# Geometric data structures

Sudebkumar Prasant Pal
Department of Computer Science and Engineering
IIT Kharagpur, 721302.
email: spp@cse.iitkgp.ernet.in

April 4, 2010

# Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.

# Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.

# Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.

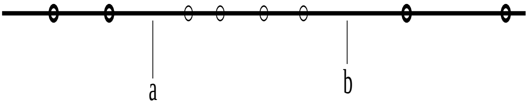
# Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.
- **Segment trees**
  For reporting (portions of) all segments inside a query window.

# Scope of the lecture

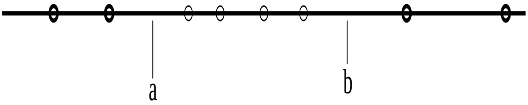
- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.
- **Segment trees**
  For reporting (portions of) all segments inside a query window.
- **Planar point location**
  Using triangulation refinement.

# 1-dimensional Range searching



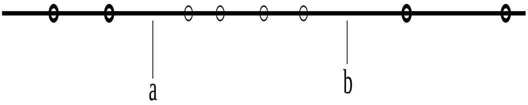

- Problem: Given a set $P$ of $n$ points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of $P$ that lie in the range $[a, b]$, $a \leq b$.

# 1-dimensional Range searching



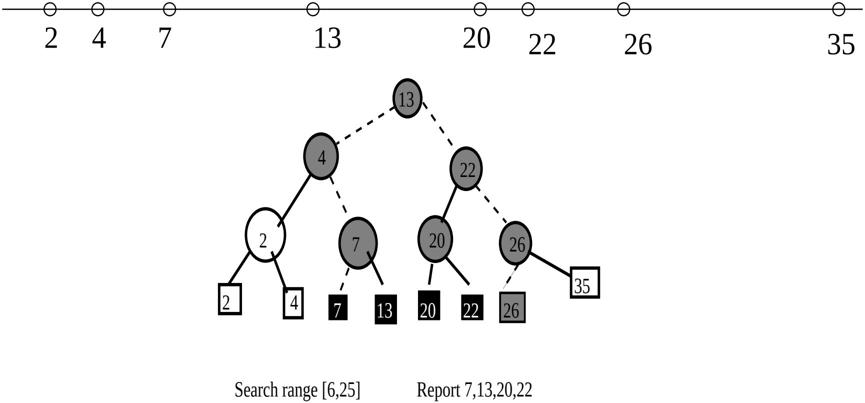

- Problem: Given a set $P$ of $n$ points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of $P$ that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where $k$ is the number of points of $P$ in $[a, b]$.

# 1-dimensional Range searching



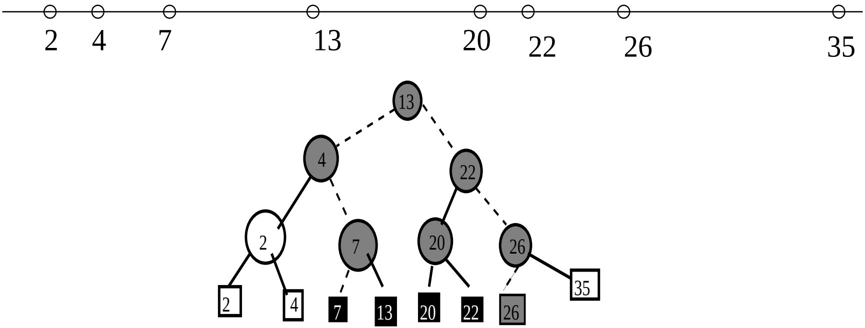

- Problem: Given a set $P$ of $n$ points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of $P$ that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where $k$ is the number of points of $P$ in $[a, b]$.
- However, when we permit insertion or deletion of points, we cannot use an array answering queries so efficiently.

# 1-dimensional Range searching




- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.

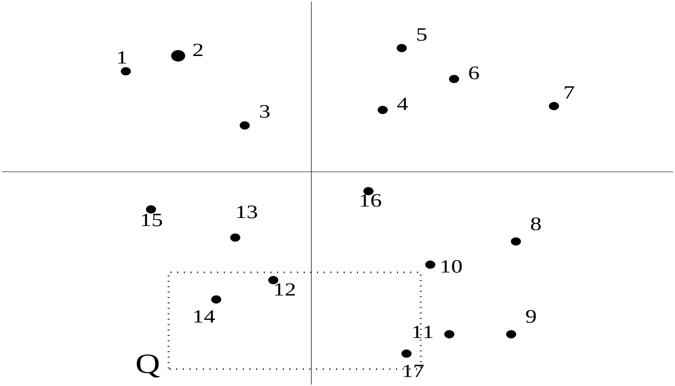
# 1-DIMENSIONAL RANGE SEARCHING




Search range [6,25] Report 7,13,20,22

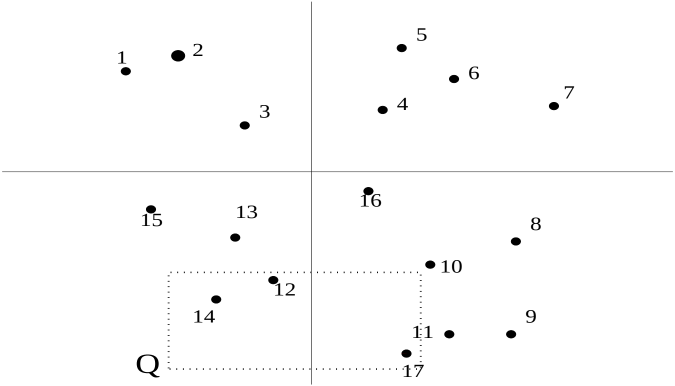
- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.
- Each internal node stores the x-coordinate of the rightmost point in its left subtree for guiding search.

# 2-dimensional Range Searching



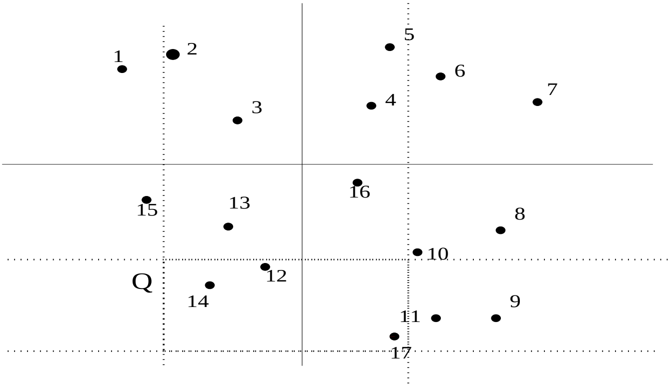

- Problem: Given a set $P$ of $n$ points in the plane, report points inside a query rectangle $Q$ whose sides are parallel to the axes.

# 2-dimensional Range Searching



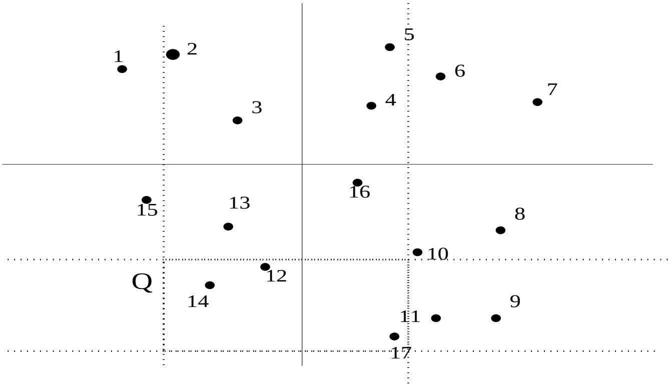

- Problem: Given a set $P$ of $n$ points in the plane, report points inside a query rectangle $Q$ whose sides are parallel to the axes.
- Here, the points inside $R$ are 14, 12 and 17.

# 2-dimensional Range Searching




- Using two 1-d range queries, one along each axis, solves the 2-d range query.

# 2-dimensional Range Searching




- Using two 1-d range queries, one along each axis, solves the 2-d range query.
- The cost incurred may exceed the actual output size of the 2-d range query.

# Range searching with range trees and Kd-trees

- Given a set $S$ of $n$ points in the plane, we can construct a *2d-range tree* in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.

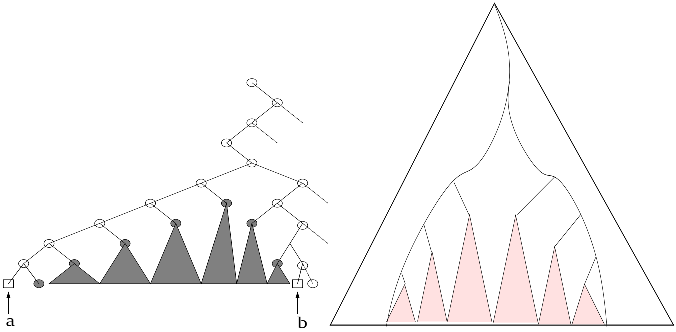
# Range searching with range trees and Kd-trees

- Given a set $S$ of $n$ points in the plane, we can construct a *2d-range tree* in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.
- The query time can be improved to $O(\log n + k)$ using the technique of *fractional cascading*.

# Range searching with range trees and Kd-trees

- Given a set $S$ of $n$ points in the plane, we can construct a *2d-range tree* in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.
- The query time can be improved to $O(\log n + k)$ using the technique of *fractional cascading*.
- Given a set $S$ of $n$ points in the plane, we can construct a Kd-tree in $O(n \log n)$ time and $O(n)$ space, so that *rectangle queries* can be executed in $O(\sqrt{n} + k)$ time. Here, the number of points in the query rectangle is $k$.

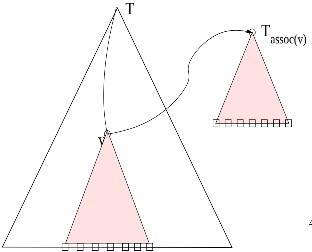
# Range searching in the plane using range trees

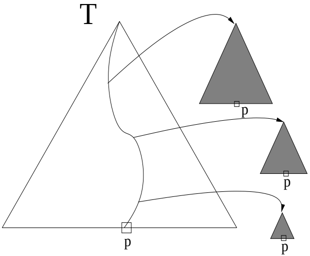



Given a 2-d rectangle query $[a, b]X[c, d]$, we can identify subtrees whose leaf nodes are in the range $[a, b]$ along the X-direction.

Only a subset of these leaf nodes lie in the range $[c, d]$ along the Y-direction.

# Range searching in the plane using range trees



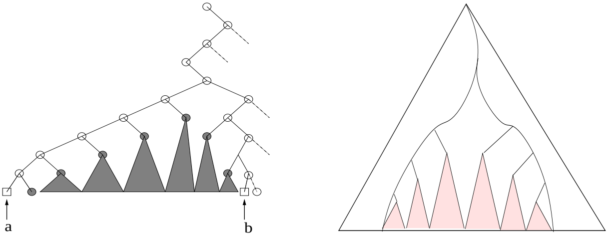




$T_{assoc(v)}$ is a binary search tree on y-coordinates for points in the leaf nodes of the subtree tooted at $v$ in the tree $T$.

The point $p$ is duplicated in $T_{assoc(v)}$ for each $v$ on the search path for $p$ in tree $T$.

The total space requirement is therefore $O(n \log n)$.

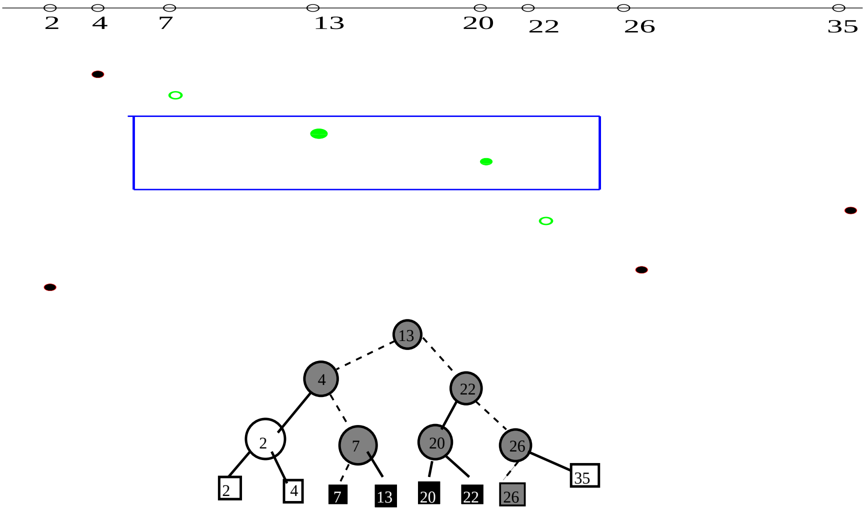
# Range searching in the plane using range trees




We perform 1-d range queries with the y-range $[c, d]$ in each of the subtrees adjacent to the left and right search paths within the x-range $[a, b]$ in the tree $T$.

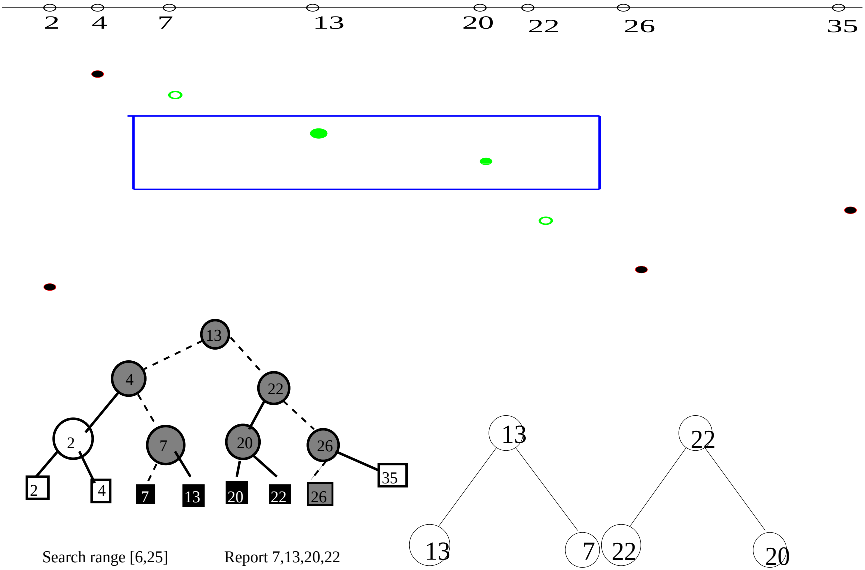
Since the search path is $O(\log n)$ in size, and each y-range query requires $O(\log n)$ time, the total cost of searching is $O(\log^2 n)$. The reporting cost is $O(k)$ where $k$ points lie in the query rectangle.

# 2-RANGE TREE SEARCHING

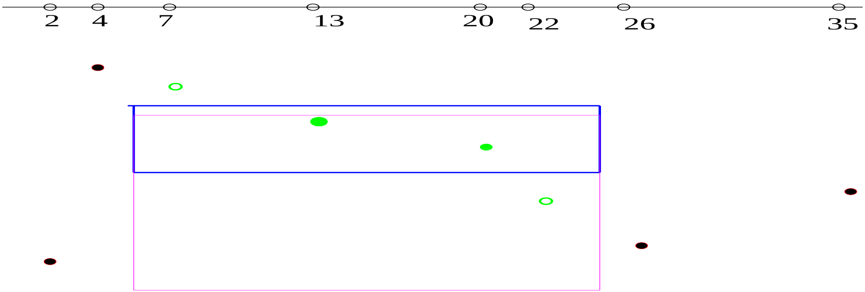


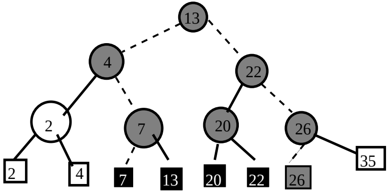

Search range [6,25] Report 7,13,20,22

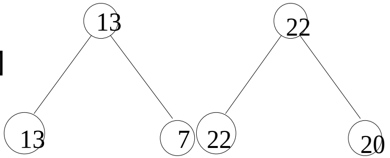
# 2-RANGE TREE SEARCHING



# 2-RANGE TREE SEARCHING



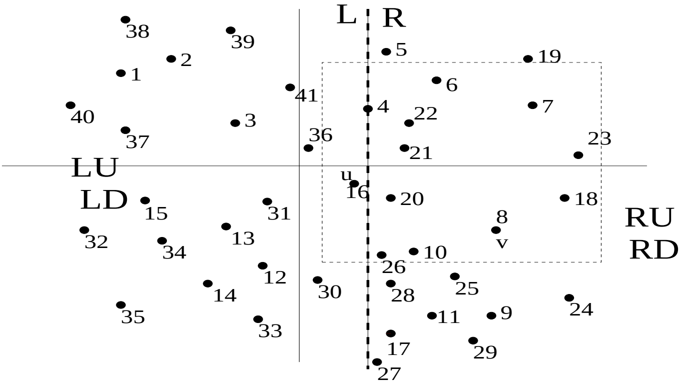



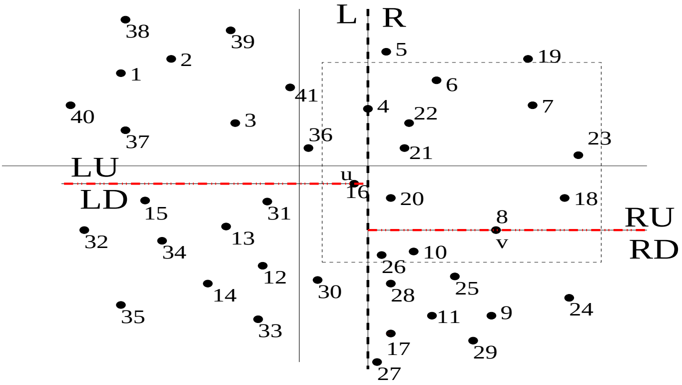

Search range [6,25]

Report 7,13,20,22

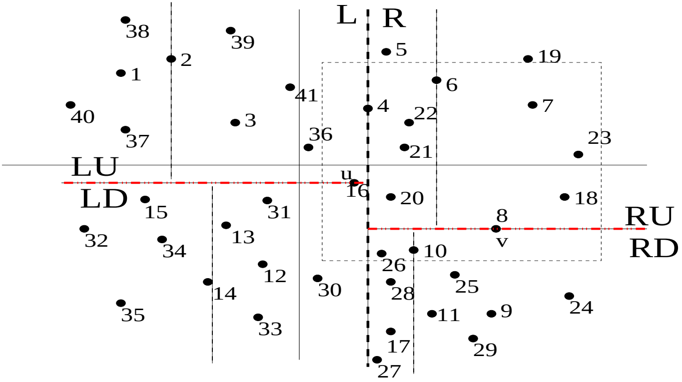
# Partition by the median of x-coordinates



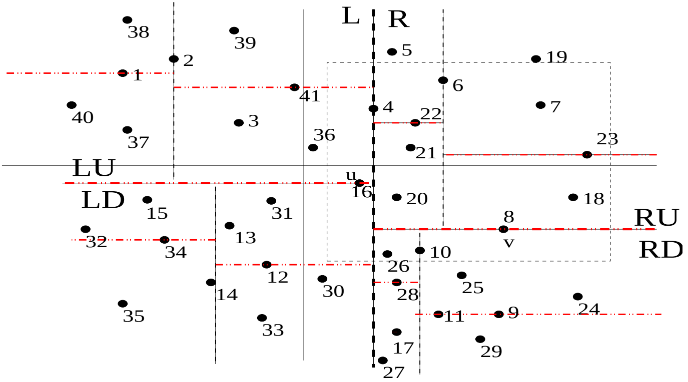
# Partition by the median of y-coordinates



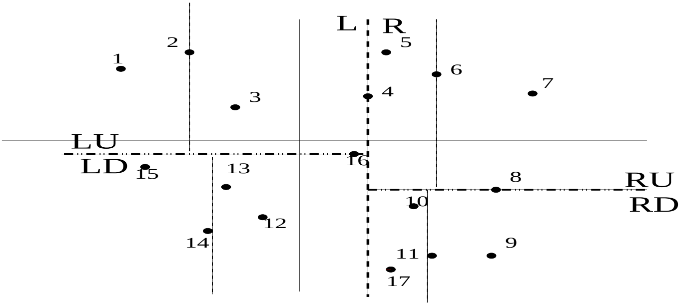
# Partition by the median of x-coordinates

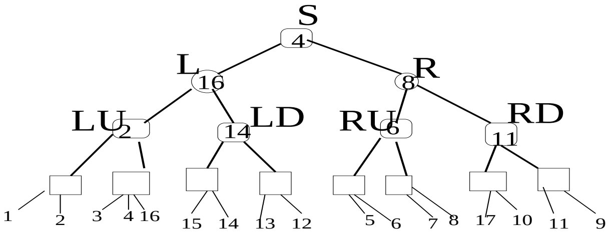


# Partition by the median of y-coordinates

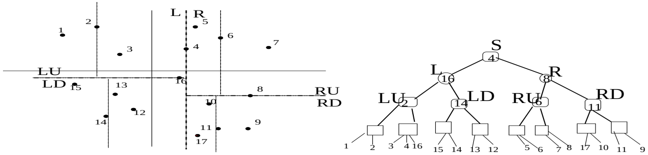


# 2-dimensional range searching using Kd-trees



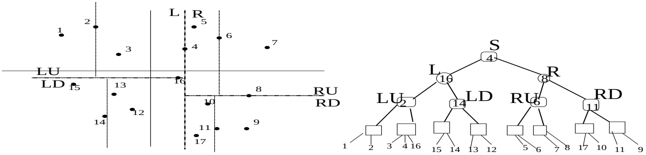



# Description of the Kd-tree



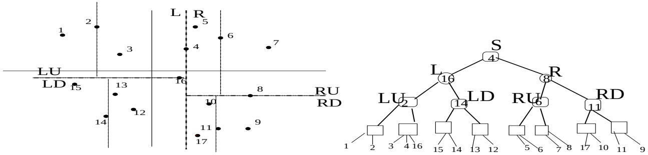

- The tree $T$ is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in $P$ using x- and y- coordinates, respectively as follows.

# Description of the Kd-tree



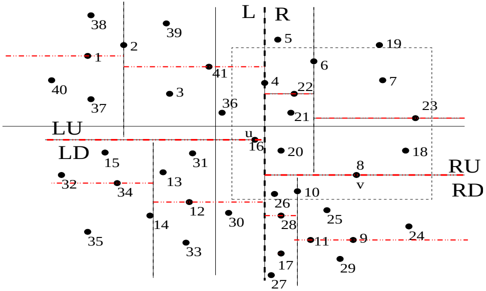

- The tree $T$ is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in $P$ using x- and y- coordinates, respectively as follows.
- The point $r$ stored in the root vertex $T$ splits the set $S$ into two roughly equal sized sets $L$ and $R$ using the median $x$-cooordinate $xmedian(S)$ of points in $S$, so that all points in $L$ ($R$) have abscissae less than or equal to (strictly greater than) $xmedian(S)$.

# Description of the Kd-tree



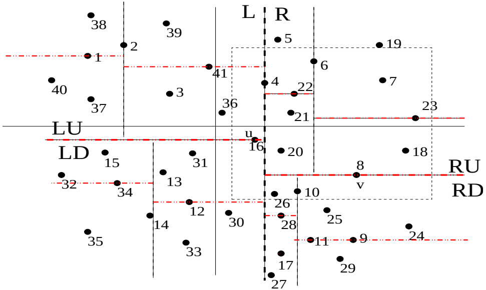

- The tree $T$ is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in $P$ using x- and y- coordinates, respectively as follows.
- The point $r$ stored in the root vertex $T$ splits the set $S$ into two roughly equal sized sets $L$ and $R$ using the median $x$-cooordinate $xmedian(S)$ of points in $S$, so that all points in $L$ $(R)$ have abscissae less than or equal to (strictly greater than) $xmedian(S)$.
- The entire plane is called the $region(r)$.

# Answering rectangle queries



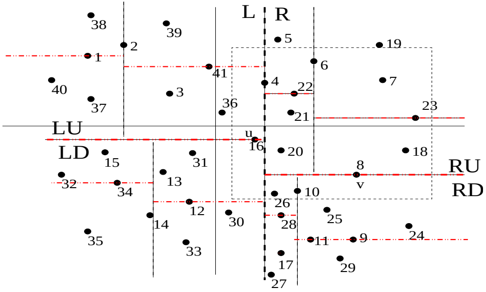

- A query rectangle $Q$ may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.

# Answering rectangle queries



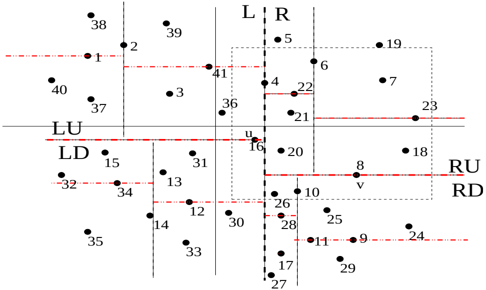

- A query rectangle $Q$ may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If $R$ contains the entire bounded $region(p)$ of a point $p$ defining a node $N$ of $T$ then report all points in $region(p)$.

# Answering rectangle queries



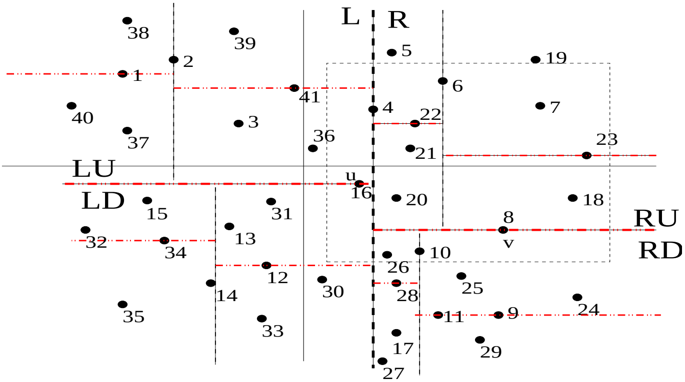

- A query rectangle $Q$ may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If $R$ contains the entire bounded $region(p)$ of a point $p$ defining a node $N$ of $T$ then report all points in $region(p)$.
- If $R$ misses the $region(p)$ then we do not treverse the subtree rooted at this node.

# Answering rectangle queries



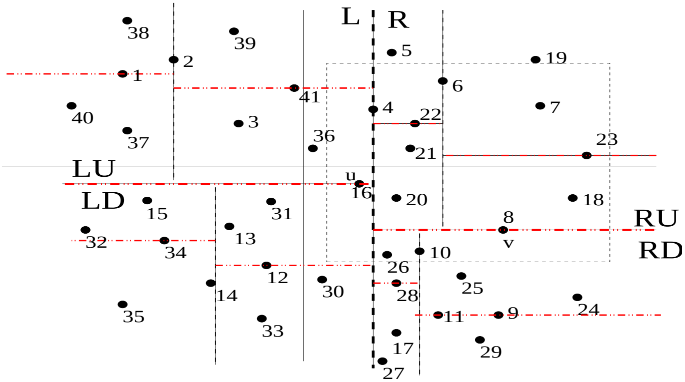

- A query rectangle $Q$ may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If $R$ contains the entire bounded $region(p)$ of a point $p$ defining a node $N$ of $T$ then report all points in $region(p)$.
- If $R$ misses the $region(p)$ then we do not treverse the subtree rooted at this node.
- If $R$ overlaps $region(p)$ then we check whether $R$ also overlaps the two regions of the children of the node $N$.

# 2-dimensional Range Searching: Kd-trees



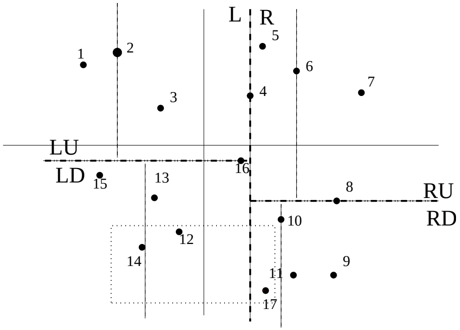

- The set $L$ ($R$) is split into two roughly equal sized subsets $LU$ and $LD$ ($RU$ and $RD$), using point $u$ ($v$) that has the median $y$-coordinate in the set $L$ ($R$), and including $u$ in $LU$ ($RU$).

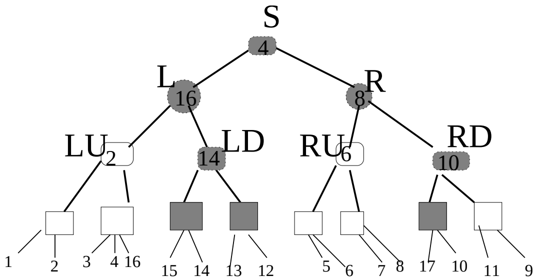
# 2-dimensional Range Searching: Kd-trees



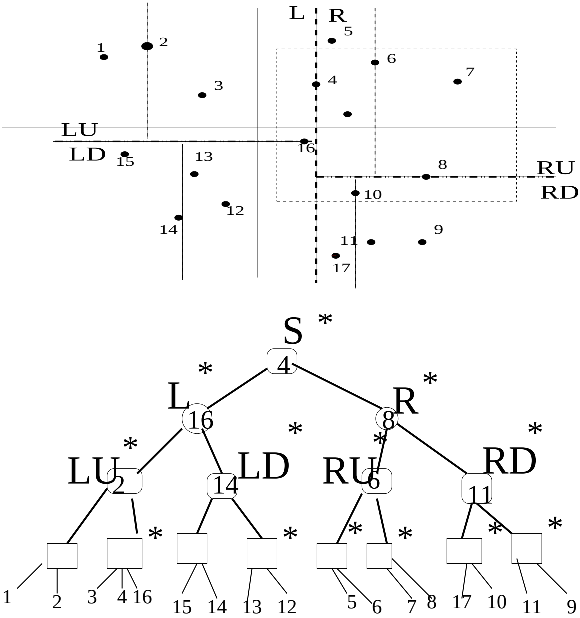

- The set $L$ ($R$) is split into two roughly equal sized subsets $LU$ and $LD$ ($RU$ and $RD$), using point $u$ ($v$) that has the median $y$-coordinate in the set $L$ ($R$), and including $u$ in $LU$ ($RU$).
- The entire halfplane containing set $L$ ($R$) is called the *region*($u$) (*region*($v$)).

# Nodes traversed in the Kd-tree



# Nodes traversed in the Kd-tree



# Time complexity of output point reporting

- Reporting points within $R$ contributes to the output size $k$ for the query.

# Time complexity of output point reporting

- Reporting points within $R$ contributes to the output size $k$ for the query.
- No leaf level region in $T$ has more than 2 points.

# Time complexity of output point reporting

- Reporting points within $R$ contributes to the output size $k$ for the query.
- No leaf level region in $T$ has more than 2 points.
- So, the cost of inspecting points outside $R$ but within the intersection of leaf level regions of $T$ can be charged to the internal nodes traversed in $T$.

# Time complexity of output point reporting

- Reporting points within $R$ contributes to the output size $k$ for the query.
- No leaf level region in $T$ has more than 2 points.
- So, the cost of inspecting points outside $R$ but within the intersection of leaf level regions of $T$ can be charged to the internal nodes traversed in $T$.
- This cost is borne for all leaf level regions intersected by $R$.

# Worst-case cost of traversal

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.

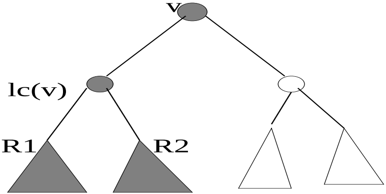
# Worst-case cost of traversal

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting $S$ can intersect either $L$ or $R$ but not both, but it can meet both $RU$ and $RD$ ($LU$ and $LD$).

# Worst-case cost of traversal

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting $S$ can intersect either $L$ or $R$ but not both, but it can meet both $RU$ and $RD$ ($LU$ and $LD$).
- Any horizontal line intersecting $R$ can intersect either $RU$ or $RD$ but not both, but it can meet both children of $RU$ ($RD$).

# Time complexity of rectangle queries for Kd-trees

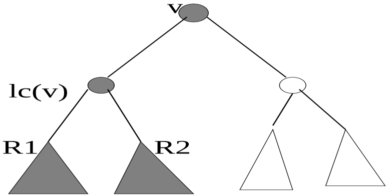



- Therefore, the time complexity $T(n)$ for an $n$-vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T(\frac{n}{4})$$

$$T(1) = 1$$

# Time complexity of rectangle queries for Kd-trees



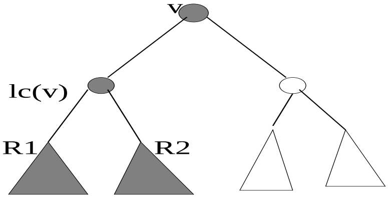

- Therefore, the time complexity $T(n)$ for an $n$-vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T(\frac{n}{4})$$

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt(n))$.

# Time complexity of rectangle queries for Kd-trees



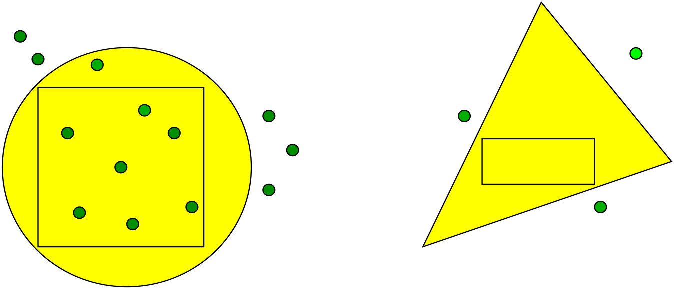

- Therefore, the time complexity $T(n)$ for an $n$-vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T(\frac{n}{4})$$

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt(n))$.
- The total cost of reporting $k$ points in $R$ is therefore $O(\sqrt(n) + k)$.

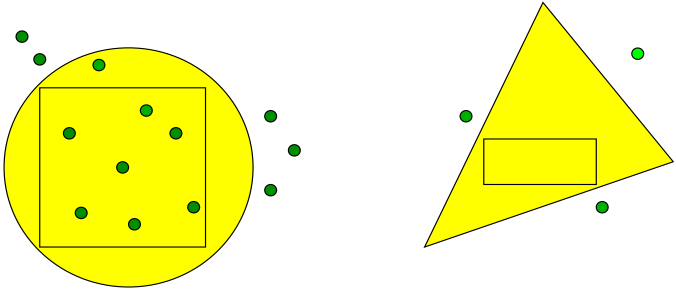
# More general queries



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.

# More general queries



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.
- Circles cannot be simulated by triangles either.

# Triangle queries

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k))$ time, where $k$ is the number of points inside the query triangle.

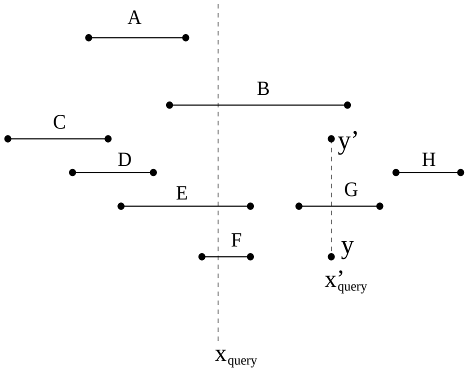
Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

# Triangle queries

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k))$ time, where $k$ is the number of points inside the query triangle.
- For counting the number $k$ of points inside a query triangle, worst-case optimal $O(\log n)$ time suffices.

Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

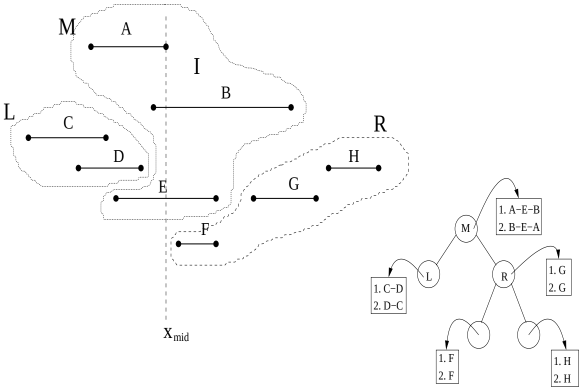
# Finding intervals containing a vertical query line/segment




Simpler queries ask for reporting all intervals intersecting the vertical line $X = x_{query}$.

More difficult queries ask for reporting all intervals intersecting a vertical segment joining $(x'_{query}, y)$ and $(x'_{query}, y')$.

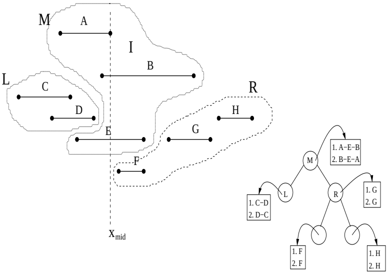
# Constructing the interval tree




The set $M$ has intervals intersecting the vertical line $X = x_{mid}$, where $x_{mid}$ is the median of the x-coordinates of the $2n$ endpoints.

The root node has intervals $M$ sorted in two independent orders (i) by right end points (B-E-A), and (ii) left end points (A-E-B).

# Answering queries using an interval tree



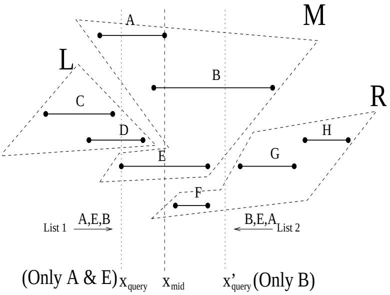

The set $L$ and $R$ have at most $n$ endpoints each.

So they have at most $\frac{n}{2}$ intervals each.

Clearly, the cost of (recursively) building the interval tree is $O(n \log n)$.

The space required is linear.

# Answering queries using an interval tree

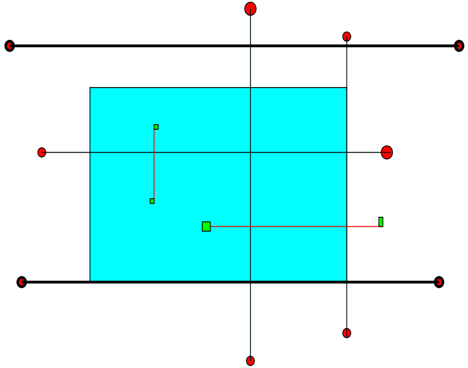



For $x_{query} < x_{mid}$, we do not traverse subtree for subset $R$.

For $x'_{query} > x_{mid}$, we do not traverse subtree for subset $L$.

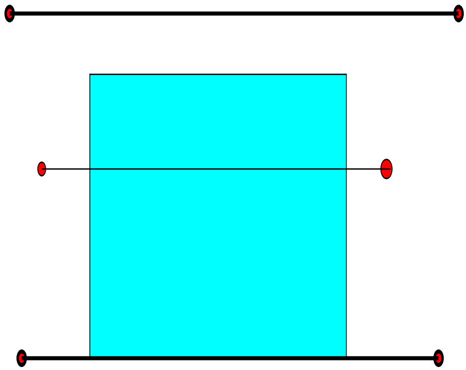
Clearly, the cost of reporting the $k$ intervals is $O(\log n + k)$.

# Reporting (portions of) all rectilinear segments inside a query rectangle



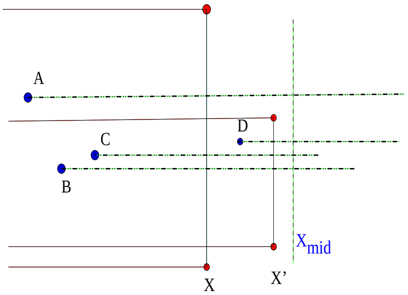
For detecting segments with one (or both) ends inside the rectangle, it is sufficient to maintain rectangular range query apparatus for output-sensitive query processing.

# Reporting segments with no endpoints inside the query rectangle



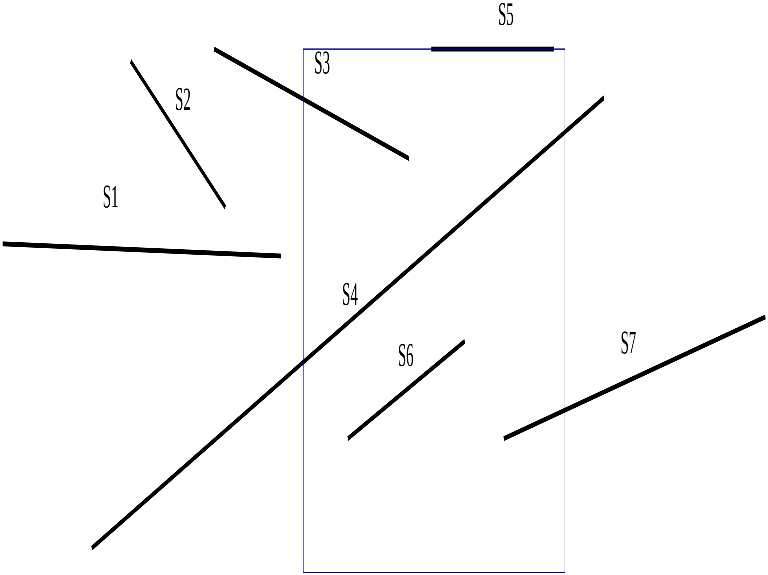
Report all (horizontal) segments that cut across the query rectangle or include an entire (top/bottom) bounding edge.
Use either the right (or left) edge, and the top (or bottom) edge of the query rectangle.

# Right edges X and X' of two query rectangles

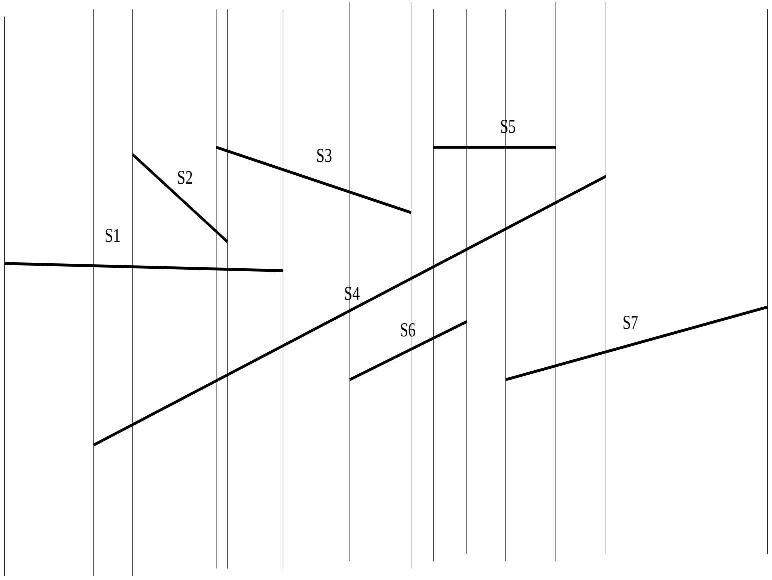



Use an interval tree of all the horizontal segments and the right bounding edge of the query rectangle like X or X'.
This helps reporting all segments cutting the right edge of the query rectangle.
Use the rectangle query for vertical segment X and find points A, B and C in the rectangle with left edge at minus infinity. For X', report B, C and D, similarly.

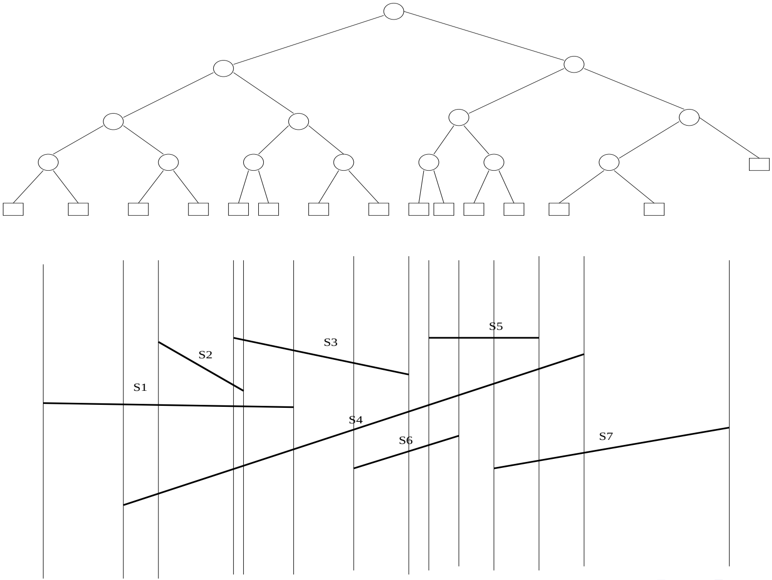
# Reporting (portions) of slanted edges inside a query rectangle



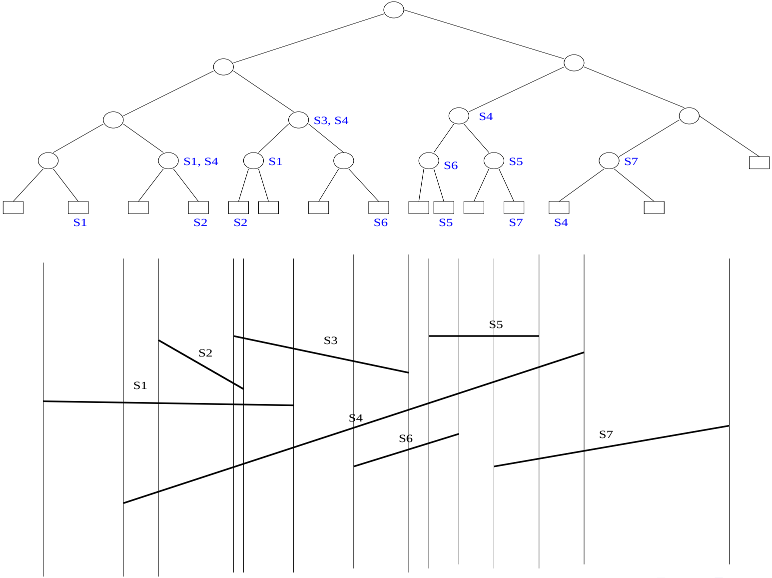
# Sort the 2$n$ endpoints by x-coordinates



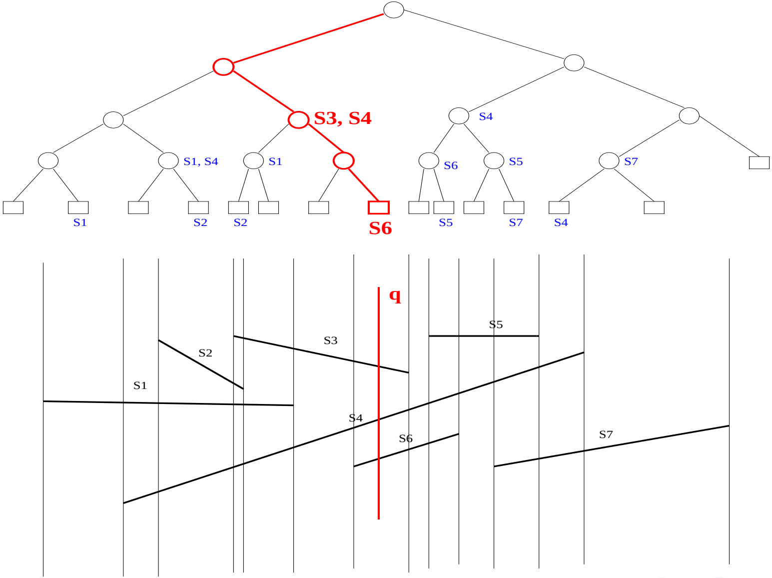
# Build a balanced binary search tree of $2n + 1$ intervals



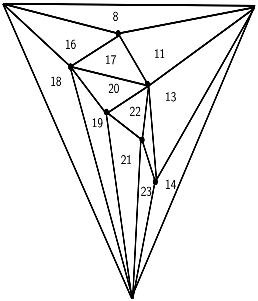
# Represent each segment at nodes whose decendants it covers



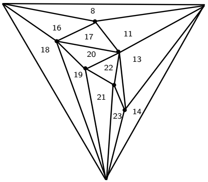
# Collect segments intersecting a rectangle's edge

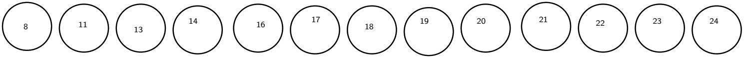


# Planar point location by triangulation refinement

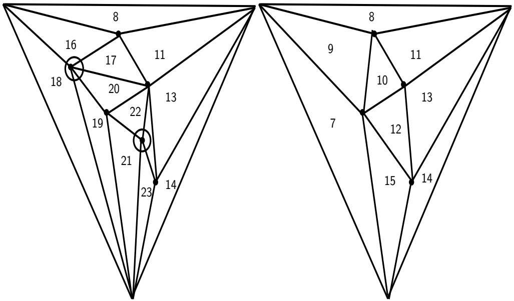


# Each triangle is represented by a node

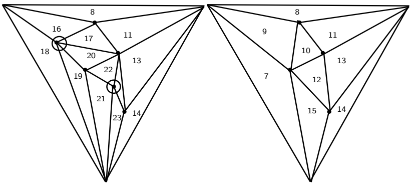


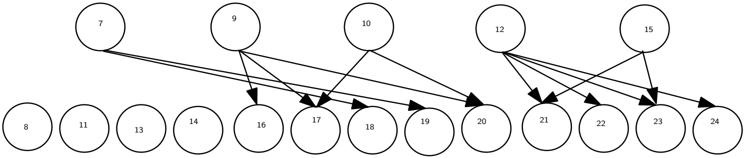



# Each (new) gross triangle overlaps some finer triangles of the lowest layer

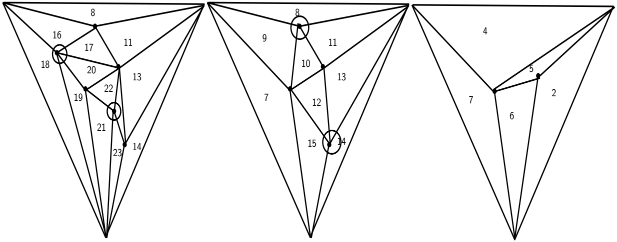


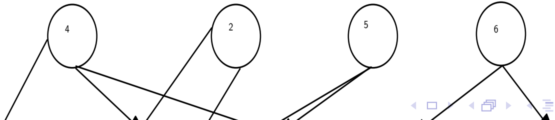
# The second layer of gross triangulation



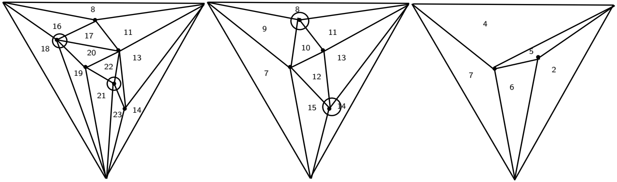



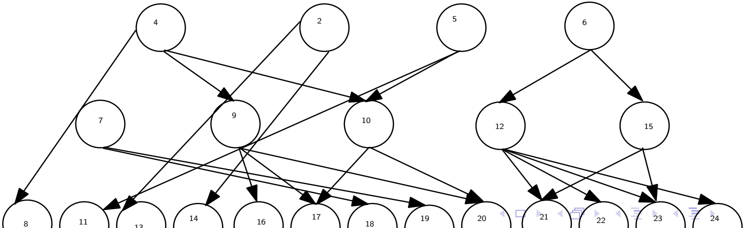
# Each (new) gross triangle of the third layer overlaps some finer triangles of the second layer



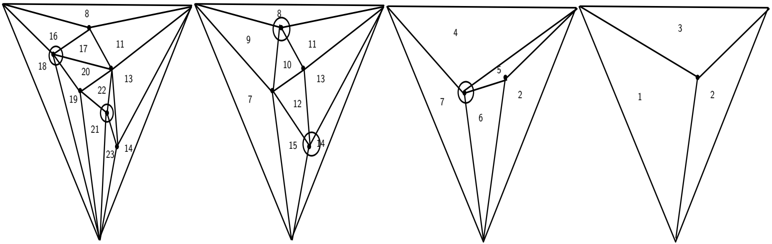



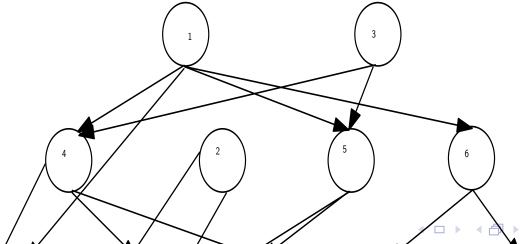
# Grosser traingulations are formed by dropping mutually non-adjacent vertices of degree $\leq 12$ and retriangulating.



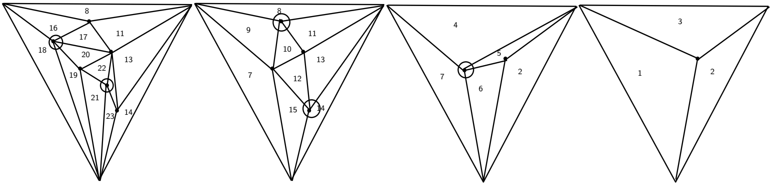



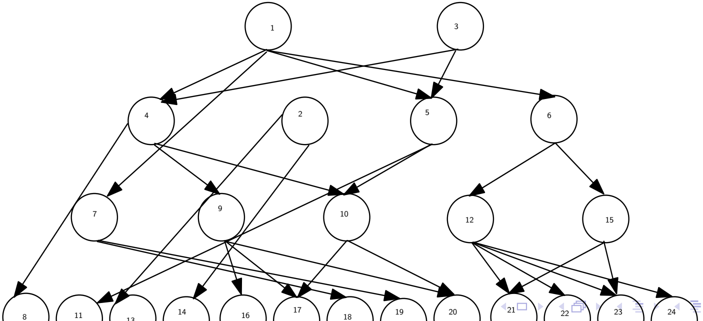
# Each (new) gross triangle of the fourth layer overlaps some finer triangles of the third layer. What is the space complexity?



# The DAG so constructed has bounded (constant) fanout, yielding overall $O(\log n)$ query time.



- Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld and Mark Overmars, Computational Geometry: Algorithms and Applications, Springer.
- S. K. Ghosh, Visibility Algorithms in the Plane, Cambridge University Press, Cambridge, UK, 2007.
- Kurt Mehlhorn, Data Structures and Algorithms, Vol. 3, Springer.
- F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, New York, NY, Springer-Verlag, 1985.