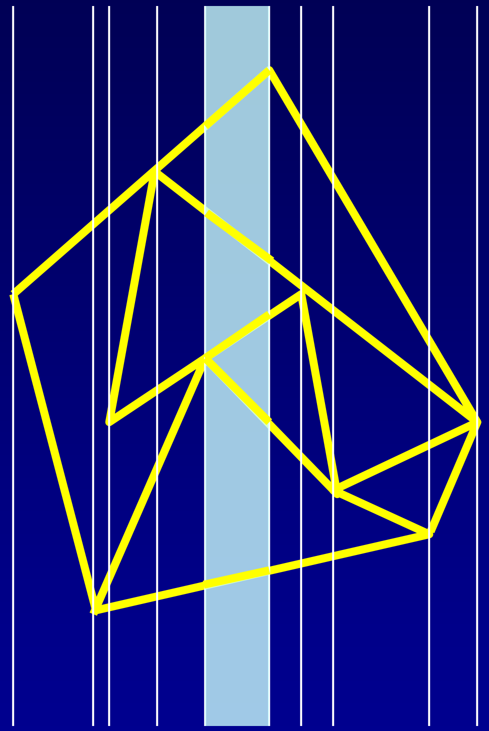
- **Application:**
  Which state is Baltimore located in?
  Answer: Maryland

- **Trivial Solution:** O(*n*) query time, where *n* is the complexity of the map.
  (Question: Why is the query time only O(*n*)?)

Naïve Solution

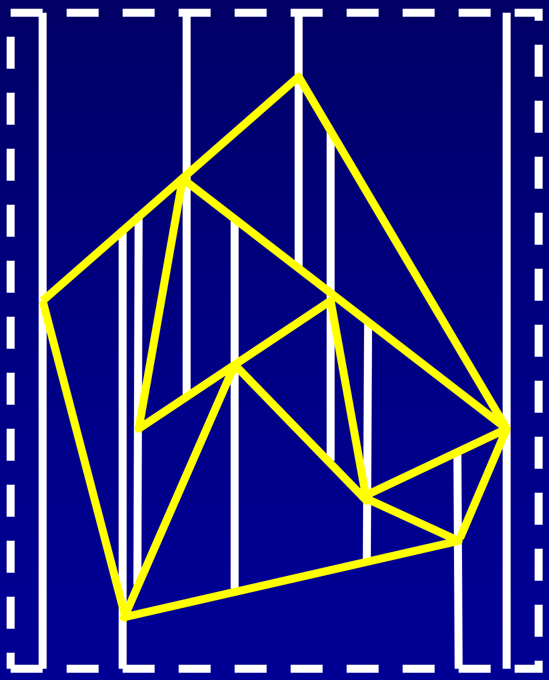
- Draw vertical lines through all the vertices of the subdivision.
- Store the *x*-coordinates of the vertices in an ordered binary tree.
- Within each slab, sort the segments separately along *y*.
- Query time: $O(\log n)$.
- **Problem**: Too delicate subdivision, of size $\Theta(n^2)$ in the worst case.

(Give such an example!)

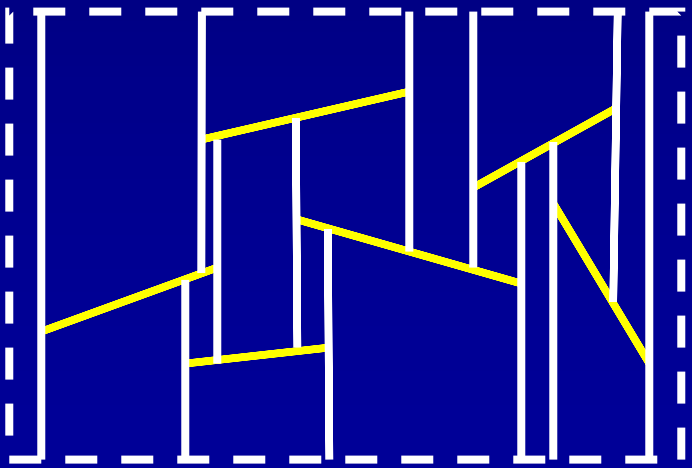


The Trapezoidal Map

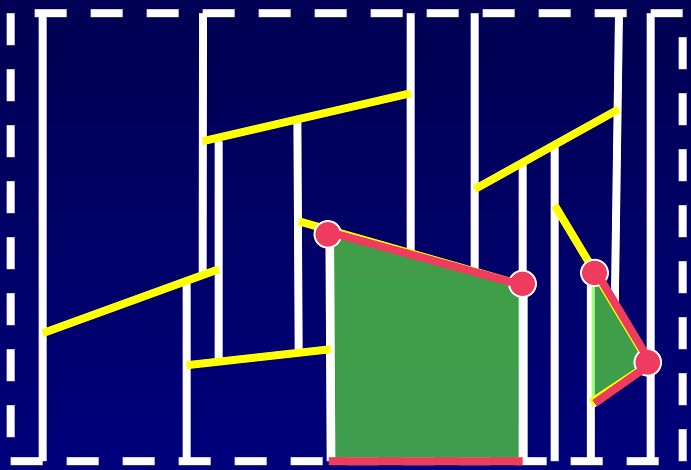
- Construct a bounding box.
- Assume general position: unique x coordinates.

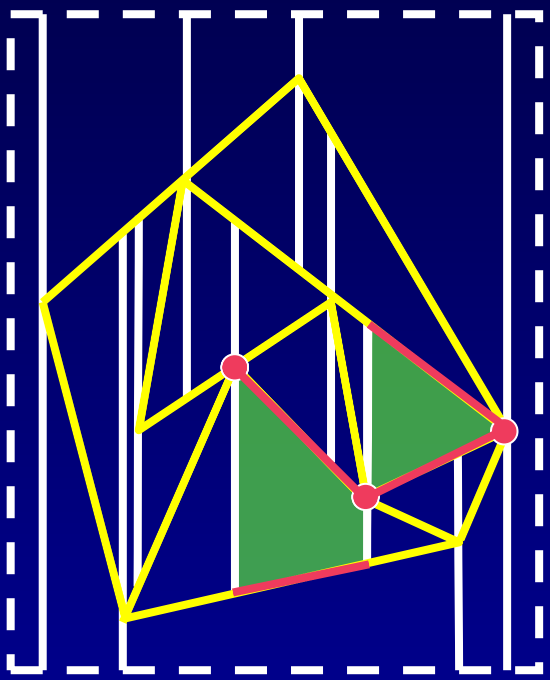


- Extend upward and downward the vertical line from each vertex until it touches another segment.
- This works also for noncrossing line segments.



Properties





- Contains triangles and trapezoids.
- Each trapezoid or triangle is determined:
  - By two vertices that define vertical sides; and
  - By two segments that define nonvertical sides.
- More *sensitive* than the original subdivision.

Notation

Every trapezoid (or triangle) Δ is defined by

- Left(Δ): a segment endpoint (right or left);
- Right(Δ): a segment endpoint (right or left);
- Top(Δ): a segment;
- Bottom(Δ): a segment.

Complexity

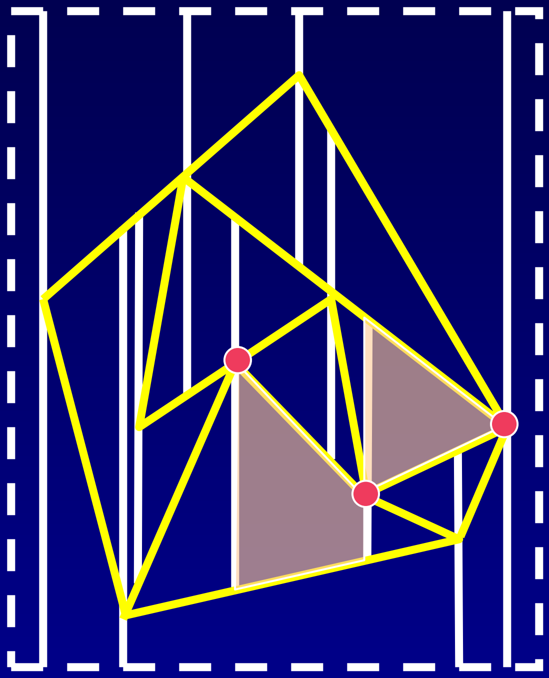
- **Theorem** (linear complexity)**:** A trapezoidal map of n segments contains at most $6n+4$ vertices and at most $3n+1$ faces.
- **Proof:**

1. Vertices:

$$\underbrace{2n}_{\text{original}} + \underbrace{4n}_{\text{extensions}} + \underbrace{4}_{\text{box}} = 6n + 4$$

2. Faces: Count Left(Δ).

$$\underbrace{2n}_{\text{left e.p.}} + \underbrace{n}_{\text{right e.p.}} + \underbrace{1}_{\text{box}} = 3n + 1$$



Question:

Why does the proof hold for degenerate situations?

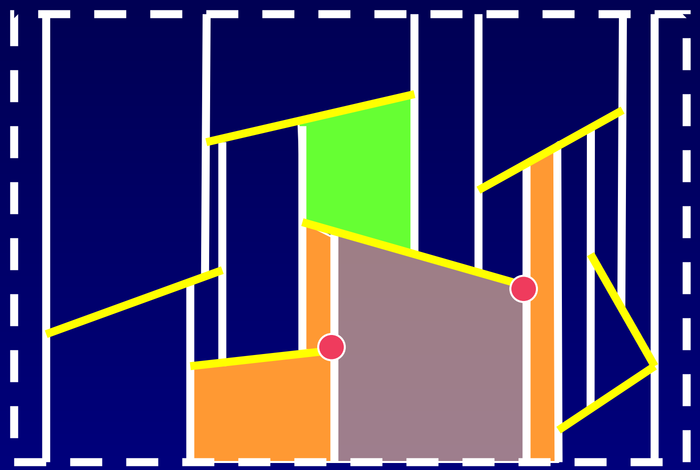
Map Data Structure

- Possibly by DCEL.

An alternative:

For each trapezoid store:

- The vertices that define its right and left sides;
- The top and bottom segments;
- The (up to *two*) neighboring trapezoids on right and left;
- (Optional) The neighboring trapezoids from above and below. This number might be linear in *n*, so only the leftmost of these trapezoids is stored.

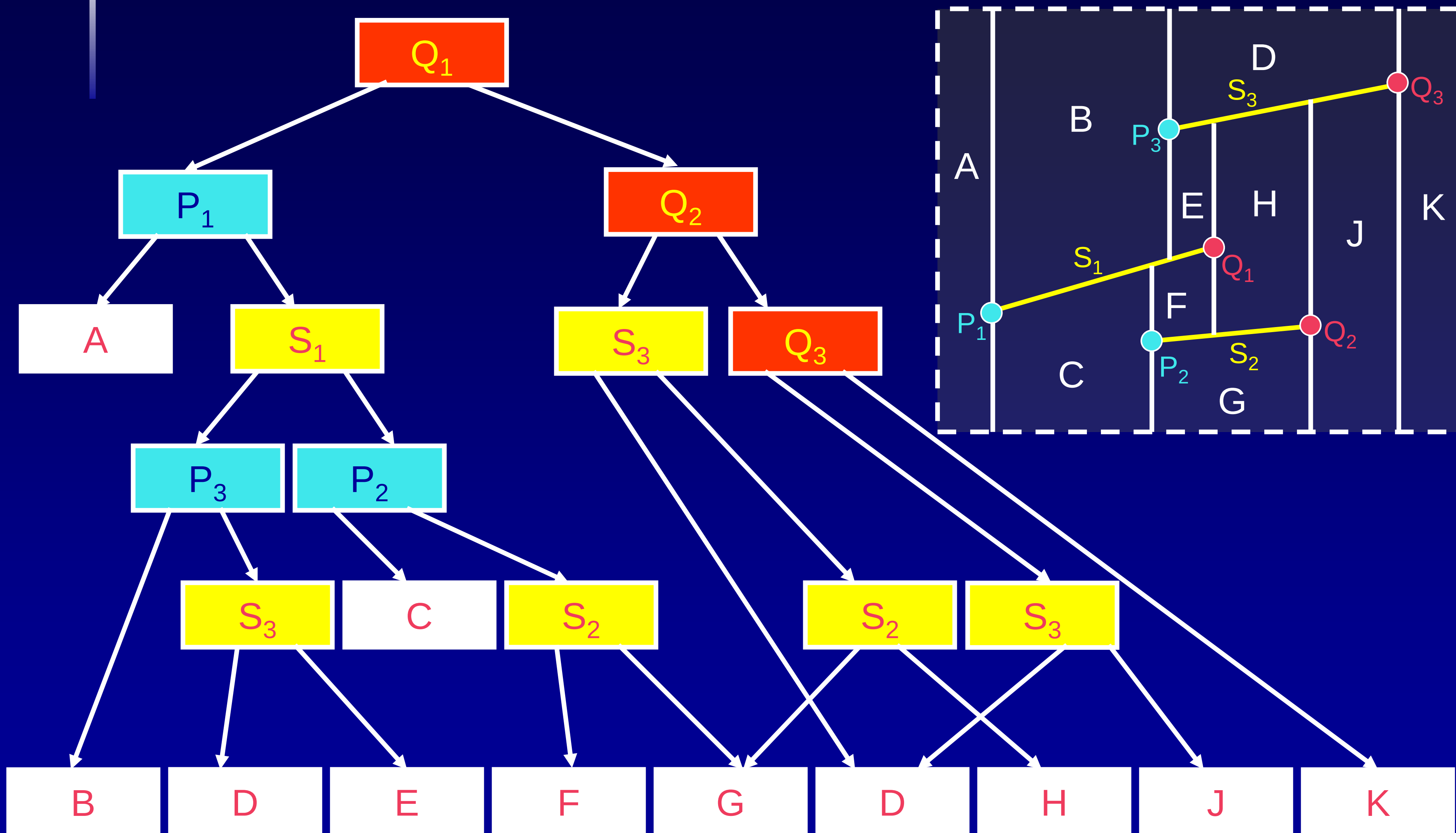


Note: Computing any trapezoid from the trapezoidal structure can be done in constant time.

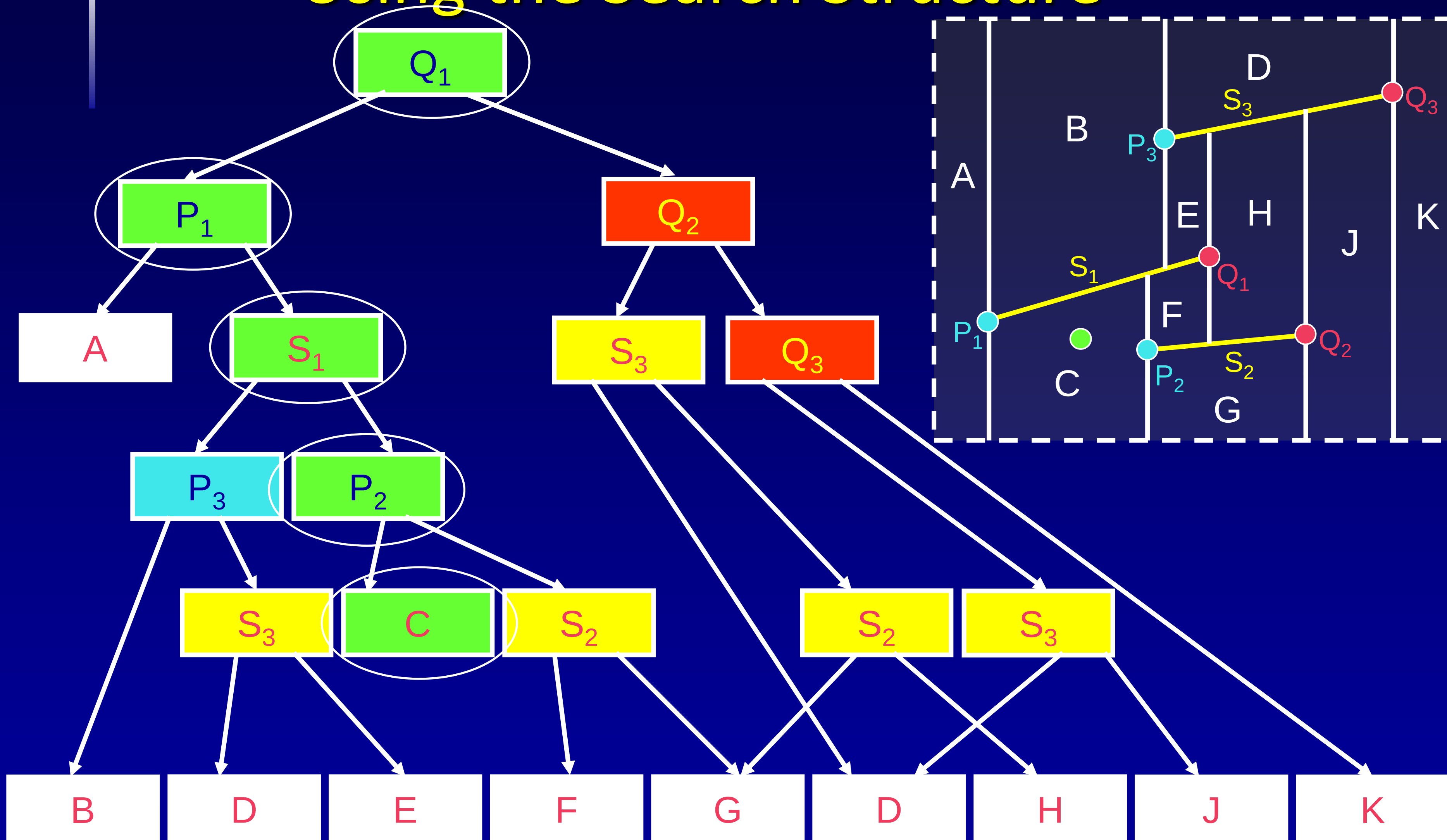
Search Structure: Branching Rules

- Query point *q*, search-structure node *s*.
- *s* is a segment **endpoint**:
  - *q* is to the **right** of *s*: go **right**;
  - *q* is to the **left** of *s*: go **left**;
- *s* is a segment:
  - *q* is **below** *s*: go **right**;
  - *q* is **above** *s*: go **left**;

The DAG Search Structure

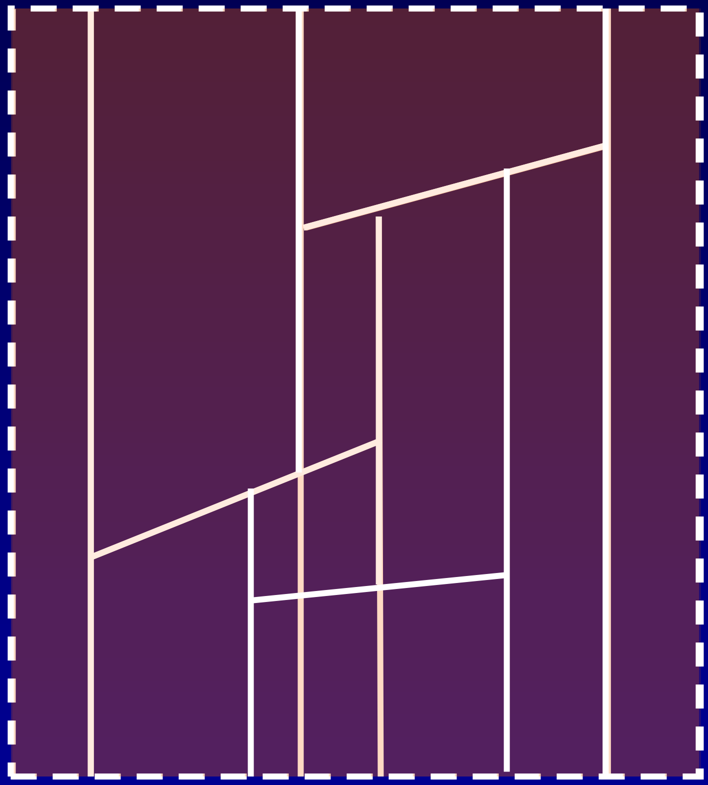


Using the Search Structure



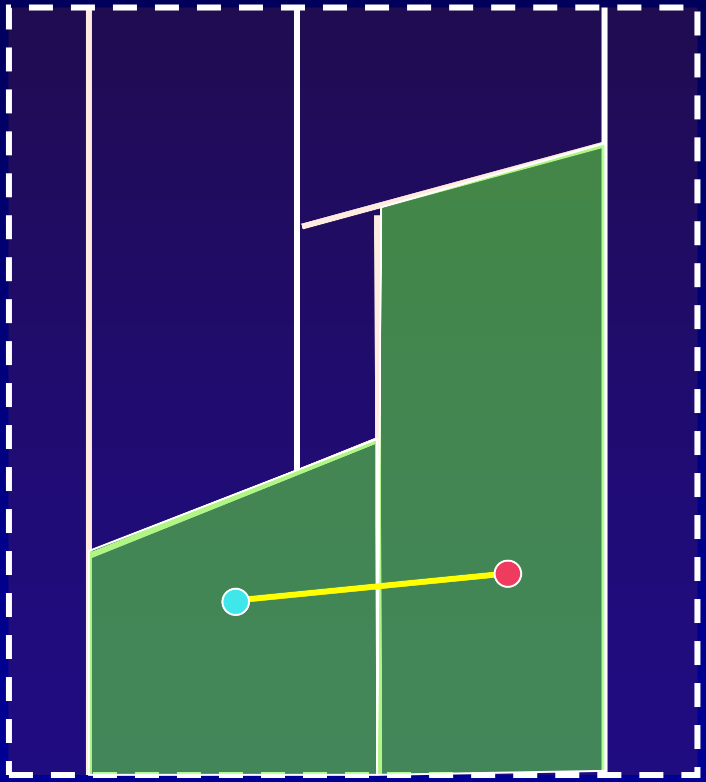
Search Structure: Construction

- Find a Bounding Box.
- Randomly permute the segments.
- Insert the segments one by one into the map.
- Update the map and search structure in each insertion.
- The size of the map is $\Theta(n)$. (This was proven earlier.)
- The size of the search structure depends on the order of insertion (will be analyzed later).



Updating the Structures (High Level)

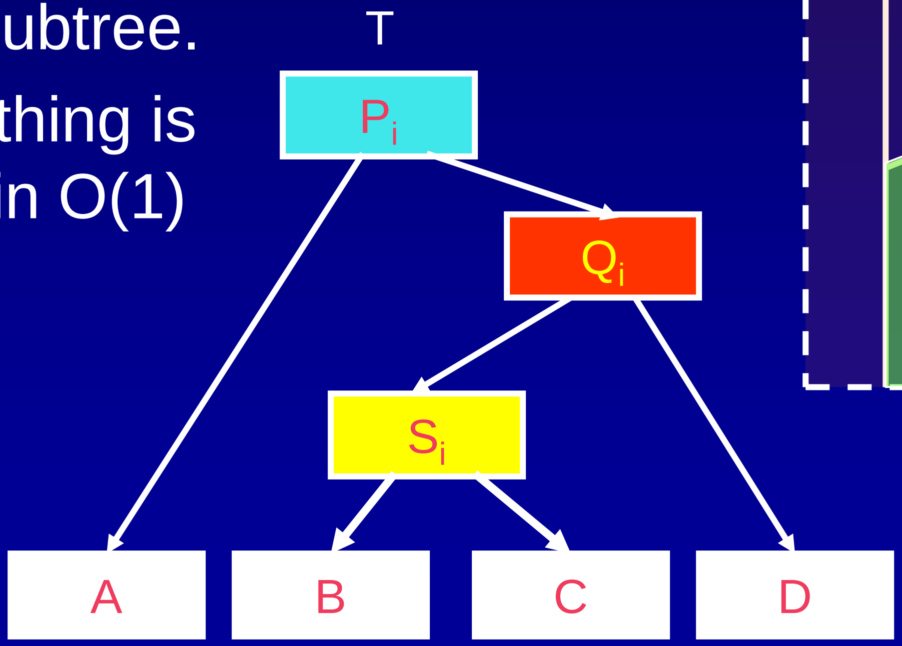
- Find **in the existing structure** the face that contains the left endpoint of the new segment. (*)
- Find all other trapezoids intersected by this segment by moving to the right. (In each move choose between two options: Up or Down.)
- Update the map M_i and the search structure D_i.

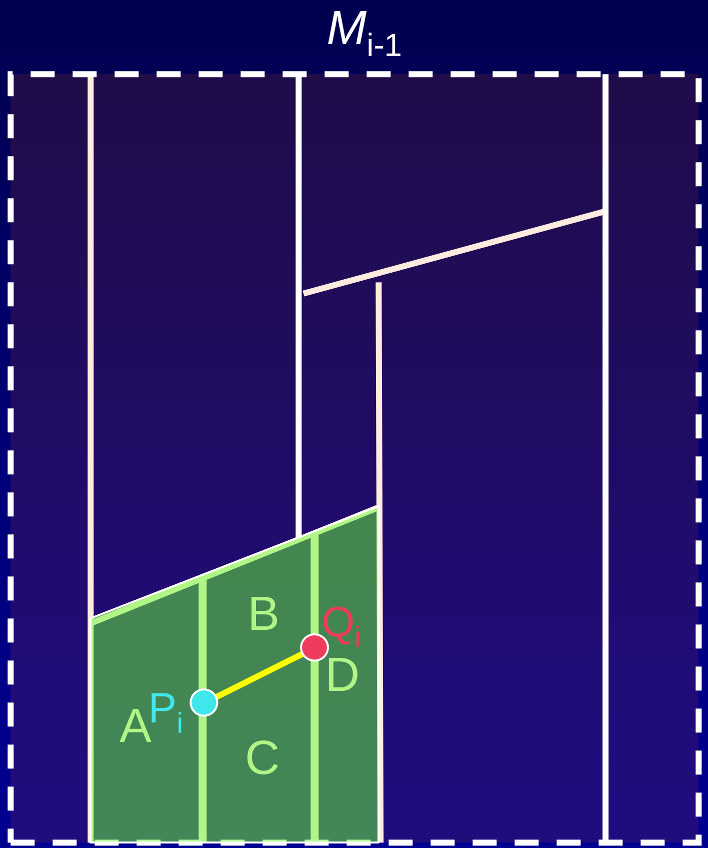


(*) Note: Since endpoints may be shared by segments, we need to consider its segment while searching.

Update: Simple Case

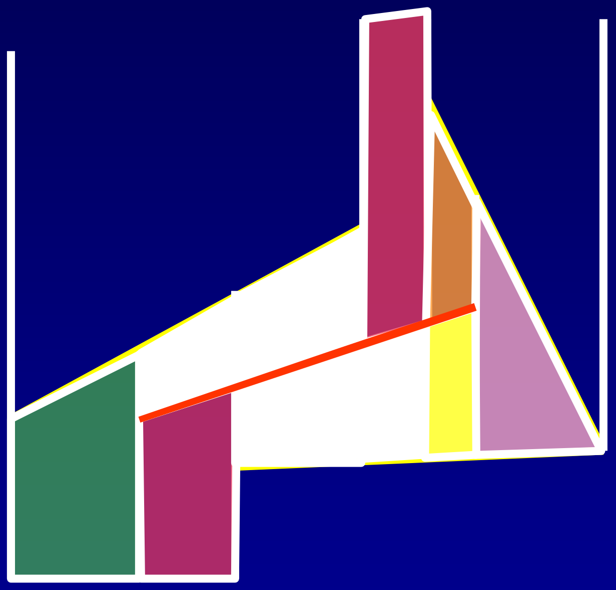
- The segment is contained entirely in one trapezoid.
- In M_{i-1}: Split the trapezoid into four trapezoids.
- In D_{i-1}: The leaf will be replaced by a subtree.
- Everything is done in O(1) time.





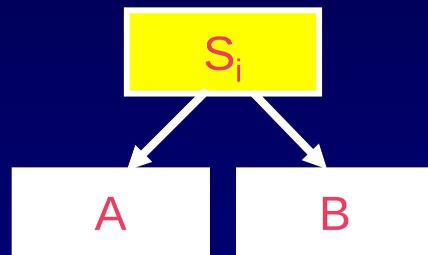
Update *M*: General Case

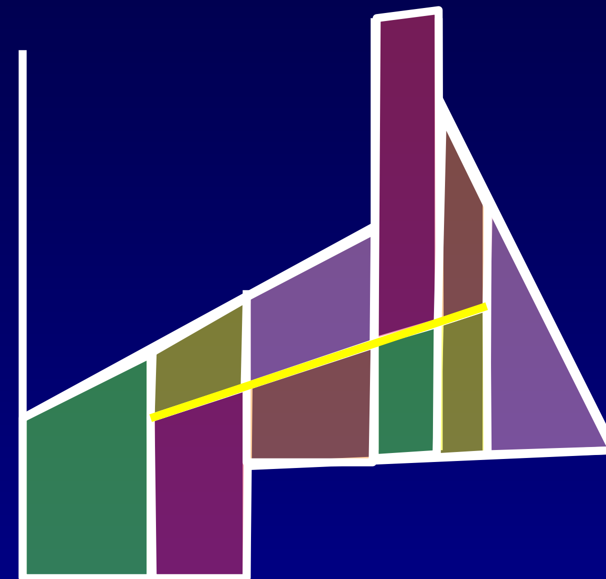
- General Case: The i^{th} segment intersects with $k_i>1$ trapezoids.
- Split trapezoids.
- Merge trapezoids that can be united.
- Total update time: $O(k_i)$.



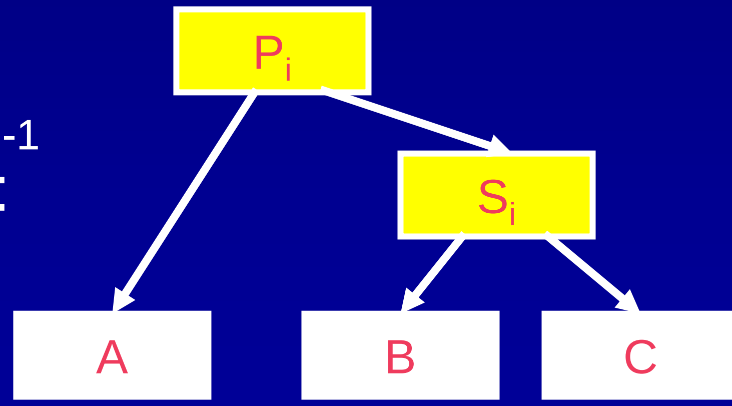
Updating D: Split

- Each *inner* trapezoid in D_{i-1} is replaced by:





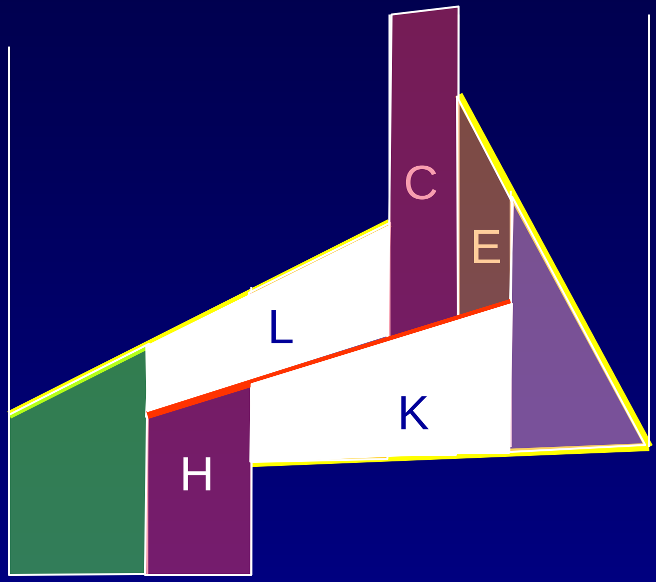
- Each *outer* (e.g., left) trapezoid in D_{i-1} is replaced by:

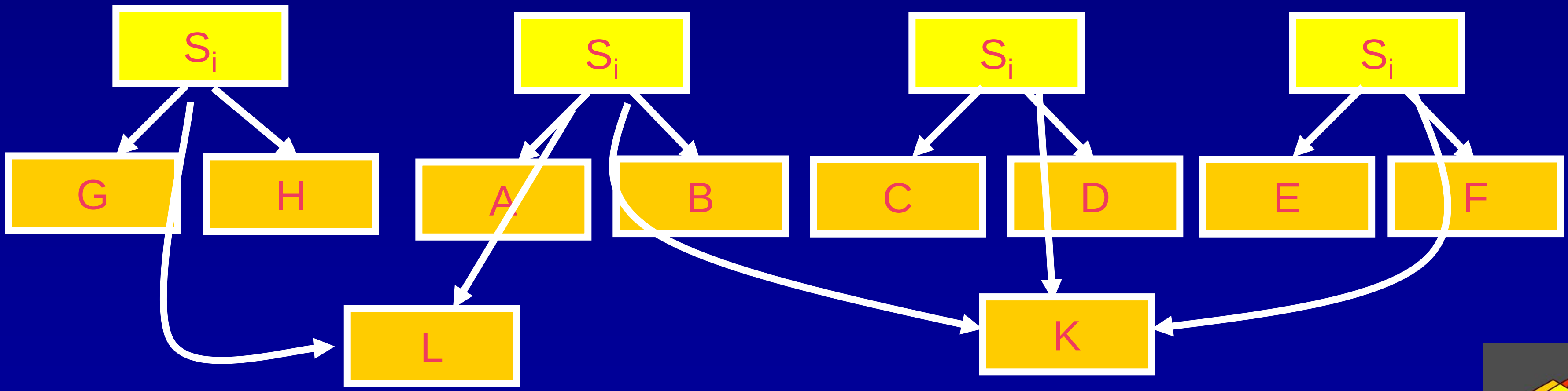


Updating *D*: Merge

- Leaves are eliminated and replaced by one common leaf.

- Total update time: $O(k_i)$.





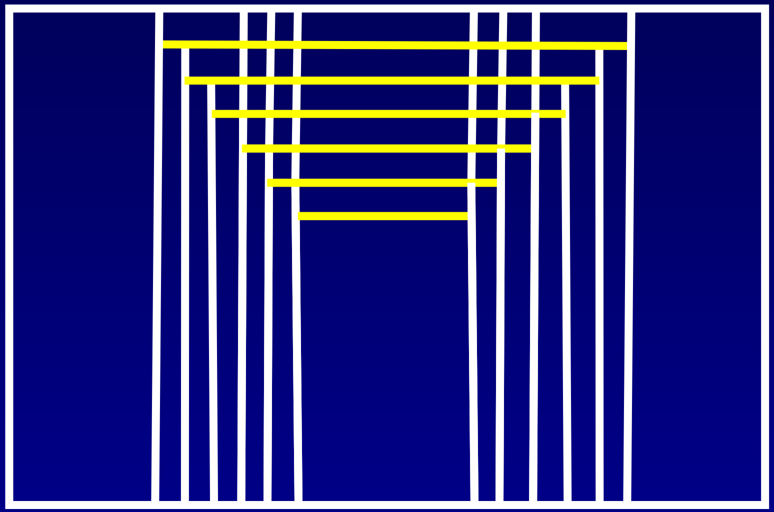
Construction: Worst-Case Analysis

- Each segment adds trees of depth at most (4-1=) 3, so the depth of D_i is at most $3i$.
- Query time (depth of D_i): $O(i)$, $\Theta(i)$ in the worst case.
- The i^{th} segment, s_i, intersects with **O(*i*)** trapezoids (**Θ(*i*)** in the worst case)!
- The size of *D* and its construction time are then bounded from above by

$$\sum_{i=1}^{n} O(i) = O(n^2) .$$

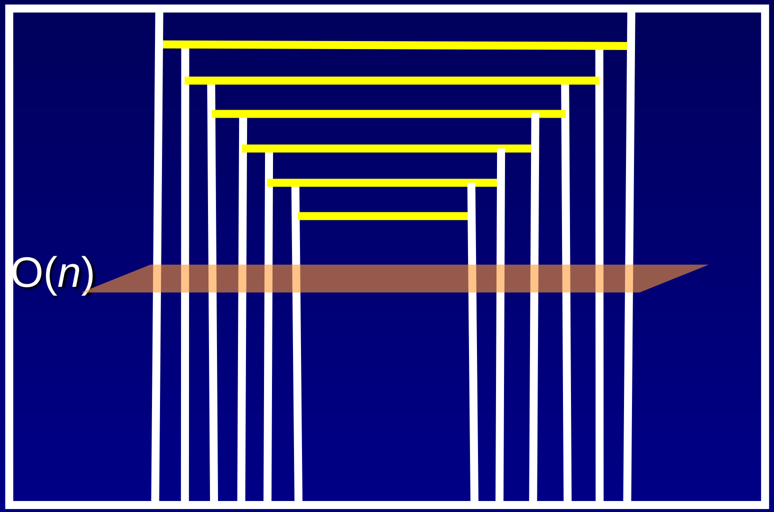
Construction: Worst-Case Analysis (cont.)

Worst-case example:



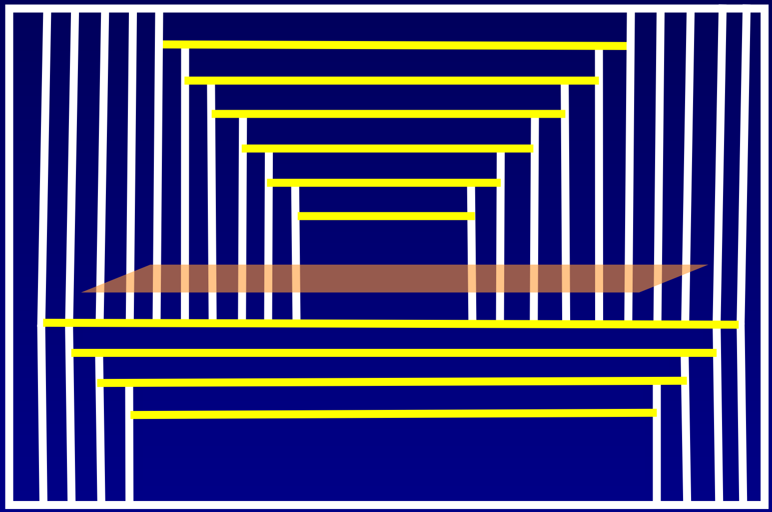
Construction: Worst-Case Analysis (cont.)

Worst-case example:



Construction: Worst-Case Analysis (cont.)

Worst-case example:



The size of D and its construction time is in the worst case.

$$\sum_{i=1}^{\frac{n}{2}} \Theta(1) + \sum_{i=\frac{n}{2}+1}^{n} \Theta(n) = \Theta(n^2)$$

Average-Case Analysis

- We first consider the expected depth of D.
- q: A point, to be searched in D.
- p_i: The probability that a new vertex of D was created in the path leading to q in the i^{th} iteration.

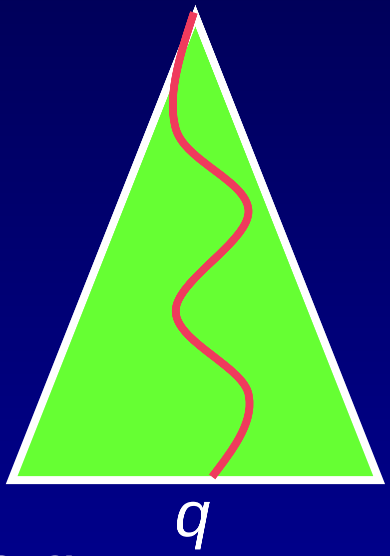
Compute p_i by backward analysis:

- $\Delta_q(M_{i-1})$: The trapezoid containing q in M_{i-1}.
- Since a new vertex of D was created in the i^{th} iteration, $\Delta_q(M_i) \neq \Delta_q(M_{i-1})$.
- Delete s_i from M_i.

p_i = Prob$[\Delta_q(M_i) \neq \Delta_q(M_{i-1})] \leq 4/i$. (Why?)

Expected Depth of *D*

- x_i: The number of vertices **created in the i^{th}** iteration in the path leading to the leaf *q*.



- The expected length of the path leading to *q*:

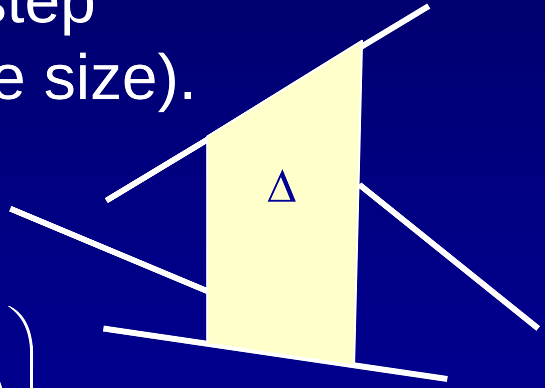
$$E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i] \le \sum_{i=1}^{n} (3p_i) \le \sum_{i=1}^{n} \tfrac{12}{i} = O(\log n).$$

Expected Size of *D*

- Define an indicator

$$\delta_i(\Delta, s) = \begin{cases} 1 & \Delta \text{ disappears from } M_i \text{ if } s \text{ is removed} \\ 0 & \text{otherwise} \end{cases}$$

- k_i: Number of leaves created in the i^{th} step (same order of magnitude(*) as the entire size).
- S_i: The set of the first *i* segments.
- Average on *s*:



$$\mathrm{E}[k_i] = \tfrac{1}{i} \sum_{s \in S_i} \left(\sum_{\Delta \in M_i} \delta_i(\Delta, s) \right) = \tfrac{1}{i} \left(\sum_{s \in S_i} \sum_{\Delta \in M_i} \delta_i(\Delta, s) \right)$$

$$\leq \tfrac{1}{i} (4 \,|\, M_i \,|) \quad \text{(same backward analysis)}$$

$$= \frac{\mathrm{O}(i)}{i} = \mathrm{O}(1).$$

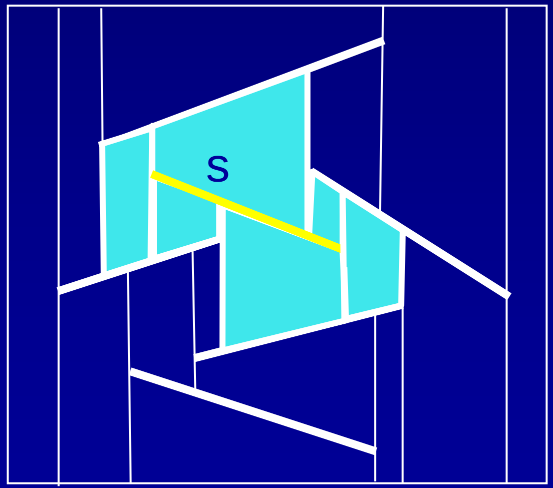
Expected Size of *D* (cont.)

- k_i-1: Number of internal nodes created in the i^{th} step.
- Total size:

$$O(n)+E\left(\sum_{i=1}^{n}(k_i-1)\right)=O(n)+E\left(\sum_{i=1}^{n}k_i\right)=O(n).$$

↑ leaves

↑ internal(*)



Expected Construction Time of *D*

$$\sum_{i=1}^{n}\left(\mathrm{O}(\mathrm{E}(\mathrm{depth}_i)) + \mathrm{O}(\mathrm{E}[k_i])\right)$$

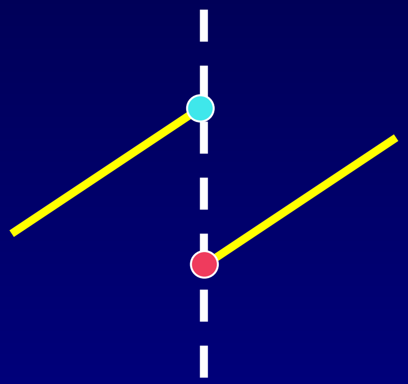
$$= \sum_{i=1}^{n}(\mathrm{O}(\log i) + \mathrm{O}(1)) = \mathrm{O}(n \log n)$$

Finding the first trapezoid

The rest of the work in the i^{th} step

Handling Degeneracies

- What happens if two segment endpoints have the same *x* coordinate?
- Use a shearing transformation: $$\varphi\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + \varepsilon y \\ y \end{pmatrix}$$



- Higher points will move more to the right.
- ε should be small enough so that this transform will not change the order of two points with different *x* coordinates.
- In fact, there is no need to shear the plane. Comparison rules **mimic** the shearing.
- Prove: The entire algorithm remains correct.