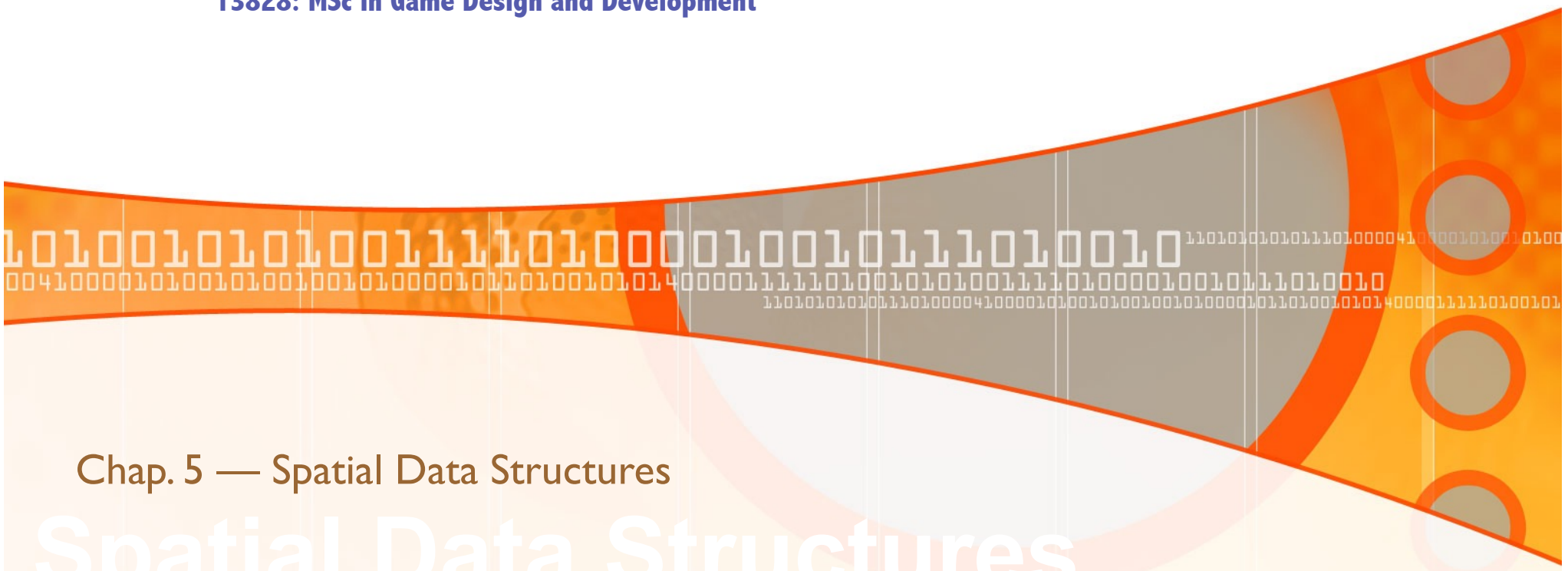
# Computer Graphics for Digital Games
# Video Game Technologies

**14475: MSc in Computer Science and Engineering**
**13828: MSc in Game Design and Development**



Chap. 5 — Spatial Data Structures

Spatial Data Structures

# Outline

*Based on:*
https://courses.cs.washington.edu/courses/cse373/02wi/

...:

- Introduction.
- Taxonomy of spatial data structures.
- K-d trees: overview, construction, and complexity.
- K-d trees: rectangular range query, nearest neighbor search.
- Point quadtrees: overview, construction, and complexity.
- Point quadtrees: point insertion, point deletion, rectangular range query, nearest neighbor search.
- Region quadtrees.

# Spatial data structures: introduction

**Leading idea:**

- Spatial organization of points, lines, planes, etc., in support of *faster processing*.

**Applications:**

- Map information
- Graphics - computing object intersections
- Data compression - nearest neighbor search
- Decision Trees - machine learning

**Taxonomy:**

- K-d trees
- Quadtrees
- BSP trees



# K-d Trees

# K-d trees

*Trees used to store spatial data.*

**Contribution:**

- They were introduced by Jon Bentley while an undergraduate (1975).

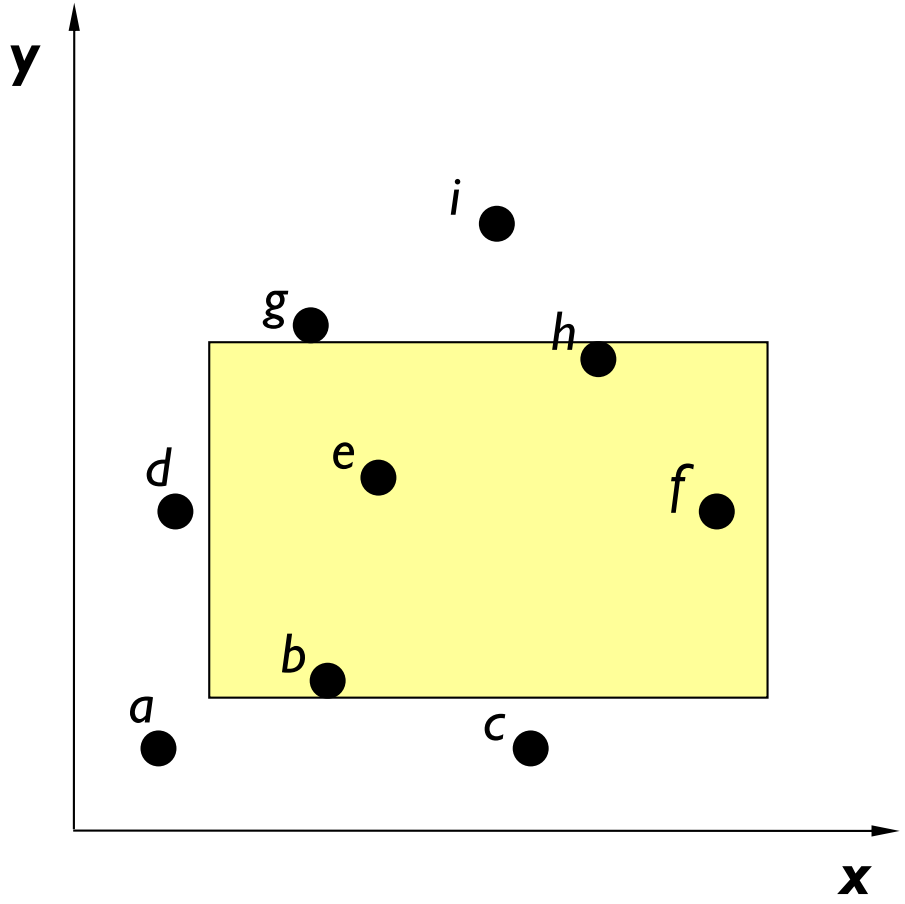
**Utilities:**

- Nearest neighbor search.
- Range queries.
- Fast look-up

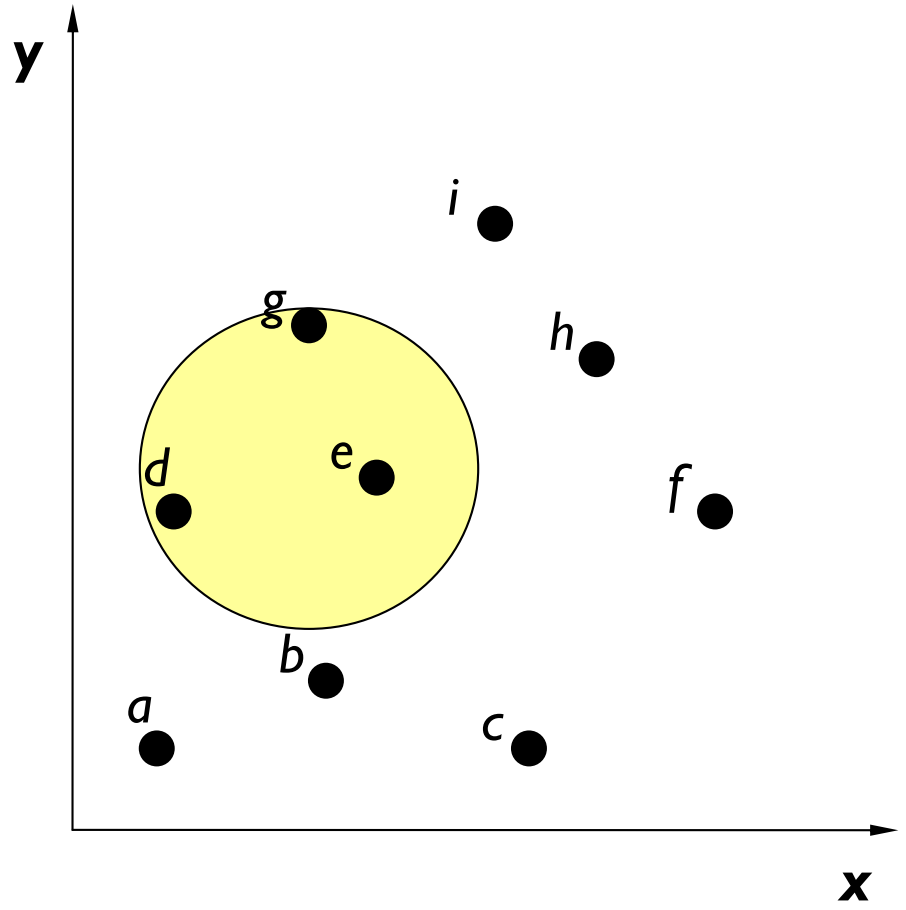
**Complexity:**

- k-d tree are $\log_2$ n depth where n is the number of points in the set.
- Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

# Range queries

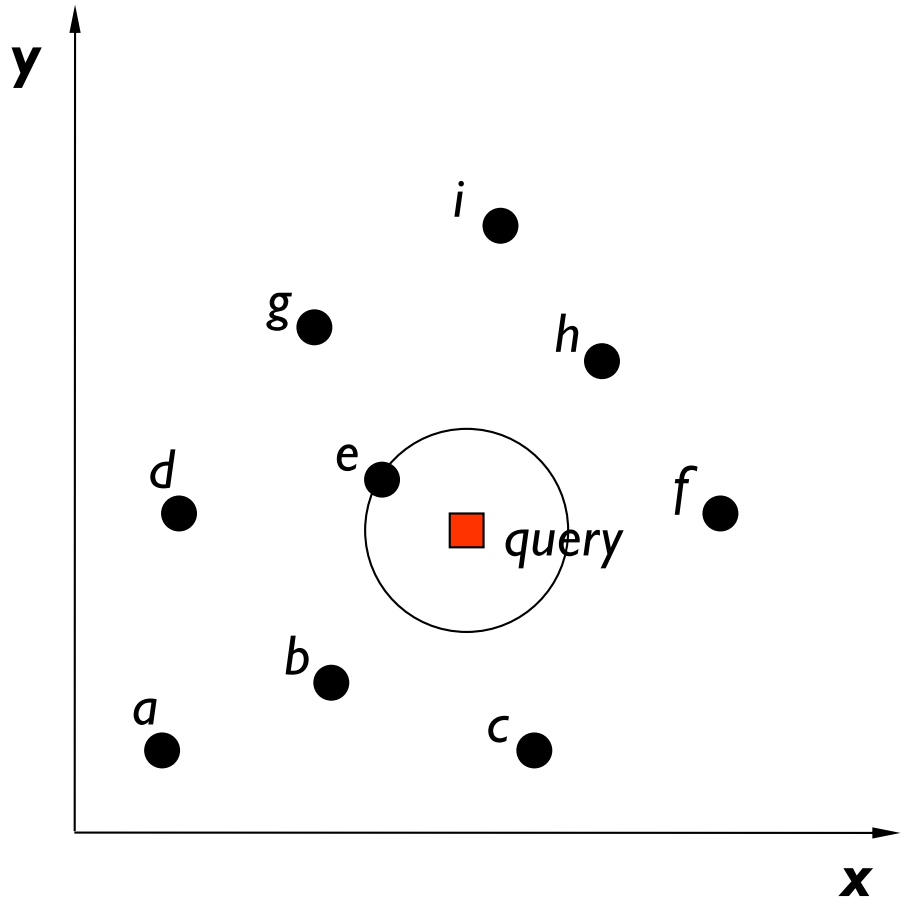



*Rectangular query*




*Circular query*

# Nearest neighbor search




*Nearest neighbor is e.*

# K-d tree construction

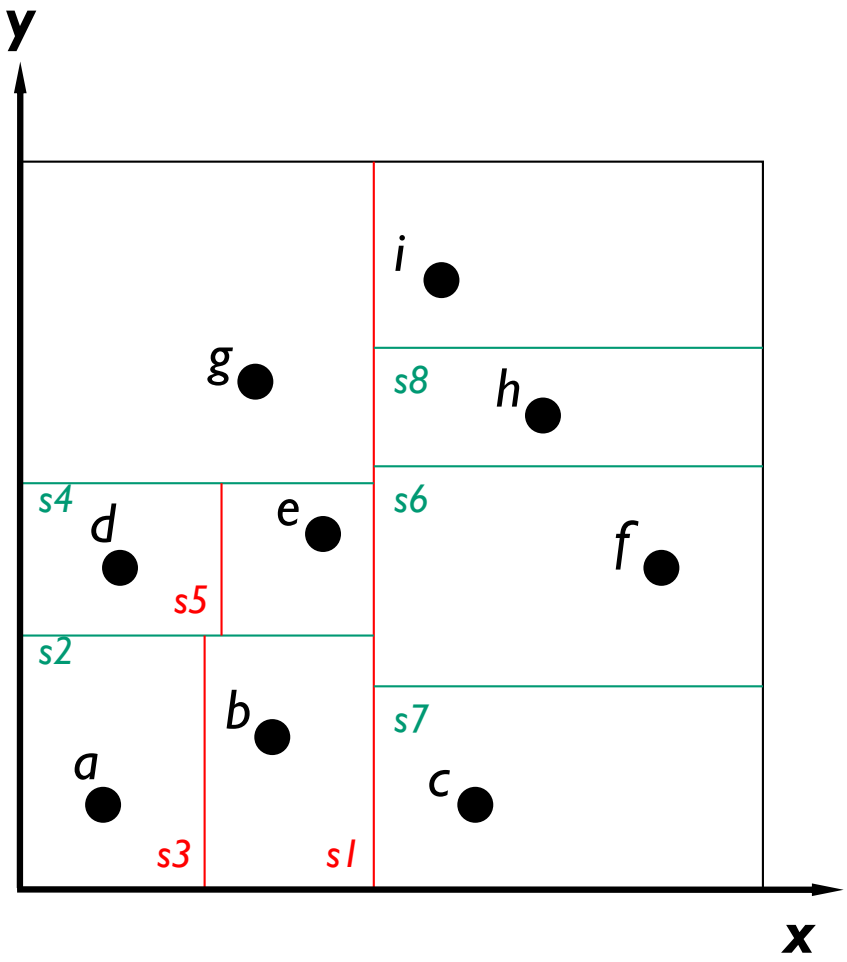
*Algorithm outline:*

1. If there is just one point, form a leaf with such a point.
2. Otherwise, divide the points in half by a line perpendicular to one of the axes.
3. Recursively construct k-d trees for the two sets of points.

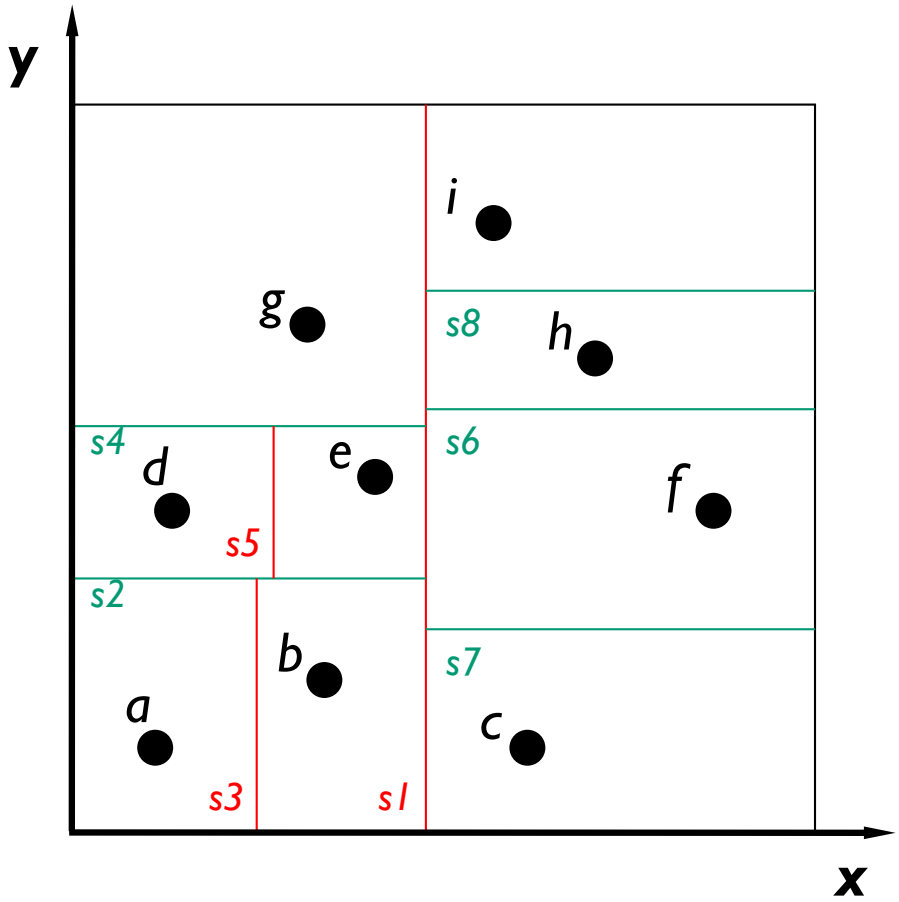
***Division strategies*** (step 2 above):

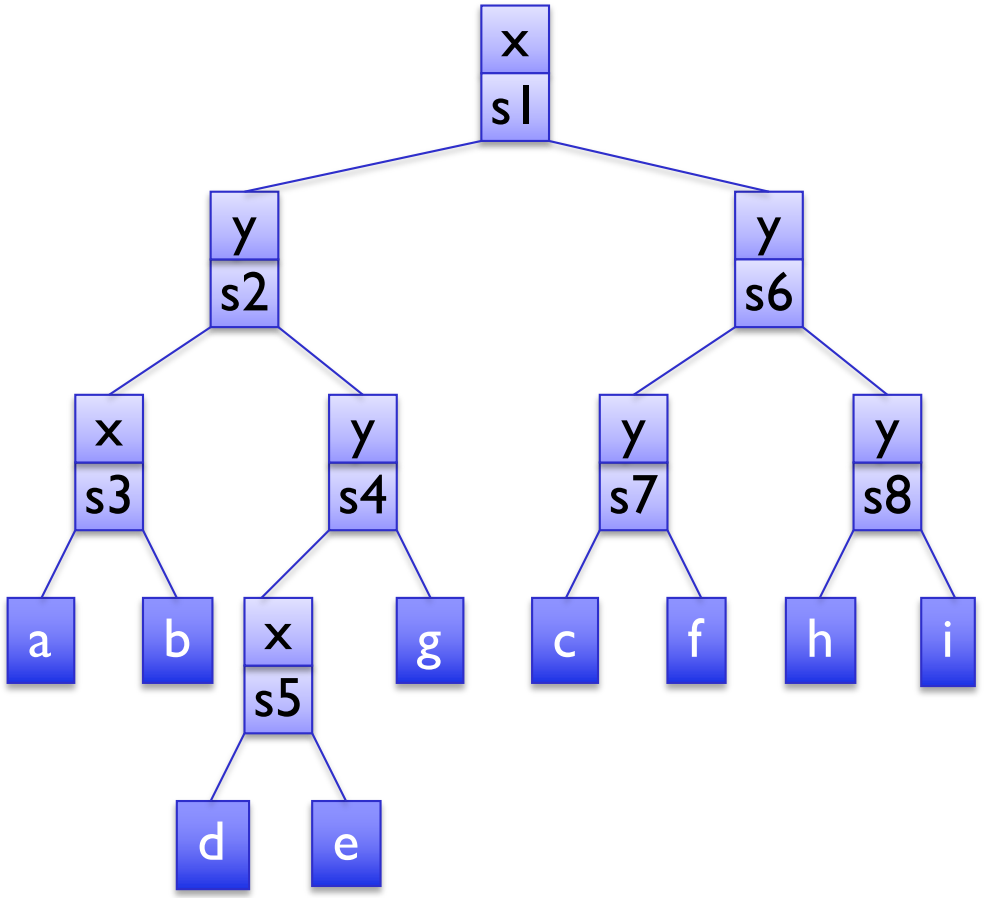
- divide points perpendicular to the axis with widest spread.
- divide in a round-robin fashion (x first, then y)

***Tree node representation***: A node has 5 fields: axis (splitting axis), value (splitting value), left (subtree), right (subtree), point (holds a point if left and right children are null)



# K-d tree construction *(contd.)*



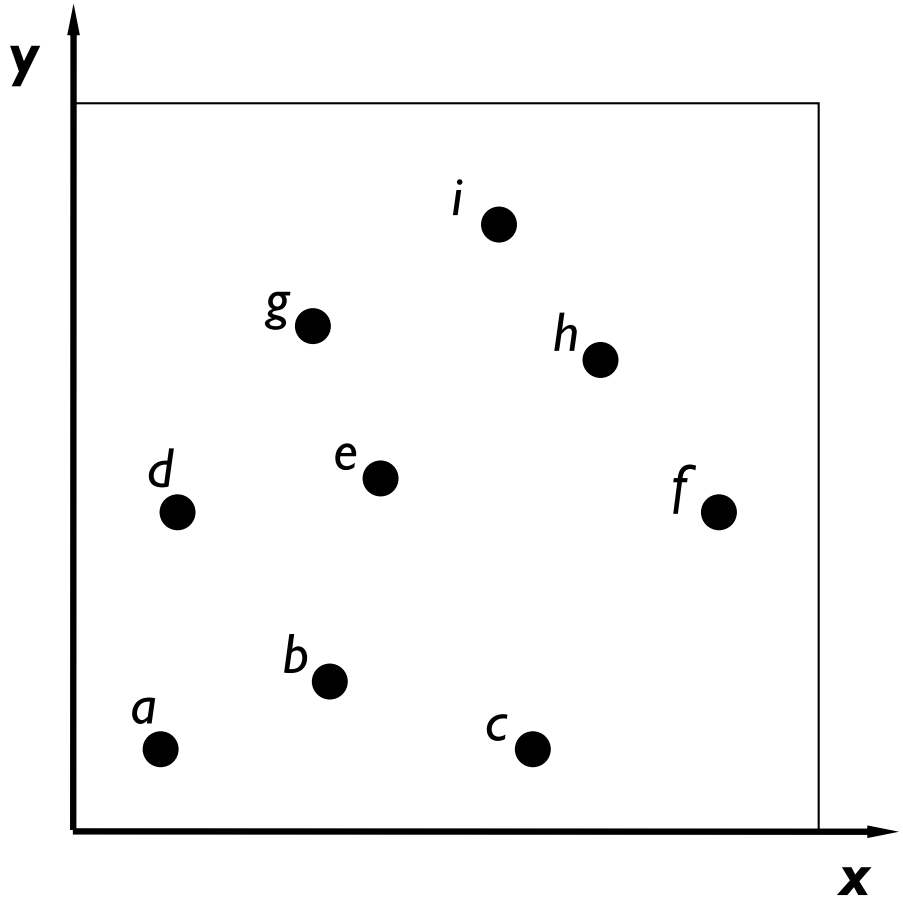



# K-d tree splitting (using widest spread)

- Sort points in each dimension

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **x** | a | d | g | b | e | i | c | h | f |
| **y** | a | c | b | d | f | e | h | g | i |

- max spread is the max of $f_x - a_x$ and $i_y - a_y$.
- In the selected dimension the middle point in the list splits the data.
- To build the sorted sublists for the other dimensions, scan the sorted list adding each point to one of two sorted lists.



# K-d tree splitting *(contd.)*

- Sorted points in each dimension.

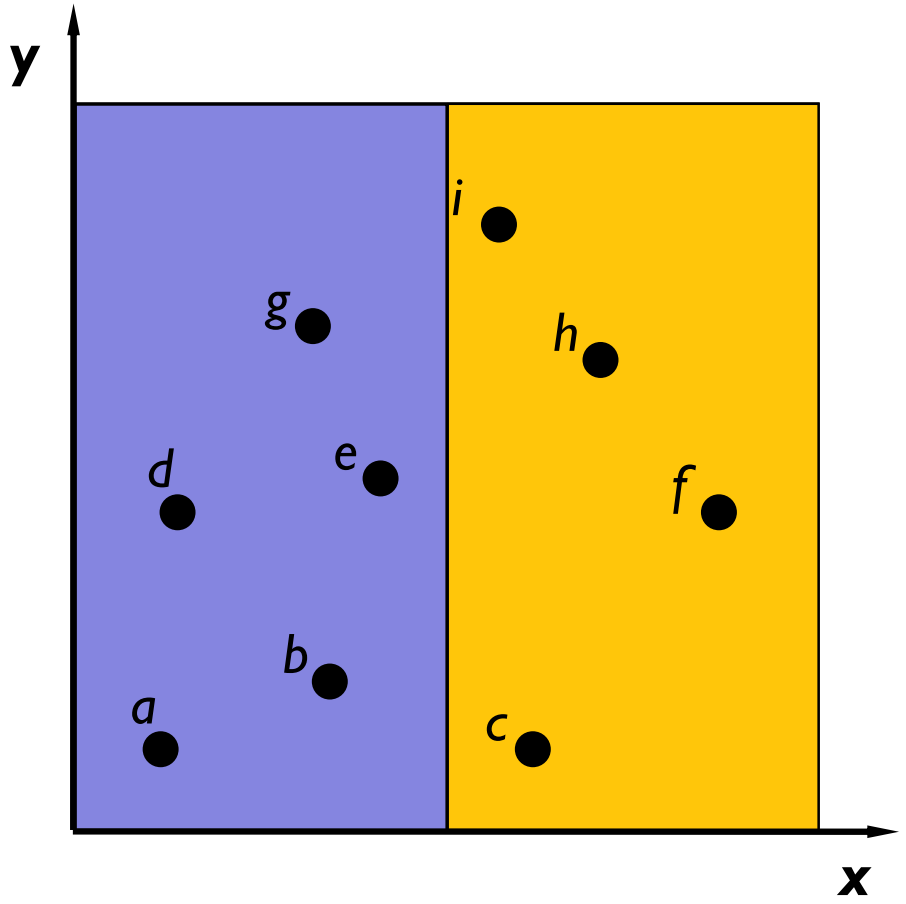
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **x** | a | d | g | b | e | i | c | h | f |
| **y** | a | c | b | d | f | e | h | g | i |

- Indicator for each set

| a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

- Scan sorted points in y dimension and add to correct set

| **y** | a | b | d | e | g | c | f | h | i |
|---|---|---|---|---|---|---|---|---|---|



# K-d tree construction complexity

First sort the points in each dimension.

- O(d.n.log n) time and d.n storage.
- These are stored in A[1..d, 1..n]

Finding the widest spread and equally divide into two subsets can be done in O(d.n) time.

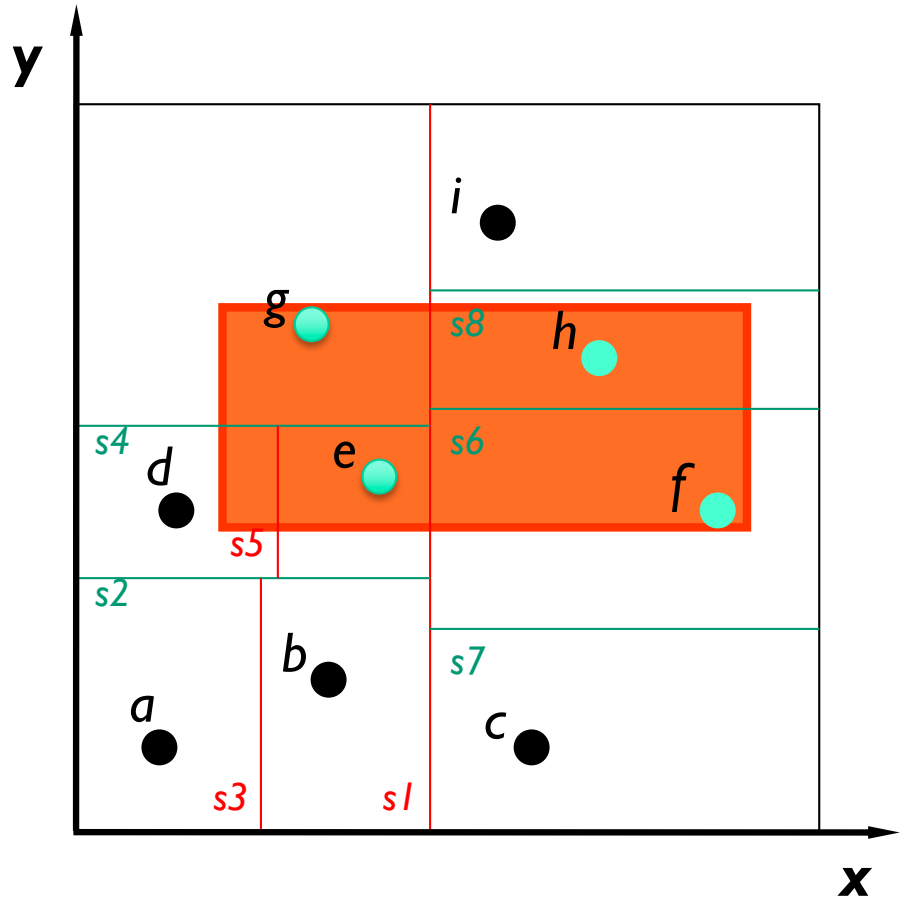
We have the recurrence:

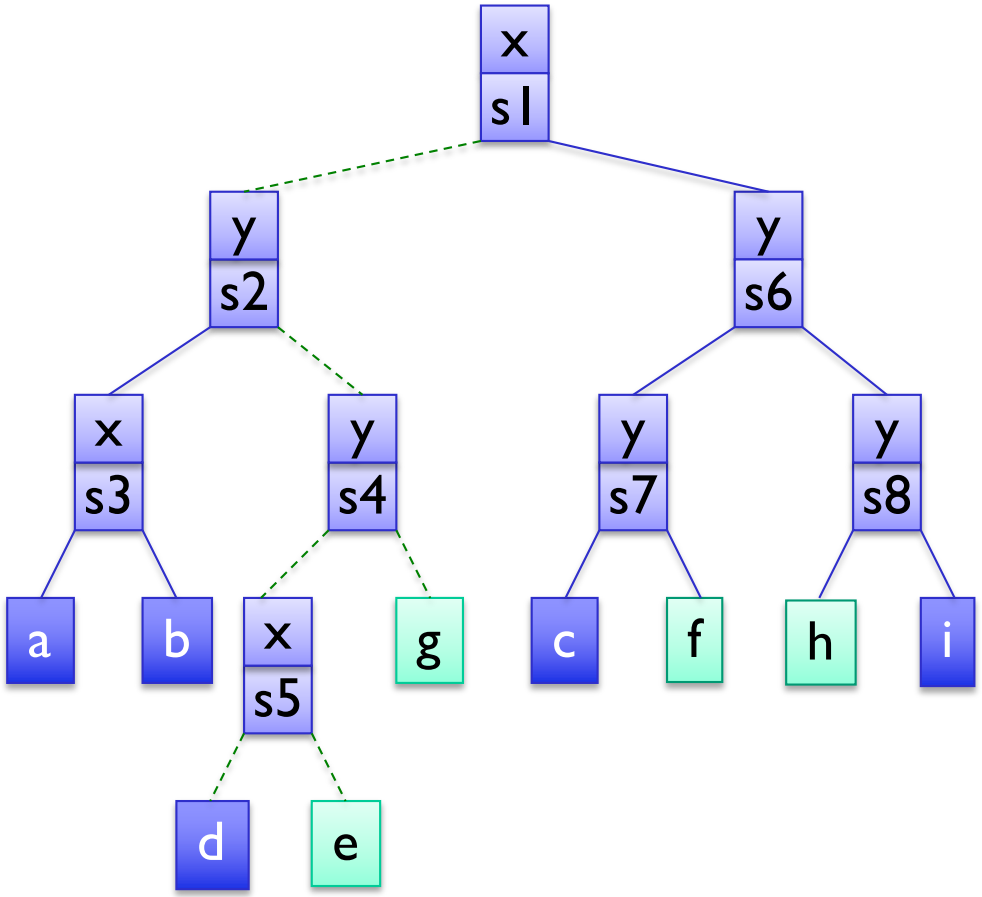
- $T(n,d) \leq 2T(n/2,d) + O(dn)$

Constructing the k-d tree can be done in O(d.n.log n) and d.n storage

# Rectangular range query

- Recursively search every cell that intersects the rectangle.
- Thus, this problem reduces to another problem: box-box intersection.



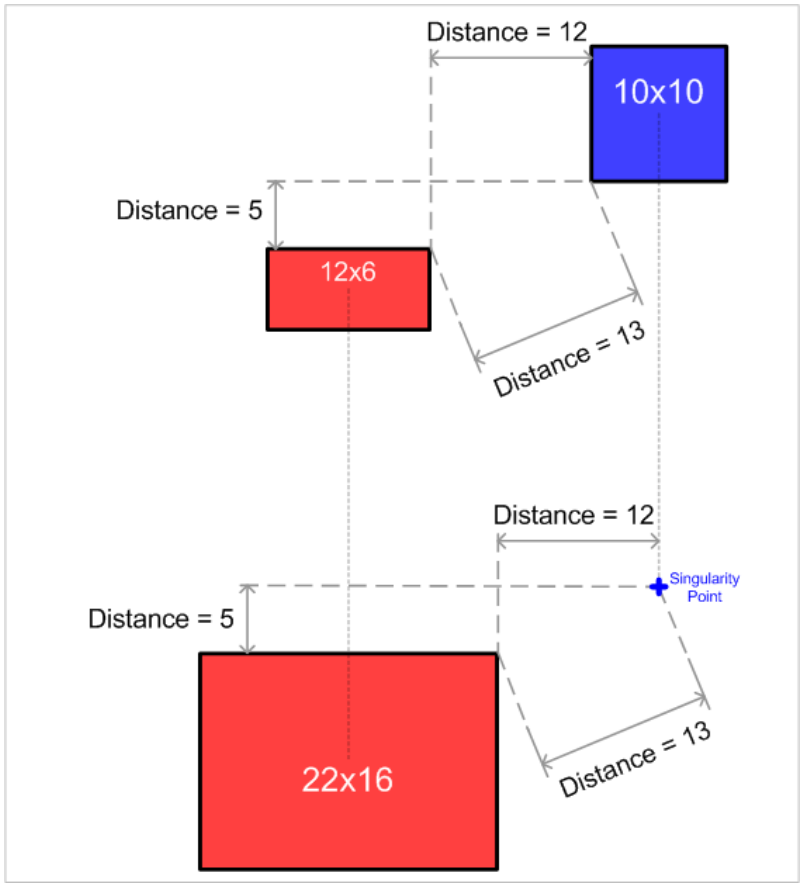
# Rectangular range query (*contd.*)

**Algorithm 1** (box-box intersection):

1) Determine the center of the 2nd box (blue);
2) Add the length of the 2nd box (blue)to the length of the 1st box (red);
3) Add the height of the 2nd box (blue)to the height of the 1st box (red);
4) Check whether the center of the 2nd box (blue) is inside the 1st augmented box (point-box membership)

**Algorithm 2** (point-box membership):

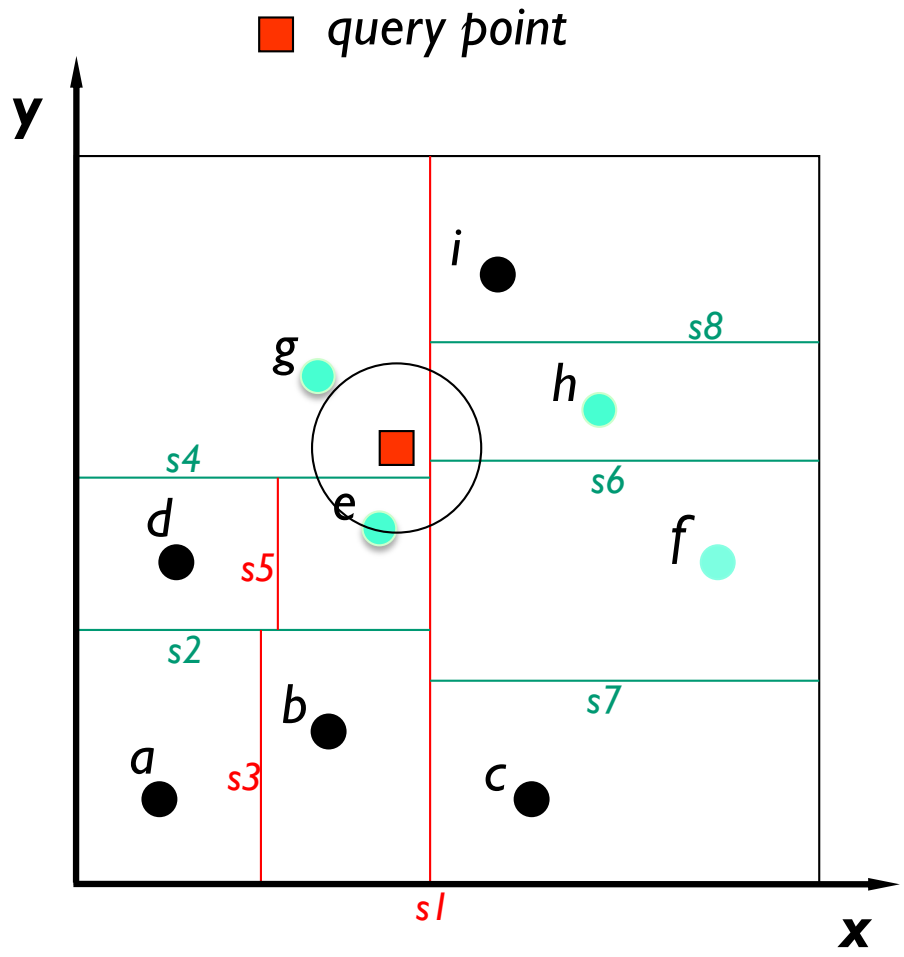
```
IN: point P
IN: left-bottom corner L
IN: right-top corner R
OUT: boolean value (either true or false)
1)  if
2)  (
3)      (p.x > L.x) and (p.x < R.x)
4)      and
5)      (p.y > L.y) and (p.y < R.y)
6)  )
7)       return TRUE;
8)  else
9)       return FALSE;
```

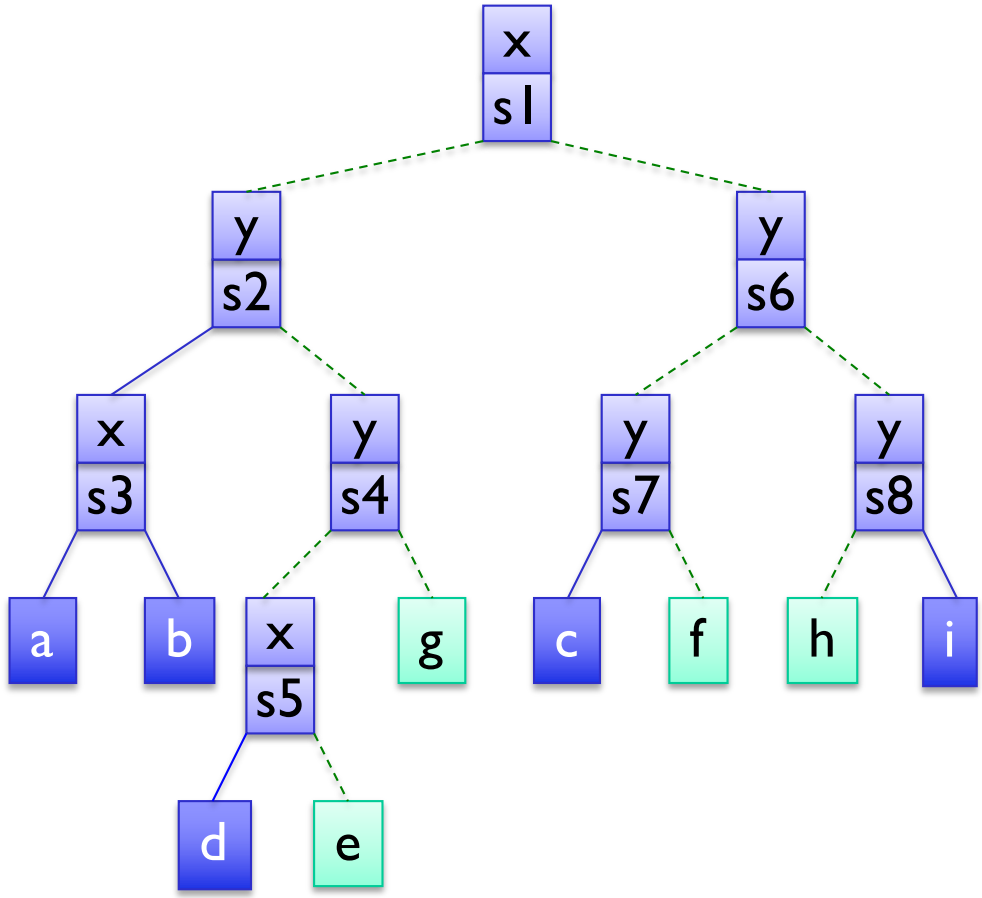


# K-d tree nearest neighbor search

Search recursively to find the point in the same cell as the query.

On the return search each subtree where a closer point than the one you already know about might be found.



# Further notes on k-d NNS

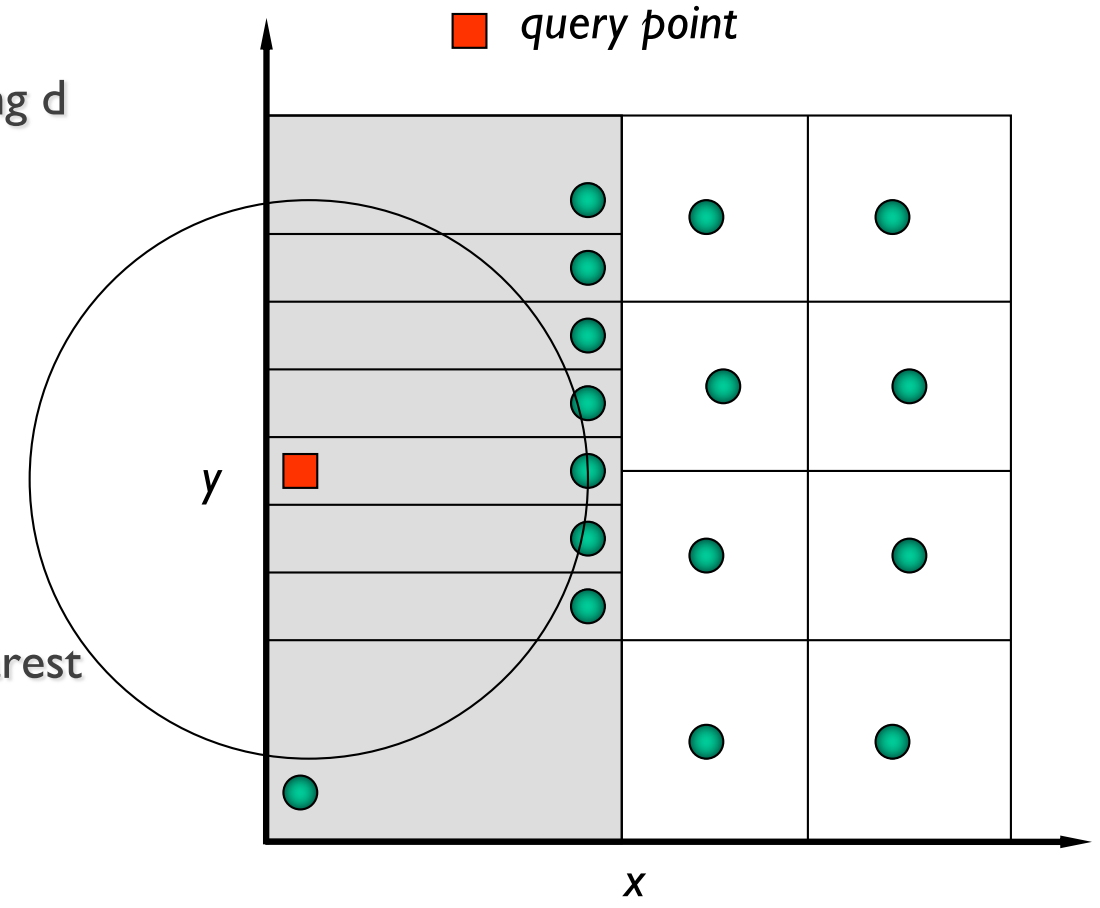
Has been shown to run in O(log n) average time per search in a reasonable model.

Storage for the k-d tree is O(n).

Preprocessing time is O(n log n) assuming d is a constant.

Worst-case for NNS:

- Half of the points visited for a query
- Worst case O(n)
- But: on average (and in practice) nearest neighbor queries are O(log N)





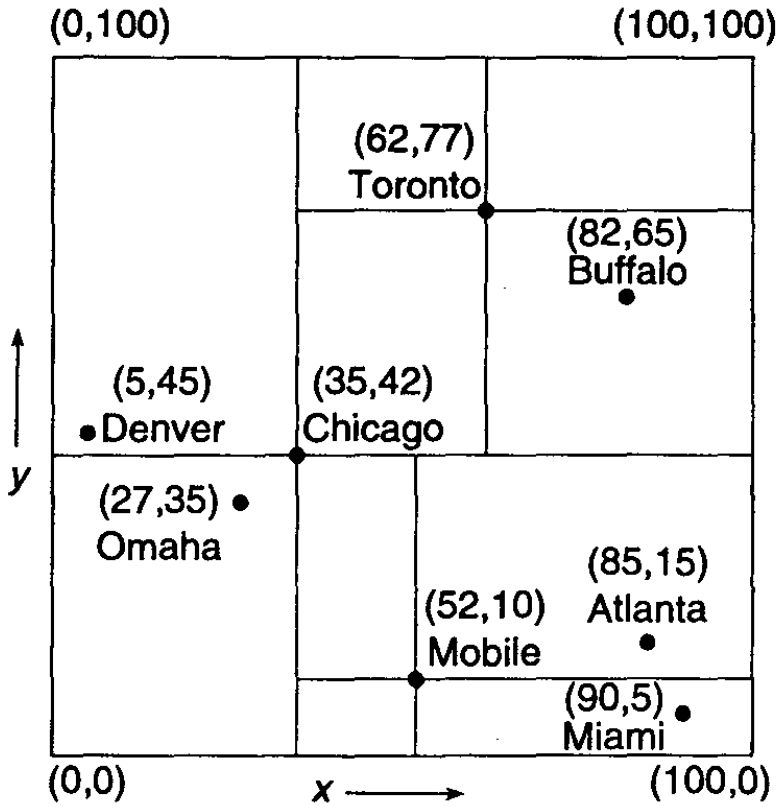
# Quad Trees

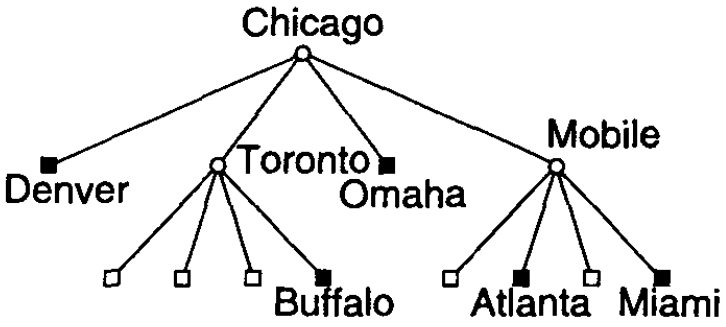
*Motivation for studying quadtrees*:

- Allows efficient querying of points in multidimensional space by *pruning* the search space.
- Applications:
  - photon mapping
  - point cloud processing

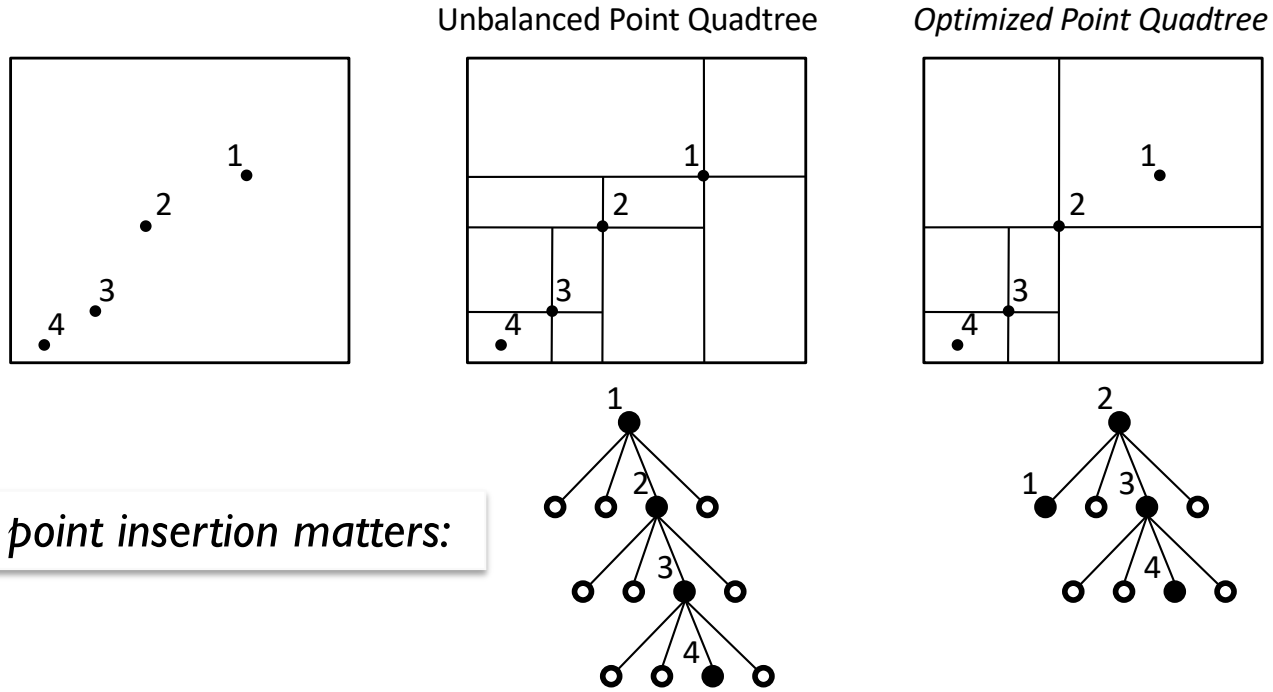
# Point quadtrees

*Spatial decomposition at data points.*



# Point insertion




*Order of data point insertion matters:*

An ***optimized point quadtree*** is constructed such that for any node x, the number of nodes in any of its quadrants will not exceed half the total number of nodes in the subtree rooted at x.

Procedure for constructing optimized point quadtrees:

(1) Sort points by x-value.

(2) Assign median point m as root of tree.

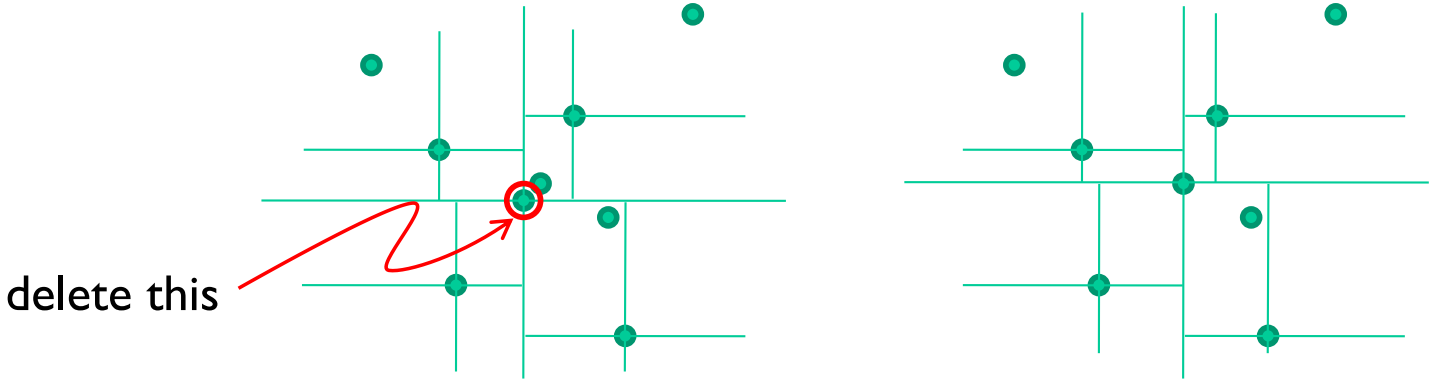
(3) By choosing m, remaining points get divided into 4 groups.

(4) Repeat procedure on each group.

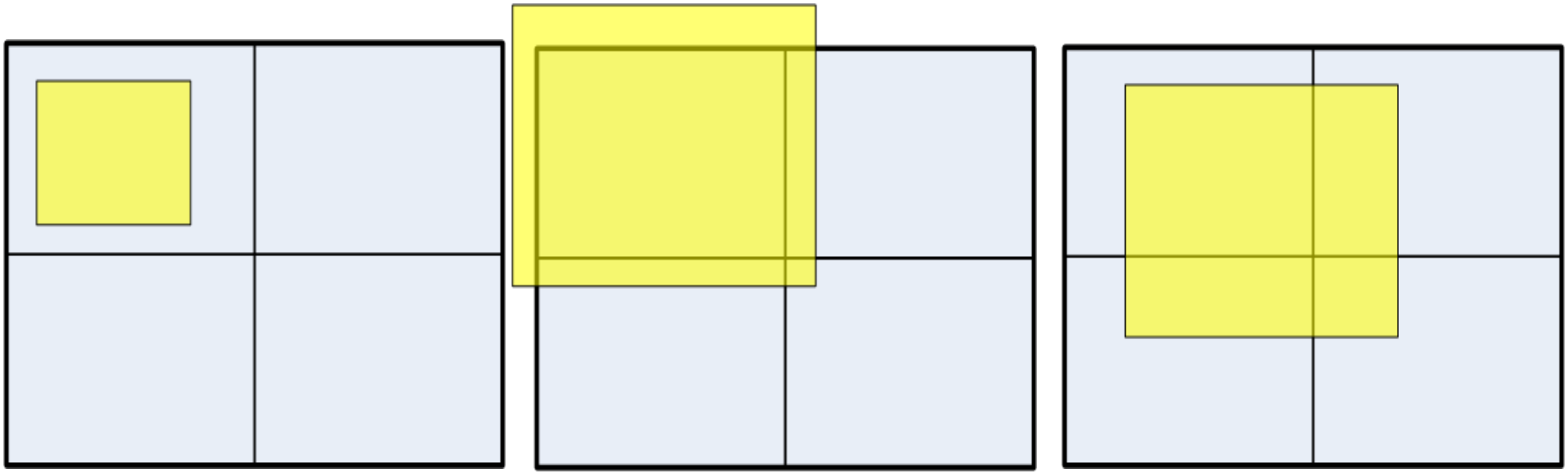
# Point deletion

***Naïve approach***: Remove node x and re-insert nodes in subtree(x). This is supposedly very inefficient.

***Smart approach***: The general idea is to pick a replacement node that minimizes the number of nodes requiring re-insertion. As illustrated below, in a good scenario, we only need to replace node x without having to restructure the tree.



# Rectangular range query (search)



To query a quadtree for items that are inside a particular rectangle, the tree is traversed. Each quad is tested for intersection with the query area.

1. Quads that do not intersect are not traversed, allowing large regions of the spatial index to be rejected rapidly.

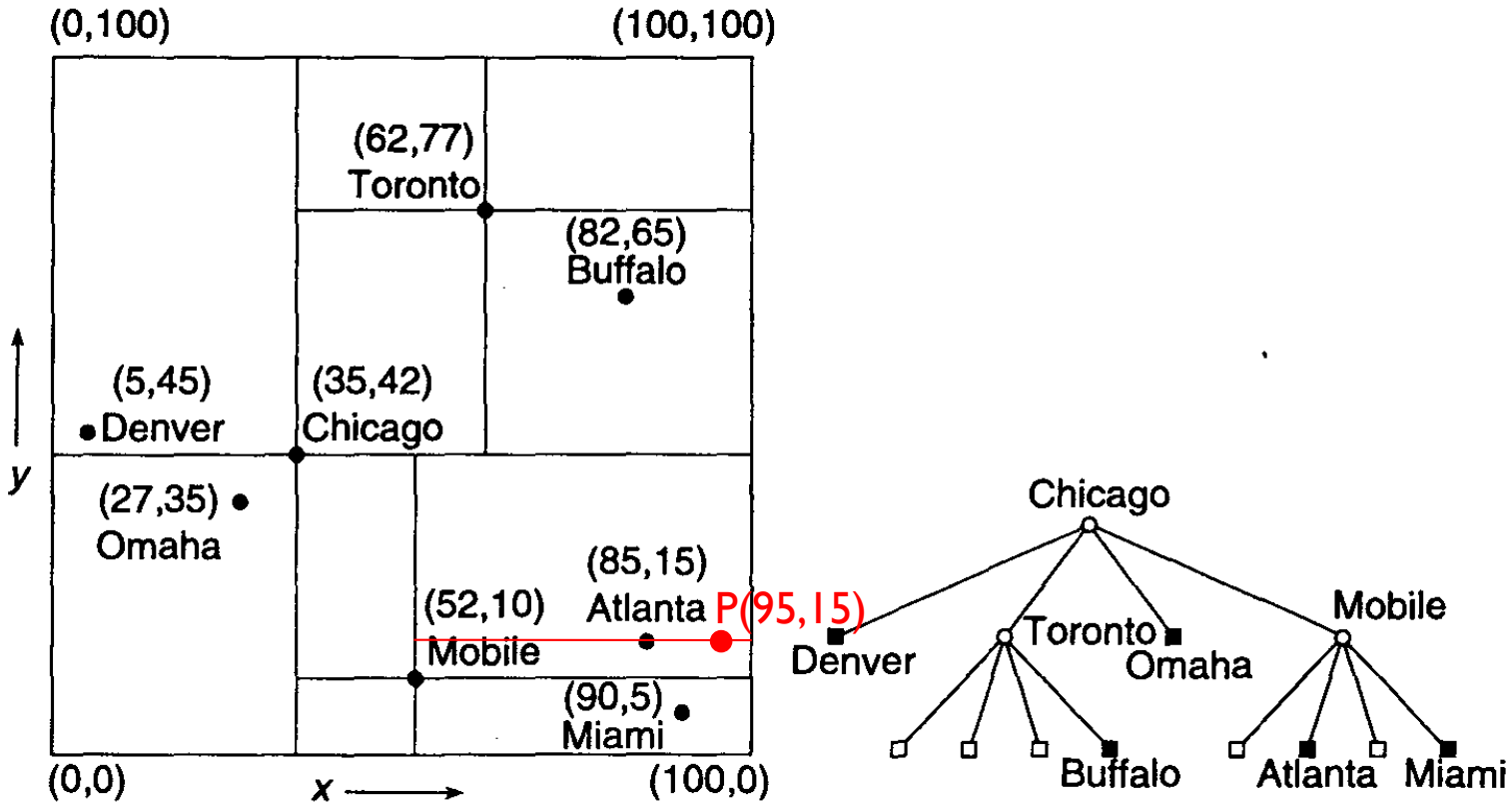
2. Quads that are wholly contained by the query region have their sub-trees added to the result set without further spatial tests: this allows large regions to be covered without further expensive operations.

3. Quads that intersect are traversed, with each sub-quad tested for intersection recursively.

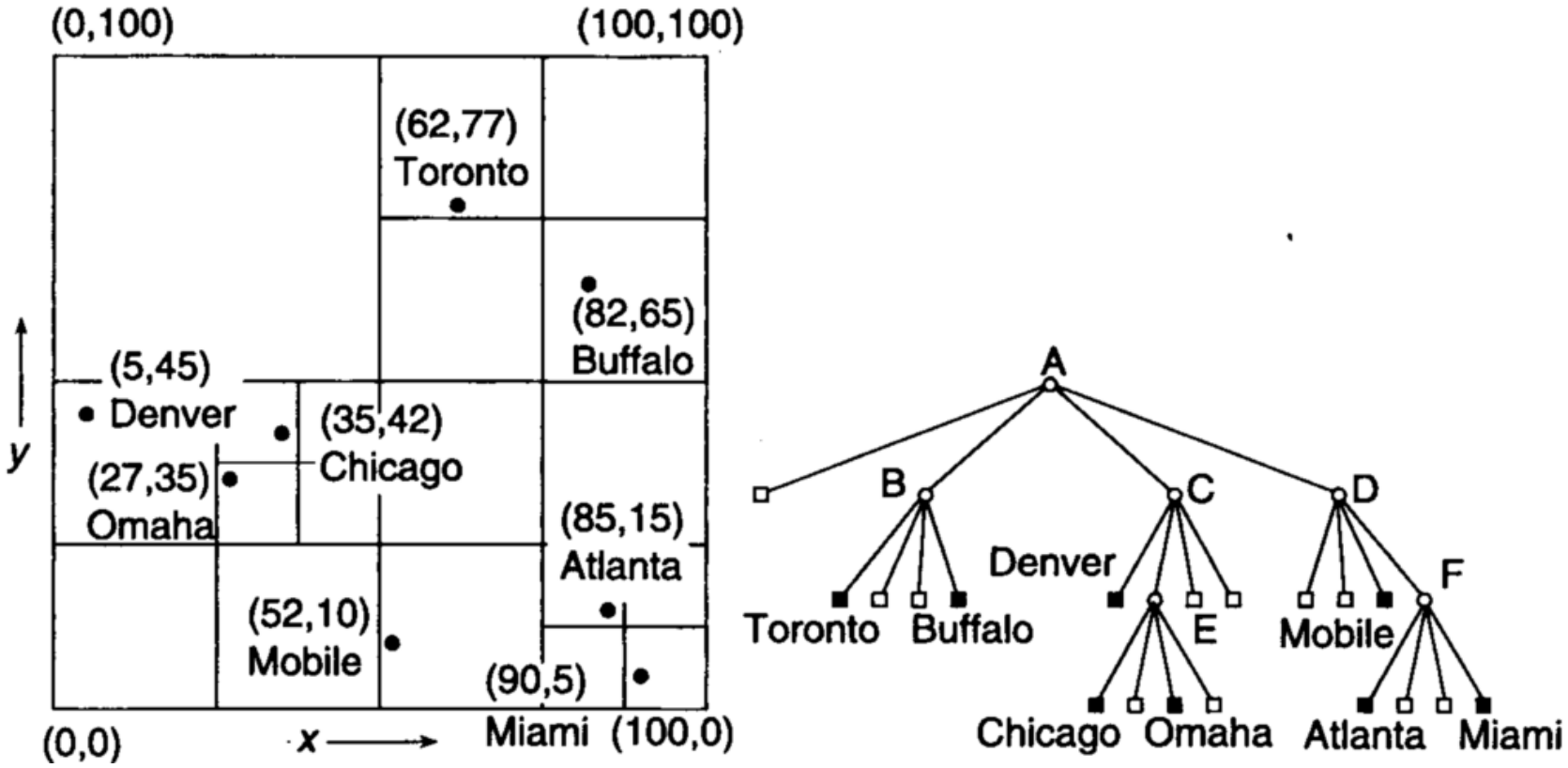
When a quad is found with no sub-quads, its contents are individually tested for intersection with the query rectangle.

# Nearest neighbor search

***Problem***:
Given a point quadtree T and a point P, find the node in T that is the closest to P.
***Idea***:
Traverse the quadtree maintaining a priority list, candidates, based on the distance from P to the *quadrants* containing the candidate nodes.



# PR (point region) quadtrees




Space subdivided repeatedly into congruent quadrants until all quadrants contain no more than one data point

The *bucket PR quadtree* allows no more than some b > 1 number of data points per quadrant.

# Quadtree comparison

## Point Quadtree

- Pros:
  - Compact, because number of tree nodes equals number of data points
  - Shorter search paths (~$\log_4 N$) compared to kd-trees (~$\log_2 N$)
- Cons:
  - Deletion of a node involves finding a suitable replacement and rearranging the nodes of its subtree.
  - Tree shape depends on order of data point insertion. Inserting at arbitrary order may result in unbalanced trees.

## PR Quadtree

- Pros:
  - Tree shape independent of order of data point insertion. It depends only on arrangement of data points in space
  - Deletion is straightforward, since all data points reside in leaf nodes.
- Cons:
  - Certain quadrants may require many subdivisions to separate densely clumped points, leading to a deep search paths.

# Relevant references on quadtrees

y's webpage: Contains a cool Java Applet showing how various data structures work.
http://donar.umiacs.umd.edu/quadtree/

D.T. Lee and C. K. Wong. *Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees*

Samet. *Deletion in Two-Dimensional Quad Trees*

Finkel and Bentley. *Quad Trees: A Data Structure for Retrieval on Composite Keys.*

http://en.wikipedia.org/wiki/Quadtree

http://www.xnawiki.com/index.php/QuadTree

*Video showing why spatial data structures are important in games:*

https://www.gdcvault.com/play/1017645/Physics-for-Game-Programmers-Spatial

# Summary:

...:

- Introduction.
- Taxonomy of spatial data structures.
- K-d trees: overview, construction, and complexity.
- K-d trees: rectangular range query, nearest neighbor search.
- Point quadtrees: overview, construction, and complexity.
- Point quadtrees: point insertion, point deletion, rectangular range query, nearest neighbor search.
- Region quadtrees.