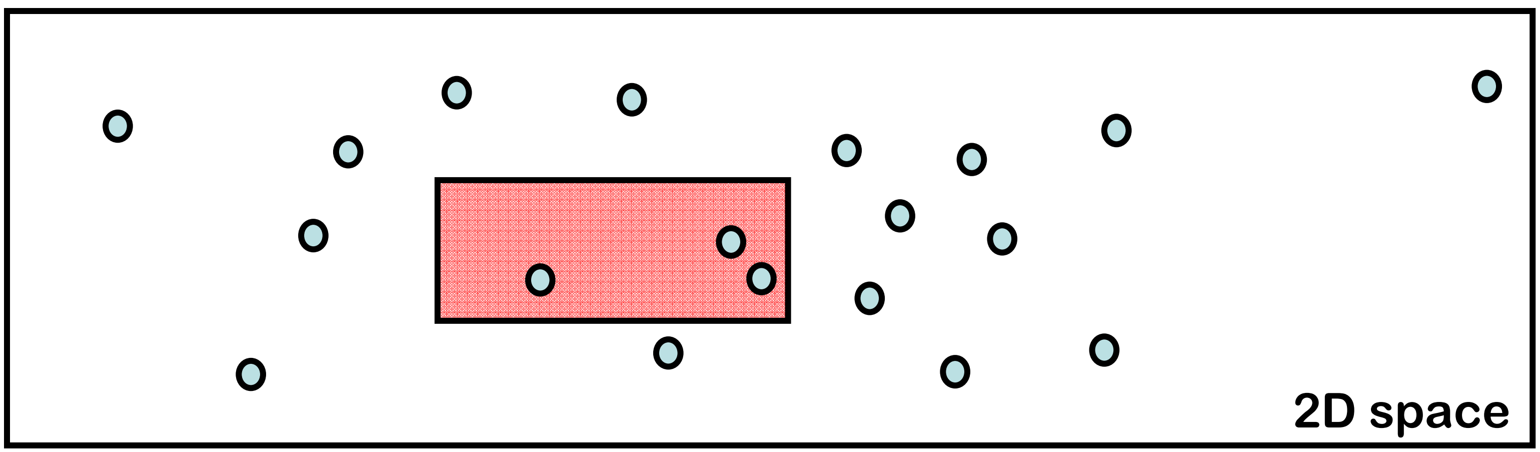
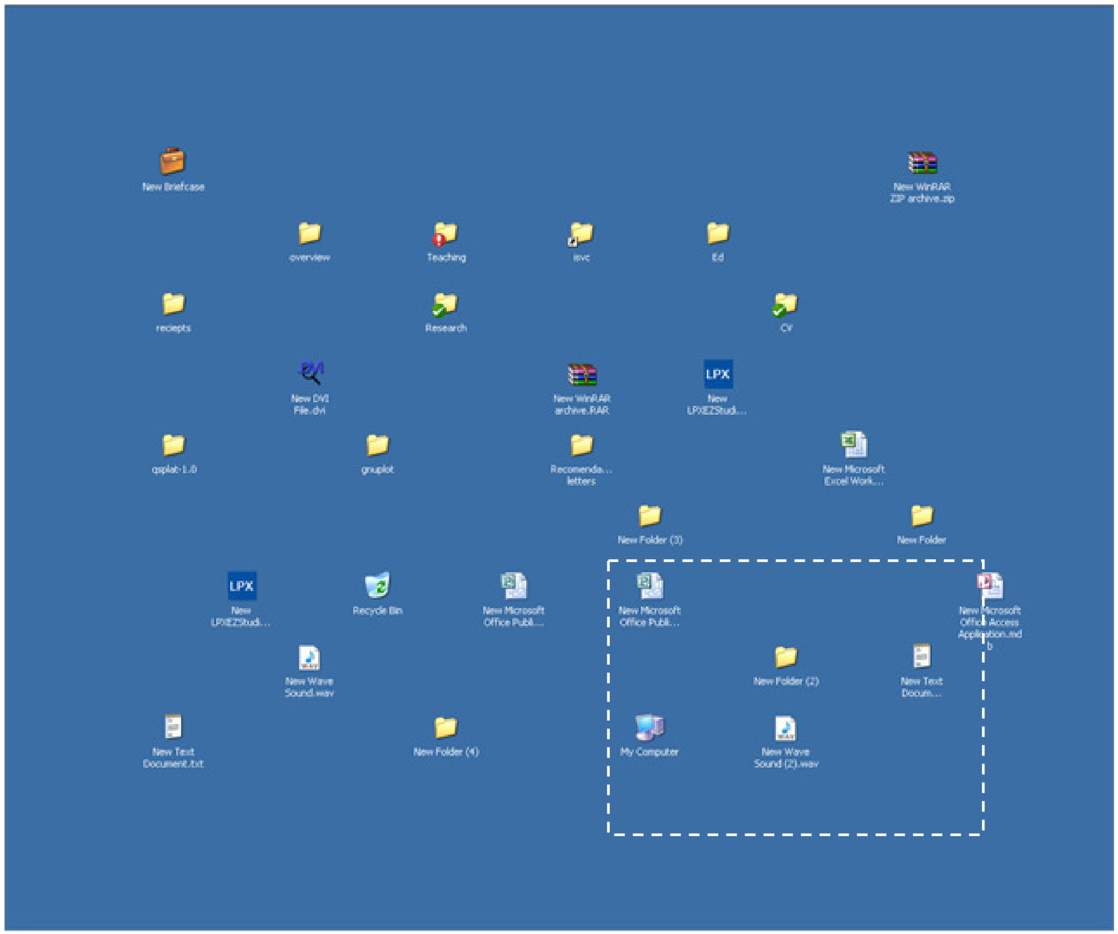
Orthogonal Range Searching

- Given a set of k-D points and an orthogonal range (whose boundaries are parallel to the coordinate axes), find all points enclosed by this query range



- Brute force: O(n), is this necessary?

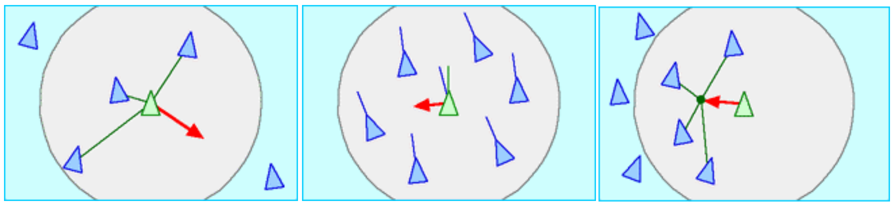
Selecting Desktop Icons



Orthogonal Range Searching

- Driving Applications
  - Database
  - Geographic Information System
  - Simulating group behaviors (bird homing)

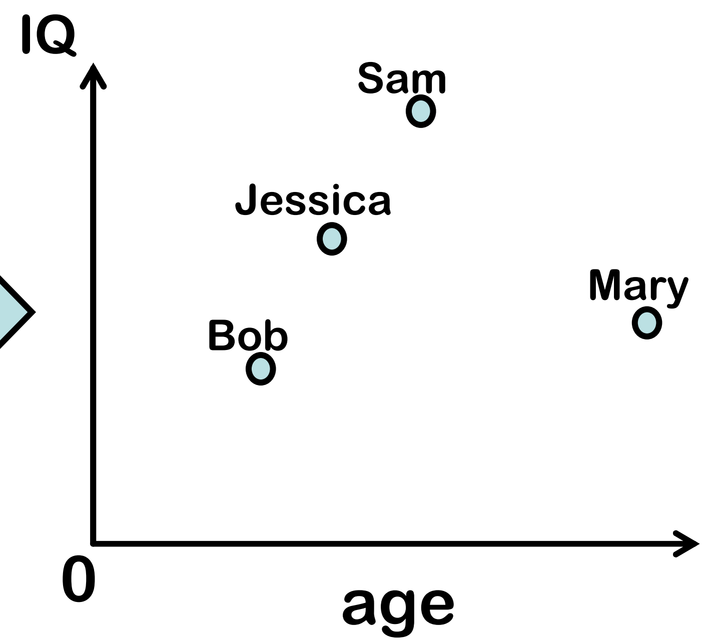
QuickTime™ and a
YUV420 codec decompressor
are needed to see this picture.



Interpret DB Queries Geometrically

- Transform records in database into points in multi-dimensional space.

name	age	IQ
Bob	12	75
Jessica	21	132
Mary	88	89
Sam	34	180

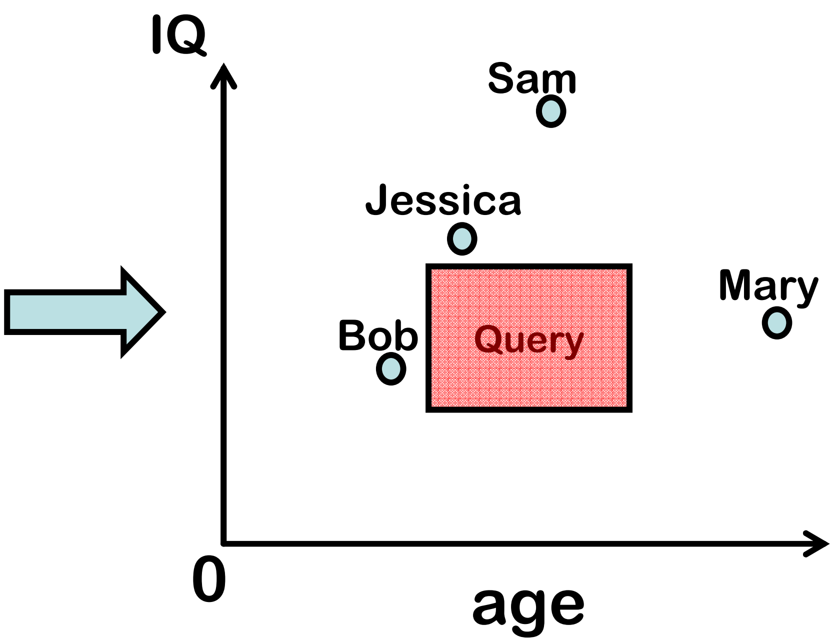


Interpret DB Queries Geometrically

- Transform queries on d-fields of records in the database into queries on this set of points in d-dimensional space

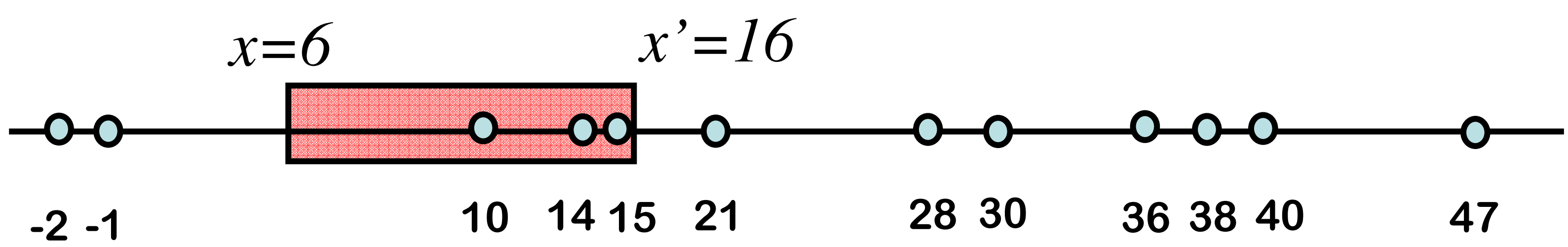
Query: age between 18 and 38, IQ between 70 and 110

name	age	IQ
Bob	12	75
Jessica	21	132
Mary	88	89
Sam	34	180



1-D Range Searching

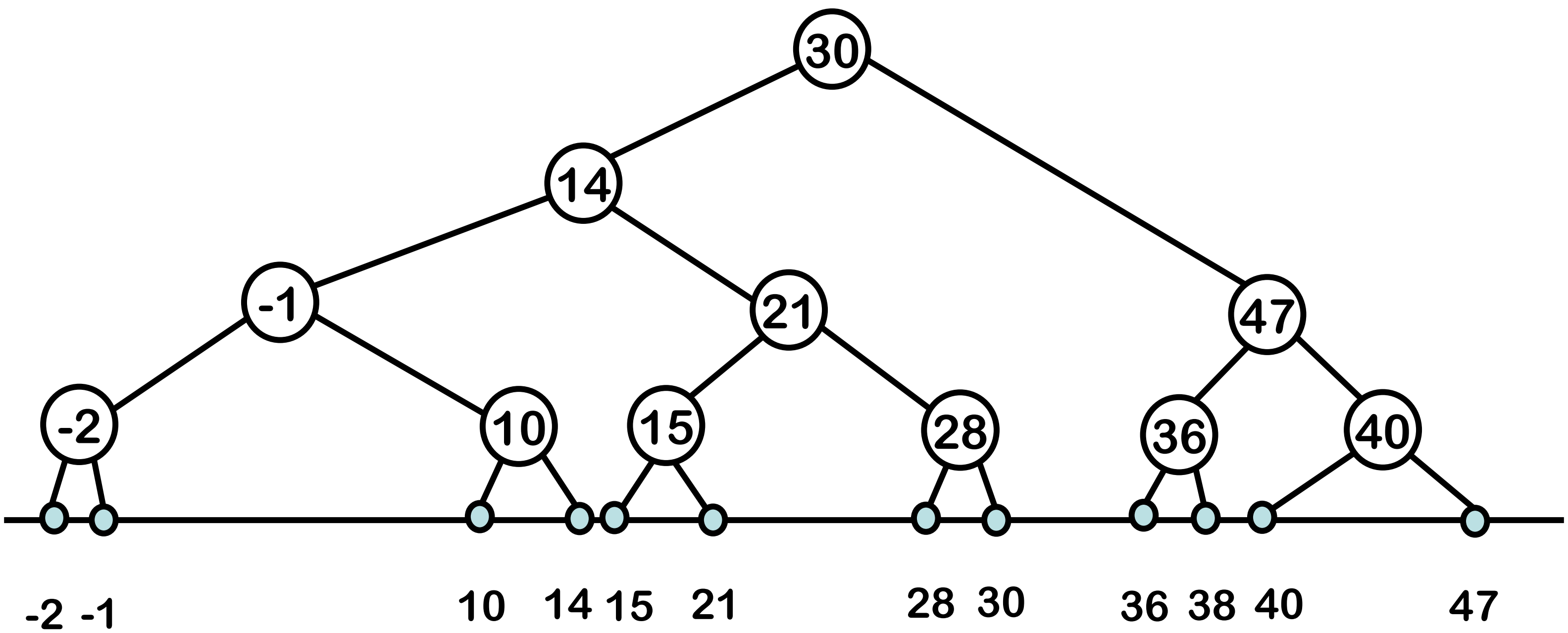
- **Let's solve a simple problem first**
  - **Let $P := \{p_1, p_2, \ldots, p_n\}$ be a given set of points on the real line. A query asks for the points inside a 1-D query rectangle -- i.e. an interval $[x:x']$**



George Mason University
Department of Computer Science
Volgenau School of Information Technology and Engineering

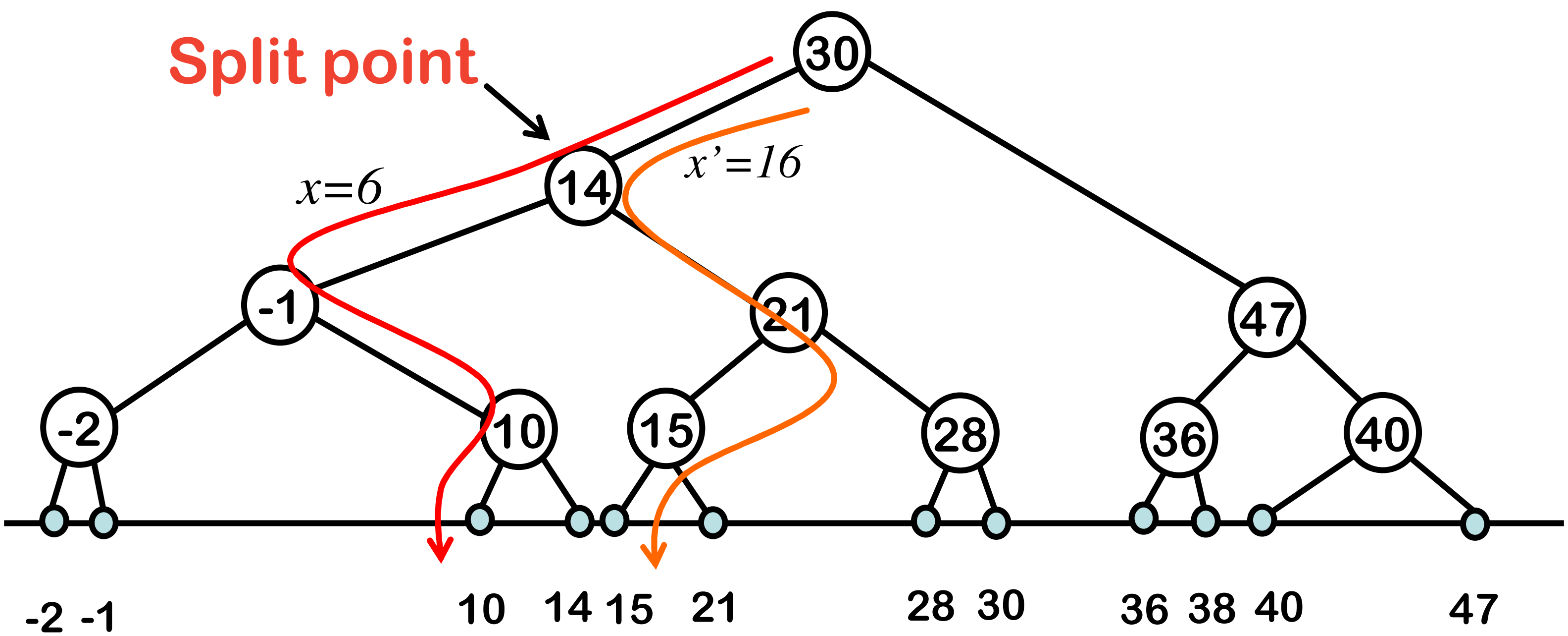
1-D Range Searching

- Use a balanced binary search tree T.
  - The leaves of T store the points of P
  - The internal nodes of T store splitting values to guide the search
    - The largest value in the left sub-tree



1-D Range Searching

- To report points in [x:x'], we search with x and x' in T.
  - Let u and u' be the two leaves where the search ends resp.
  - Then the points in [x:x'] are the ones stored in leaves between u and u', plus possibly points stored at u & u'.



1D Range Query

Input: A range tree T and a range $[x:x']$
Output: All points that lie in the range.

1. $v_{split} \leftarrow \text{FindSplitNode}(T, x, x')$
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported
4. **else** (* Follow the path to x and report the points in subtrees right of the path *)
5. $v \leftarrow lc(v_{split})$
6. **while** v is not a leaf
7. **do** **if** $x \leq x_v$
8. **then** ReportSubTree($rc(v)$) → move to left
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$ → move to right
11. Check if the point stored at leaf v must be reported
12. Similarly, follow the path to x'

Find Split Node

Input: A tree T and two values x and x' with $x \leq x'$

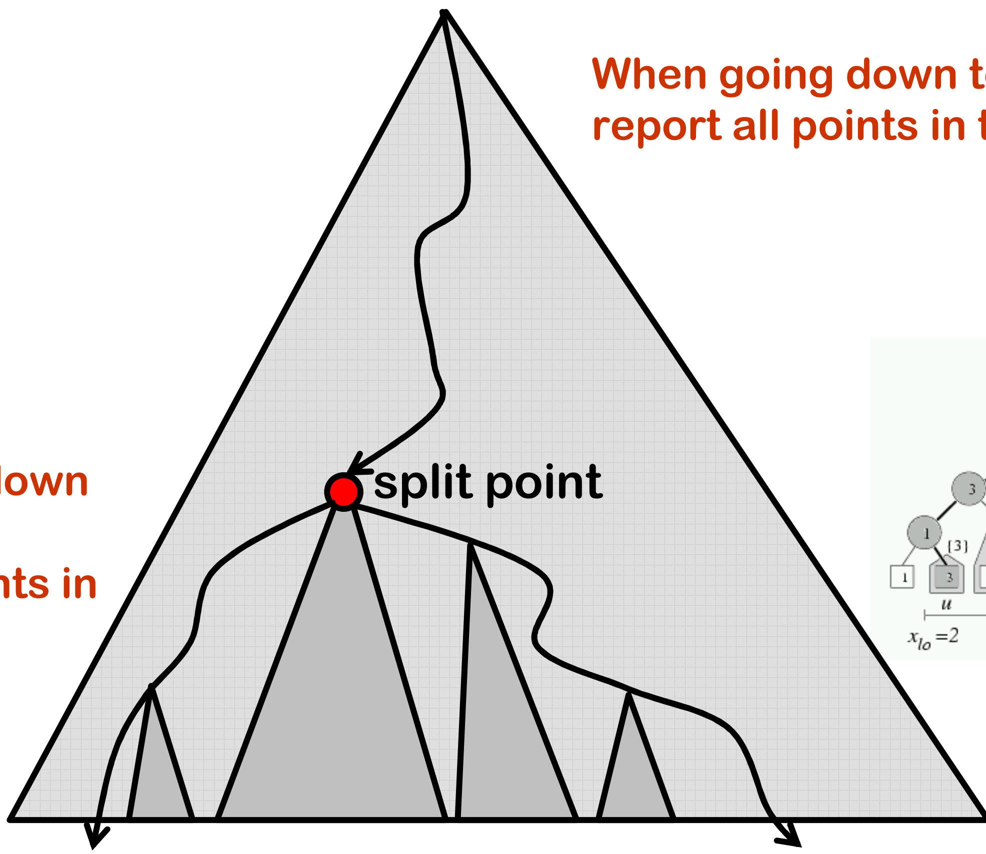
Output: The node v where the paths to x and x' splits, or the leaf where both paths end.

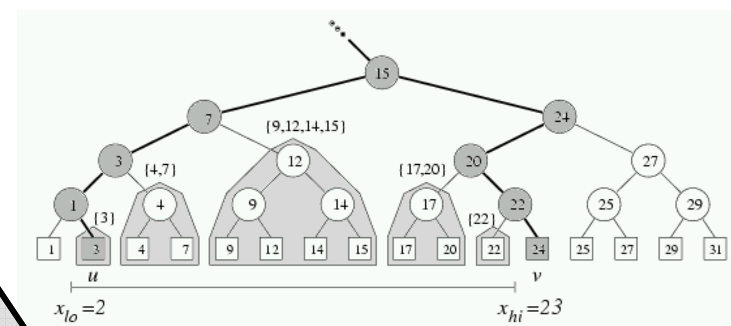
1. $v \leftarrow root\ (T)$
2. while v is not a leaf and ($x' \leq x_v$ or $x > x_v$)
3. do if $x' \leq x_v$
4. then $v \leftarrow lc(v)$ (* left child of the node v *)
5. else $v \leftarrow rc(v)$ (* right child of the node v *)
6. return v

1-D Range Searching

When going down to the right, report all points in the left

When going down to the left, report all points in the right





1D-Range Search Algorithm Analysis

- **Let P be a set of n points in one-dimensional space**
  - **uses $O(n)$ storage and has $O(n \log n)$ construction time**
  - **The points in a query range can be reported**
    - **Time $O(k + \log n)$, where k is the number of reported points**
      - **The time spent in "ReportSubtree" is linear in the number of reported points, i.e. $O(k)$.**
      - **The remaining nodes that are visited on the search path of x or x'. The length is $O(\log n)$ and time spent at each node is $O(1)$, so the total time spent at these nodes is $O(\log n)$.**