Recap: Retroactive Data Structures

**Operations**

- **INSERT**\( (t, \text{operation}) \): insert operation at time \( t \)
- **DELETE**\( (t) \): delete operation at time \( t \)
- **QUERY**\( (t, \text{query}) \): ask \( \text{query} \) at time \( t \)

- for a priority queue updates are
  - insert
  - delete-min

- time is integer 1 for simplicity otherwise use order-maintenance data structure

<table>
<thead>
<tr>
<th>insert(7)</th>
<th>insert(2)</th>
<th>insert(3)</th>
<th>del-min</th>
<th>del-min</th>
<th>queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>now</td>
</tr>
</tbody>
</table>
Recap: Retroactive Data Structures

Operations
- INSERT \( (t, \text{operation}) \): insert operation at time \( t \)
- DELETE \( (t) \): delete operation at time \( t \)
- QUERY \( (t, \text{query}) \): ask \( \text{query} \) at time \( t \)

- for a priority queue updates are
  - insert
  - delete-min

- time is integer \( \dagger \) for simplicity otherwise use order-maintenance data structure

Definition: Partial Retroactivity
QUERY is only allowed for \( t = \infty \ \dagger \) now

Insert(7) Insert(2) Insert(3) Del-min Del-min

0 1 2 3 4 now time

Institute of Theoretical Informatics, Algorithm Engineering
**Recap: Retroactive Data Structures**

**Operations**
- **INSERT**\((t, operation)\): insert operation at time \(t\)
- **DELETE**\((t)\): delete operation at time \(t\)
- **QUERY**\((t, query)\): ask query at time \(t\)

- for a priority queue updates are
  - insert
  - delete-min

- time is integer for simplicity otherwise use order-maintenance data structure

**Definition: Partial Retroactivity**

QUERY is only allowed for \(t = \infty \) now

**Definition: Full Retroactivity**

QUERY is allowed at any time \(t\)

### Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>insert(7) insert(2) insert(3) del-min del-min</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Now</td>
<td></td>
</tr>
</tbody>
</table>
Recap: Retroactive Data Structures

Operations
- **INSERT**\((t, \text{operation})\): insert operation at time \(t\)
- **DELETE**\((t)\): delete operation at time \(t\)
- **QUERY**\((t, \text{query})\): ask query at time \(t\)

for a priority queue updates are
- insert
- delete-min

**time is integer** for simplicity otherwise use order-maintenance data structure

Definition: Partial Retroactivity
**QUERY** is only allowed for \(t = \infty\) \(\oplus\) now

Definition: Full Retroactivity
**QUERY** is allowed at any time \(t\)

Definition: Nonoblivious Retroactivity
**INSERT**, **DELETE**, and **QUERY** at any time \(t\) but also identify changed **QUERY** results

<table>
<thead>
<tr>
<th>query</th>
<th>insert(7)</th>
<th>insert(2)</th>
<th>insert(3)</th>
<th>del-min</th>
<th>del-min</th>
<th>now</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivation: Query Set of Points

- given set of points \( P = \{p_1, \ldots, p_n\} \) with \( p_i = (x_i, y_i) \)
- find all points in \([x, y] \times [x', y']\)
- higher dimensions are possible

- think about database queries
- each dimension is a property
- searching for objects fulfilling all properties of range
Motivation: Query Set of Points

- given set of points $P = \{p_1, \ldots, p_n\}$ with $p_i = (x_i, y_i)$
- find all points in $[x, y] \times [x', y']$
- higher dimensions are possible

- think about database queries
- each dimension is a property
- searching for objects fulfilling all properties of range
Motivation: Query Set of Points

- given set of points $P = \{p_1, \ldots, p_n\}$ with $p_i = (x_i, y_i)$
- find all points in $[x, y] \times [x', y']$
- higher dimensions are possible

- think about database queries
- each dimension is a property
- searching for objects fulfilling all properties of range
1-Dimensional Range Searching (1/2)

- consider 1-dimensional problem
- range is $[x..x']$
- points $P = \{x_1, \ldots, x_n\}$ are just numbers
Consider 1-dimensional problem

Range is \([x .. x']\)

Points \(P = \{x_1, \ldots, x_n\}\) are just numbers

Build BBST where each leaf contains a point

Inner node \(v\) store splitting value \(x_v\)
1-Dimensional Range Searching (1/2)

- consider 1-dimensional problem
- range is \([x..x']\)
- points \(P = \{x_1, \ldots, x_n\}\) are just numbers

- build BBST where each leaf contains a point
- inner node \(v\) store splitting value \(x_v\)
1-Dimensional Range Searching (1/2)

- consider 1-dimensional problem
- range is $[x..x']$
- points $P = \{x_1, \ldots, x_n\}$ are just numbers

- build BBST where each leaf contains a point
- inner node $v$ store splitting value $x_v$

- query for both $x$ and $x'$
- find leaves $b$ and $e$ for $x$ and $x'$
- let node $v$ be node where paths to leaves split
- report all leaves between $b$ and $e$
1-Dimensional Range Searching (2/2)

- how long does it take to report all children of a subtree with $k$ leaves in a BBST?

**Lemma:** 1-Dimensional Range Searching

Let $P$ be a set of $n$ 1-dimensional points. $P$ can be stored in a BBST that requires $O(n)$ words space, can be constructed in $O(n \log n)$ time, and can answer range searching queries in $O(\log n + \text{occ})$ time.

**Proof (Sketch Time):**

- Reporting all children in a subtree requires $O(\text{occ})$ time.
- BBST has depth $O(\log n)$.
- Search paths starting at $v$ have length $O(\log n)$.
- Report all subtrees to the right of the left path.
- Report all subtrees to the left of the right path.
1-Dimensional Range Searching (2/2)

- how long does it take to report all children of a subtree with \( k \) leaves in a BBST?

**Lemma: 1-Dimensional Range Searching**

Let \( P \) be a set of \( n \) 1-dimensional points. \( P \) can be stored in a BBST that requires \( O(n) \) words space, can be constructed in \( O(n \log n) \) time, and can answer range searching queries in \( O(\log n + \text{occ}) \) time.
how long does it take to report all children of a subtree with $k$ leaves in a BBST?

**Lemma: 1-Dimensional Range Searching**

Let $P$ be a set of $n$ 1-dimensional points. $P$ can be stored in a BBST that requires $O(n)$ words space, can be constructed in $O(n \log n)$ time, and can answer range searching queries in $O(\log n + occ)$ time.

**Proof (Sketch Time)**

- reporting all children in a subtree requires $O(occ)$ time
- BBST has depth $O(\log n)$
- search paths starting at $v$ have length $O(\log n)$
- report all subtrees to the right of the left path
- report all subtrees to the left of the right path
Important

- Assume now two points have the same $x$- or $y$-coordinate

- Generalize 1-dimensional idea
  - 1-dimensional
    - Split number of points in half at each node
    - Points consist of one value
  - 2-dimensional
    - Points consist of two values
    - Split number of points in half w.r.t. one value
    - Switch between values depending on depth
Important

- assume now two points have the same x- or y-coordinate

- generalize 1-dimensional idea
  - 1-dimensional
    - split number of points in half at each node
    - points consist of one value
  - 2-dimensional
    - points consist of two values
    - split number of points in half w.r.t. one value
    - switch between values depending on depth
Important

- assume now two points have the same x- or y-coordinate

- generalize 1-dimensional idea
  - 1-dimensional
    - split number of points in half at each node
    - points consist of one value
  - 2-dimensional
    - points consist of two values
    - split number of points in half w.r.t. one value
    - switch between values depending on depth
2-Dimensional Rectangular Range Searching

Important

- assume now two points have the same $x$- or $y$-coordinate

- generalize 1-dimensional idea

1-dimensional
- split number of points in half at each node
- points consist of one value

2-dimensional
- points consist of two values
- split number of points in half w.r.t. one value
- switch between values depending on depth
Important

- assume now two points have the same \(x\)- or \(y\)-coordinate

- generalize 1-dimensional idea
- 1-dimensional
  - split number of points in half at each node
  - points consist of one value
- 2-dimensional
  - points consist of two values
  - split number of points in half w.r.t. one value
  - switch between values depending on depth
2-Dimensional Rectangular Range Searching

Important
- assume now two points have the same $x$- or $y$-coordinate
- generalize 1-dimensional idea
  - 1-dimensional
    - split number of points in half at each node
    - points consist of one value
  - 2-dimensional
    - points consist of two values
    - split number of points in half w.r.t. one value
    - switch between values depending on depth
Important
- assume now two points have the same $x$- or $y$-coordinate
- generalize 1-dimensional idea
  - 1-dimensional
    - split number of points in half at each node
    - points consist of one value
  - 2-dimensional
    - points consist of two values
    - split number of points in half w.r.t. one value
    - switch between values depending on depth
**2-Dimensional Rectangular Range Searching**

**Important**
- Assume now two points have the same $x$- or $y$-coordinate
- Generalize 1-dimensional idea
  - 1-dimensional
    - Split number of points in half at each node
    - Points consist of one value
  - 2-dimensional
    - Points consist of two values
    - Split number of points in half w.r.t. one value
    - Switch between values depending on depth
Important

- assume now two points have the same x- or y-coordinate
- generalize 1-dimensional idea
  - 1-dimensional
    - split number of points in half at each node
    - points consist of one value
  - 2-dimensional
    - points consist of two values
    - split number of points in half w.r.t. one value
    - switch between values depending on depth
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the $x$-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the $y$-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on $x$- and $y$-coordinate
- inner nodes store splitter (line)
considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the x-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the y-coordinate
- until each region contains a single point
- each leaf represents a point

splitting in linear time is complicated
- better presort based on x- and y-coordinate
- inner nodes store splitter (line)
considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the $x$-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the $y$-coordinate
- until each region contains a single point
- each leaf represents a point

splitting in linear time is complicated
- better presort based on $x$- and $y$-coordinate
- inner nodes store splitter (line)
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the $x$-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the $y$-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on $x$- and $y$-coordinate
- inner nodes store splitter (line)
considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the \( x \)-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the \( y \)-coordinate
- until each region contains a single point
- each leaf represents a point

splitting in linear time is complicated
- better presort based on \( x \)- and \( y \)-coordinate
- inner nodes store splitter (line)
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the x-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the y-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on x- and y-coordinate
- inner nodes store splitter (line)
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the $x$-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the $y$-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on $x$- and $y$-coordinate
- inner nodes store splitter (line)
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the $x$-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the $y$-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on $x$- and $y$-coordinate
- inner nodes store splitter (line)
Kd-Trees (1/4)

- considering the 2-dimensional case
- each inner node at an even depth
  - splits the leaves in its subtree in half
  - using the x-coordinate
- each inner node at an odd depth
  - splits the leaves in its subtree in half
  - using the y-coordinate
- until each region contains a single point
- each leaf represents a point

- splitting in linear time is complicated
- better presort based on x- and y-coordinate
- inner nodes store splitter (line)
Lemma: Kd-Tree Construction

A kd-tree for a set of \( n \) points requires \( O(n) \) words space and can be constructed in \( O(n \log n) \) time.

Proof (Sketch: Space)

- There are \( O(n) \) leaves.
- There are \( O(n) \) inner nodes.
- A binary tree requires \( O(1) \) words per node.
- \( O(n) \) words total space.

Proof (Sketch: Time)

- Finding the splitter is easy due to presorted points.
- Splitting requires \( T(n) \) time, with \( T(n) = O(n) + 2T(\lceil n/2 \rceil) \) when \( n > 1 \).
- Results in \( O(n \log n) \) running time.
- Dominates presorting.
Lemma: Kd-Tree Construction

A kd-tree for a set of \( n \) points requires \( O(n) \) words space and can be constructed in \( O(n \log n) \) time.

Proof (Sketch: Space)
- there are \( O(n) \) leaves
- there are \( O(n) \) inner nodes
- a binary tree requires \( O(1) \) words per node
- \( O(n) \) words total space

Proof (Sketch: Time)
Finding the splitter is easy due to presorted points. Splitting requires \( T(n) \) time with
\[
T(n) = \begin{cases} 
O(n) &\text{if } n > 1 \\
2T(\lceil n/2 \rceil) &\text{if } n \leq 1 
\end{cases}
\]
results in \( O(n \log n) \) running time, dominating presorting.
Kd-Trees (2/4)

Lemma: Kd-Tree Construction

A kd-tree for a set of $n$ points requires $O(n)$ words space and can be constructed in $O(n \log n)$ time.

Proof (Sketch: Space)
- there are $O(n)$ leaves
- there are $O(n)$ inner nodes
- a binary tree requires $O(1)$ words per node
- $O(n)$ words total space

Proof (Sketch: Time)
- finding the splitter is easy due to presorted points
- splitting requires $T(n)$ time with
  $$T(n) = \begin{cases} O(1) & n = 1 \\ O(n) + 2T(\lceil n/2 \rceil) & n > 1 \end{cases}$$
- results in $O(n \log n)$ running time
- dominates presorting
Kd-Trees (3/4)

- use splitter depending on depth to identify paths through tree
- if a region is fully contained in query: report region
- if a region is intersected by query: check if point has to be reported
Kd-Trees (3/4)

- Use splitter depending on depth to identify paths through tree
- If a region is fully contained in query: report region
- If a region is intersected by query: check if point has to be reported

- Precomputation of requires note necessary
- Current region can be computed during query
- Using splitters
Kd-Trees (3/4)

- Use splitter depending on depth to identify paths through tree.
- If a region is fully contained in query: report region.
- If a region is intersected by query: check if point has to be reported.

- Precomputation of requires note necessary.
- Current region can be computed during query.
- Using splitters.

Example on the board.
Lemma: Kd-Tree Query

A query with an axis-parallel rectangle in a Kd-tree storing \( n \) points in the plane can be performed in \( O(\sqrt{n} + \text{occ}) \) time.

Proof (Sketch)

- \( O(\text{occ}) \) time necessary to report points
- Look at number of regions intersected by any vertical line
- Upper bound for the regions intersected by query (for left and right edge of rectangle)
- Upper bound for top and bottom edges are the same

Proof (Sketch, cont.)

- For vertical lines consider every inner node at odd depth starting at root's children
- Can intersect two regions
- Number of nodes is \( \lceil n/4 \rceil \)
- Number of intersected regions is \( Q(n) \) with
  \[
  Q(n) = \begin{cases} \quad O(1) & n = 1 \\ 2 \times Q(\lfloor n/4 \rfloor) & n > 1 \\ \end{cases}
  \]

- Results in \( Q(n) = O(\sqrt{n}) \)
- Total running time is \( O(\sqrt{n} + \text{occ}) \)
Kd-Trees (4/4)

Lemma: Kd-Tree Query

A query with an axis-parallel rectangle in a Kd-tree storing \( n \) points in the plane can be performed in \( O(\sqrt{n} + \text{occ}) \) time.

Proof (Sketch)

- \( O(\text{occ}) \) time necessary to report points
- Look at number of regions intersected by any vertical line
- Upper bound for the regions intersected by query (for left and right edge of rectangle)
- Upper bound for top and bottom edges are the same

\[ O(\text{occ}) \leq O(\sqrt{n}) \]
Lemma: Kd-Tree Query

A query with an axis-parallel rectangle in a Kd-tree storing \( n \) points in the plane can be performed in \( O(\sqrt{n} + \text{occ}) \) time.

Proof (Sketch)

- \( O(\text{occ}) \) time necessary to report points
- Look at number of regions intersected by any vertical line
- Upper bound for the regions intersected by query (for left and right edge of rectangle)
- Upper bound for top and bottom edges are the same

Proof (Sketch, cnt.)

- For vertical lines consider every inner node at odd depth
- Starting at root's children
- Can intersect two regions
- Number of nodes is \( \lceil n/4 \rceil \) halved at each level
- Number of intersected regions is \( Q(n) \) with

\[
Q(n) = \begin{cases} 
O(1) & n = 1 \\
2 + 2Q(\lceil n/4 \rceil) & n > 1 
\end{cases}
\]

- Results in \( Q(n) = O(\sqrt{n}) \)
- \( O(\sqrt{n} + k) \) total running time
Range Trees (1/4)

- one BBST build on the x-coordinates
  - same as for 1-dimensional queries
- each inner node is associated with a set of points
- build a BBST for the y-coordinates of associated points for each inner node
  - store points in leaves not just y-coordinates
  - this BBST is used for reporting

- space-query-time trade-off
- faster queries but larger
- one BBST build on the $x$-coordinates
  - same as for 1-dimensional queries
- each inner node is associated with a set of points
- build a BBST for the $y$-coordinates of associated points for each inner node
  - store points in leaves not just $y$-coordinates
  - this BBST is used for reporting
- space-query-time trade-off
- faster queries but larger
Range Trees (1/4)

- one BBST build on the $x$-coordinates
  - same as for 1-dimensional queries
- each inner node is associated with a set of points
- build a BBST for the $y$-coordinates of associated points for each inner node
  - store points in leaves not just $y$-coordinates
  - this BBST is used for reporting

- space-query-time trade-off
- faster queries but larger
the BBST for the x-coordinates requires $O(n)$ words of space
how much space do the associated BBSTs require in total? PINGO
Range Trees (2/4)

- the BBST for the $x$-coordinates requires $O(n)$ words of space
- how much space do the associated BBSTs require in total?

Lemma: Space Range Tree

A range tree on a set of $n$ points in the plane requires $O(n \log n)$ words space
Range Trees (2/4)

- the BBST for the \( x \)-coordinates requires \( O(n) \) words of space
- how much space do the associated BBSTs require in total?

Lemma: Space Range Tree

A range tree on a set of \( n \) points in the plane requires \( O(n \log n) \) words space

Proof (Sketch)

- BBST for \( x \)-coordinates has depth \( O(\log n) \)
- all points are represented on each depth exactly once
Range Trees (2/4)

- the BBST for the $x$-coordinates requires $O(n)$ words of space
- how much space do the associated BBSTs require in total?

Lemma: Space Range Tree
A range tree on a set of $n$ points in the plane requires $O(n \log n)$ words space

Proof (Sketch)
- BBST for $x$-coordinates has depth $O(\log n)$
- all points are represented on each depth exactly once

Proof (Sketch, cnt.)
- all associated BBSTs on each depth contain every point exactly once
- total size of all BBSTs on each depth is $O(n)$
- total space $O(n \log n)$ words
Range Trees (2/4)

- the BBST for the $x$-coordinates requires $O(n)$ words of space
- how much space do the associated BBSTs require in total?

Lemma: Space Range Tree
A range tree on a set of $n$ points in the plane requires $O(n \log n)$ words space

Proof (Sketch)
- BBST for $x$-coordinates has depth $O(\log n)$
- all points are represented on each depth exactly once

Proof (Sketch, cnt.)
- all associated BBSTs on each depth contain every point exactly once
- total size of all BBSTs on each depth is $O(n)$
- total space $O(n \log n)$ words

how much faster is the range tree?
2-dimensional rectangular range search reduced to two 1-dimensional range searches

- look in BBST for x-coordinates same as 1-dimensional case
- instead of reporting subtrees to the right/left of paths search associated BBSTs
- report results in leaves of associated BBSTs
2-dimensional rectangular range search reduced to two 1-dimensional range searches

look in BBST for $x$-coordinates same as 1-dimensional case

instead of reporting subtrees to the right/left of paths search associated BBSTs

report results in leaves of associated BBSTs

Lemma: Rang Tree Query Time

A query with an axis-parallel rectangle in a range tree storing $n$ points requires $O(\log^2 n + \text{occ})$ time
2-dimensional rectangular range search reduced to two 1-dimensional range searches

look in BBST for $x$-coordinates same as 1-dimensional case

instead of reporting subtrees to the right/left of paths search associated BBSTs

report results in leaves of associated BBSTs

Lemma: Rang Tree Query Time

A query with an axis-parallel rectangle in a range tree storing $n$ points requires $O(\log^2 n + \text{occ})$ time

Proof (Sketch)

- each search in an associated BBST $t$ requires $O(\log n + \text{occ}_t)$ time
- $O(\log n)$ associated BSSTs $T$ are searched as seen in 1-dimensional case
- total query time $\sum_{t \in T} O(\log n + \text{occ}_t)$
- $\sum_{t \in T} O(\text{occ}_t) = O(\text{occ})$
- $\sum_{t \in T} O(\log n) = O(\log^2 n)$
- total time: $O(\log^2 n + \text{occ})$
range trees can be generalized to higher dimensions
for each dimension add an additional associated BBST
reporting in final BBST
d-dimensional queries are d 1-dimensional queries
range trees can be generalized to higher dimensions
- for each dimension add an additional associated BBST
- reporting in final BBST
- \(d\)-dimensional queries are \(d\) 1-dimensional queries

**Lemma: Higher Dimensions Range Tree**

A \(d\)-dimensional range tree (for \(d \geq 2\)) storing \(n\) points in the plane requires \(O(n \log^{d-1} n)\) words space and can answer queries in \(O(\log^d n + \text{occ})\) time
Range Trees (4/4)

- range trees can be generalized to higher dimensions
- for each dimension add an additional associated BBST
- reporting in final BBST
- $d$-dimensional queries are $d$ 1-dimensional queries

Proof (Sketch Query Time)

- recursive query time $Q_d(n)$ with $Q_2(n) = O(\log^2 n)$
- $Q_d(n) = O(\log n) + O(\log n) \cdot Q_{d-1}(n)$
- solves to $Q_d(n) = O(\log^d n)$
- $O(\text{occ})$ time for reporting

Lemma: Higher Dimensions Range Tree

A $d$-dimensional range tree (for $d \geq 2$) storing $n$ points in the plane requires $O(n \log^{d-1} n)$ words space and can answer queries in $O(\log^d n + \text{occ})$ time
Range Trees (4/4)

- range trees can be generalized to higher dimensions
- for each dimension add an additional associated BBST
- reporting in final BBST
- $d$-dimensional queries are $d$ 1-dimensional queries

Lemma: Higher Dimensions Range Tree

A $d$-dimensional range tree (for $d \geq 2$) storing $n$ points in the plane requires $O(n \log^{d-1} n)$ words space and can answer queries in $O(\log^d n + \text{occ})$ time

Proof (Sketch Query Time)

- recursive query time $Q_d(n)$ with $Q_2(n) = O(\log^2 n)$
- $Q_d(n) = O(\log n) + O(\log n) \cdot Q_{d-1}(n)$
- solves to $Q_d(n) = O(\log^d n)$
- $O(\text{occ})$ time for reporting

Proof (Sketch Construction Space)

- recursive space $S_d(n)$ with $S_2(n) = O(n \log n)$ words
- $T_d(n) = O(n \log n) + O(\log n) \cdot T_{d-1}(n)$
- solves to $S_d(n) = O(n \log^{d-1} n)$
Fractional Cascading (1/2)

- sorted sets $S_1, \ldots, S_m$
- $|S_1| = n$ and $S_{i+1} \subseteq S_i$
- report elements in range $[x..x']$ in $S_1, \ldots, S_m$
Fractional Cascading (1/2)

- sorted sets $S_1, \ldots, S_m$
- $|S_1| = n$ and $S_{i+1} \subseteq S_i$
- report elements in range $[x..x']$ in $S_1, \ldots, S_m$

how much time does a naive algorithm with binary search require?
**Fractional Cascading (1/2)**

- sorted sets $S_1, \ldots, S_m$
- $|S_1| = n$ and $S_{i+1} \subseteq S_i$
- report elements in range $[x..x']$ in $S_1, \ldots, S_m$

- how much time does a naive algorithm with binary search require?

  - $O(m \log n + \text{occ})$ time
Fractional Cascading (1/2)

- sorted sets $S_1, \ldots, S_m$
- $|S_1| = n$ and $S_{i+1} \subseteq S_i$
- report elements in range $[x..x']$ in $S_1, \ldots, S_m$

- how much time does a naive algorithm with binary search require?
  - $O(m \log n + \text{occ})$ time
  - $O(m + \log n + \text{occ})$ time possible with fractional cascading
Fractional Cascading (1/2)

- sorted sets $S_1, \ldots, S_m$
- $|S_1| = n$ and $S_{i+1} \subseteq S_i$
- report elements in range $[x..x']$ in $S_1, \ldots, S_m$

- in addition to $S_i$ store pointers to $S_{i+1}$
- for each element in $S_i$ store pointer to successor in $S_{i+1}$
- possible because $S_{i+1} \subseteq S_i$

- how much time does a naive algorithm with binary search require?
  - $O(m \log n + \text{occ})$ time
  - $O(m + \log n + \text{occ})$ time possible with fractional cascading
Lemma: Fractional Cascading

Given sets $S_1, \ldots, S_m$ with $|S_1| = n$ and $S_{i+1} \subseteq S_i$, find a range in all $S_i$'s using fractional cascading requires $O(m + \log n + \text{occ})$ time.
Lemma: Fractional Cascading

Given sets $S_1, \ldots, S_m$ with $|S_1| = n$ and $S_{i+1} \subseteq S_i$, find a range in all $S_i$'s using fractional cascading requires $O(m + \log n + occ)$ time

Proof (Sketch)

- binary search on $S_1$ requires $O(\log n)$ time
- following pointer to $S_2$ requires $O(1)$ time
- scanning $S_2$ requires $O(occ)$ time
- following pointer to $S_3$ requires $O(1)$ time
- repeat $m$ times
- total: $O(m + \log n + occ)$ time
Lemma: Fractional Cascading

Given sets $S_1,\ldots, S_m$ with $|S_1| = n$ and $S_{i+1} \subseteq S_i$, find a range in all $S_i$'s using fractional cascading requires $O(m + \log n + \text{occ})$ time.

Proof (Sketch)

- binary search on $S_1$ requires $O(\log n)$ time
- following pointer to $S_2$ requires $O(1)$ time
- scanning $S_2$ requires $O(\text{occ})$ time
- following pointer to $S_3$ requires $O(1)$ time
- repeat $m$ times
- total: $O(m + \log n + \text{occ})$ time

- how to apply to range trees?
- instead of associated BBSTs store leaf data in arrays for all nodes but root
- each node has associated data
- store two successor pointers to the associated data in the left and right child
- two pointers to cover all possible paths
- this is a layered range tree
Query Layered Range Trees

- search in BBST for x-coordinates remains the same
- to search y-coordinates first search associated BBST of root
- same as initial binary search for fractional cascading
- continue to follow pointers in associated data and scan to report queries

Lemma: Query time Layered Range Tree

A query with an axis-parallel rectangle in a layered range tree storing \( n \) points in the plane can be performed in \( O(\log n + \text{occ}) \) time.

Proof (Sketch)

The initial search requires \( O(\log n) \) time.

The search in the associated BBST of the root requires \( O(\log n) \) time.

Remaining searches in associated data require \( O(1 + \text{occ}) \) time.

Each point is reported once.

Total time: \( O(\log n + \text{occ}) \)
Query Layered Range Trees

- search in BBST for x-coordinates remains the same
- to search y-coordinates first search associated BBST of root
- same as initial binary search for fractional cascading
- continue to follow pointers in associated data and scan to report queries

**Lemma: Query time Layered Range Tree**

A query with an axis-parallel rectangle in a layered range tree storing $n$ points in the plane can be performed in $O(\log n + occ)$ time
Query Layered Range Trees

- search in BBST for \( x \)-coordinates remains the same
- to search \( y \)-coordinates first search associated BBST of root
- same as initial binary search for fractional cascading
- continue to follow pointers in associated data and scan to report queries

Proof (Sketch)

- the initial search requires \( O(\log n) \) time
- the search in the associated BBST of the root requires \( O(\log n) \) time
- remaining searches in associated data \( a \) requires \( O(1 + occ_a) \) time
- each point is reported once
- total time: \( O(\log n + occ) \)

Lemma: Query time Layered Range Tree

A query with an axis-parallel rectangle in a layered range tree storing \( n \) points in the plane can be performed in \( O(\log n + occ) \) time
General Sets of Points

- All solutions require unique $x$ and $y$-coordinates.
- Big limitation for applications.
- Remember database motivation.

Institute of Theoretical Informatics, Algorithm Engineering
all solutions require unique \( x \) and \( y \)-coordinates

big limitation for applications

remember database motivation

store \((x|k)\) as coordinate with \( x \) being the \( x \)-coordinate and \( k \) a unique key

same for \( y \)-coordinates

compare points using

\[(x|k) < (x'|k') \iff x < x' \text{ or } (x = x' \text{ and } k < k')\]
General Sets of Points

- all solutions requires unique $x$ and $y$-coordinates
- big limitation for applications
- remember database motivation

- store $(x|k)$ as coordinate with $x$ being the $x$-coordinate and $k$ a unique key
- same for $y$-coordinates
- compare points using $(x|k) < (x'|k') \iff x < x'$ or $(x = x'$ and $k < K')$

- range queries $[x..x'] \times [y..y']$ become
  
  $$[(x| - \infty)..(x'|\infty)] \times (y| - \infty)..([y'|\infty])$$
Conclusion and Outlook

This Lecture
- orthogonal range searching
Conclusion and Outlook

This Lecture
- orthogonal range searching

Next Lecture
- geometric data structures
- Q&A
- results of evaluation