CSE 326: Data Structures
Seeing the forest for the trees

Hannah Tang and Brian Tjaden
Summer Quarter 2002

Today's Outline - *k*d trees

> Too much light often blinds gentlemen of this sort,
> They cannot see the forest for the trees.
> - Christoph Martin Wieland

What's the goal for this course?

> It is not possible for one to teach others, until one can first teach herself - *Confucious*

Data Structures - what's in a name?

Shakespeare

- Stacks and Queues
- Priority Queues
  - Binary heap, Leftist heap, Skew heap, *d* - heap
- Trees
  - Binary search tree, AVL tree, Splay tree, B tree
- Hash Tables
  - Open and closed hashing, extendible, perfect, and universal hashing
- Disjoint Sets
- Graphs
  - Topological sort, shortest path algorithms, Dijkstra's algorithm, minimum spanning trees (Prim's algorithm and Kruskal's algorithm)

- Asymptotic analysis
- Sorting
  - Comparison based sorting, lower-bound on sorting, radix sorting
- World Wide Web

- Implement if you had to
- Understand trade-offs between various data structures/algorithms
- Know when to use and when not to use
- Real world applications

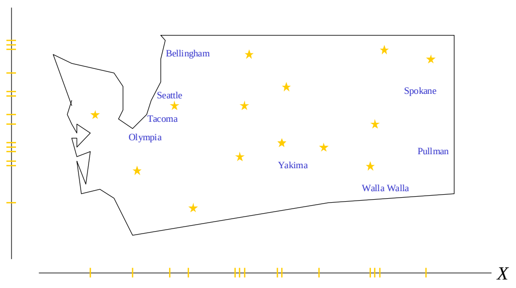
Range Query

A *range query* is a search in a dictionary in which the exact key may not be entirely specified.

Range queries are the primary interface with multi-D data structures.

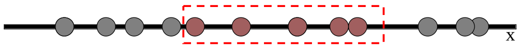
Remember Assignment #2? Give an algorithm that takes a binary search tree as input along with 2 keys, *x* and *y*, with $x \leq y$, and prints all keys *z* in the tree such that $x \leq z \leq y$.

Range Query Example



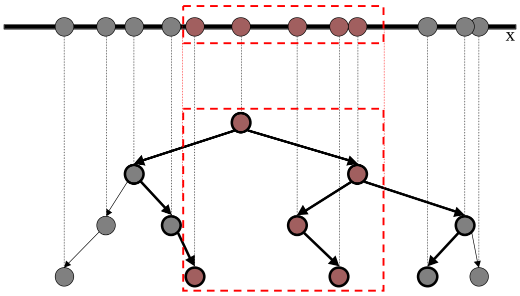
Range Querying in 1-D

Find everything in the rectangle…

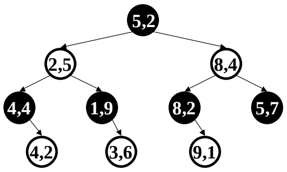


Range Querying in 1-D with a BST

Find everything in the rectangle…



Multi-Dimensional Search ADT



- Dictionary operations
  - find
  - insert
  - delete
  - range queries
- Each item has *k* keys for a *k*-dimensional search tree
- Searches can be performed on one, some, or all the keys or on ranges of the keys

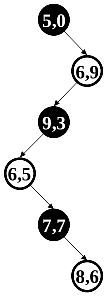
Applications of Multi-D Search

- Astronomy (simulation of galaxies) - 3 dimensions
- Protein folding in molecular biology - 3 dimensions
- Lossy data compression - 4 to 64 dimensions
- Image processing - 2 dimensions
- Graphics - 2 or 3 dimensions
- Animation - 3 to 4 dimensions
- Geographical databases - 2 or 3 dimensions
- Web searching - 200 or more dimensions

k-D Trees can be unbalanced

(but not when built in batch!)

insert(<5,0>)
insert(<6,9>)
insert(<9,3>)
insert(<6,5>)
insert(<7,7>)
insert(<8,6>)



height:

Find in a *k*-D Tree

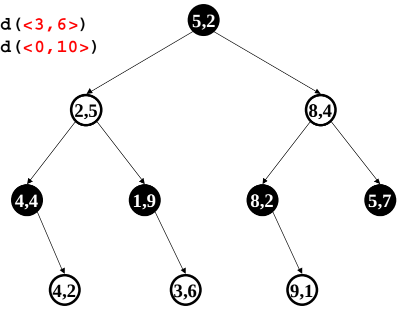
find(<x_1,x_2, …, x_k>, root) finds the node which has the given set of keys in it or returns **null** if there is no such node

```
Node find(keyVector keys, Node root) {
  int dim = root.getDimension();
  if (root == NULL)
    return root;
  else if (root.getKeys() == keys)
    return root;
  else if (keys[dim] < (root.getKeys())[dim])
    return find(keys, root.getLeft());
  else
    return find(keys, root.getRight());
}
```

runtime:

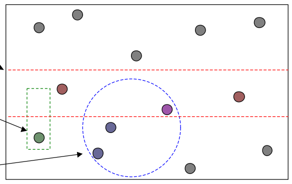
Find Example

`find(<3,6>)`
`find(<0,10>)`



Range Query Examples: Two Dimensions

- Search for items based on *just one key*
- Search for items based on *ranges for all keys*
- Search for items based on a function of several keys: e.g., a *circular range query*

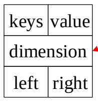


---

k-D Trees

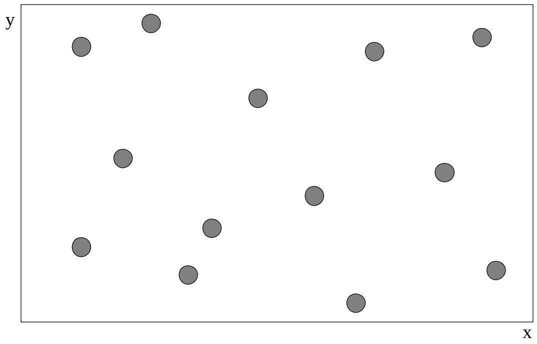
- Split on the next dimension at each succeeding level
- If building in batch, choose the median along the current dimension at each level
  - guarantees logarithmic height and balanced tree
- In general, add as in a BST

k-D tree node

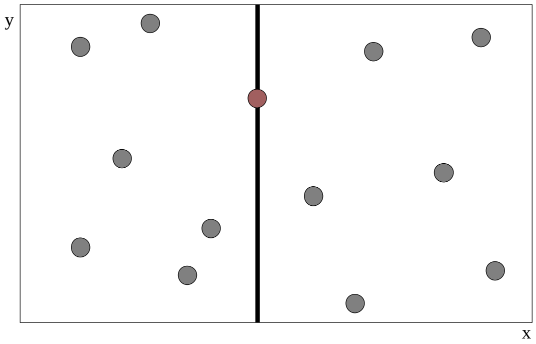


The dimension that this node splits on

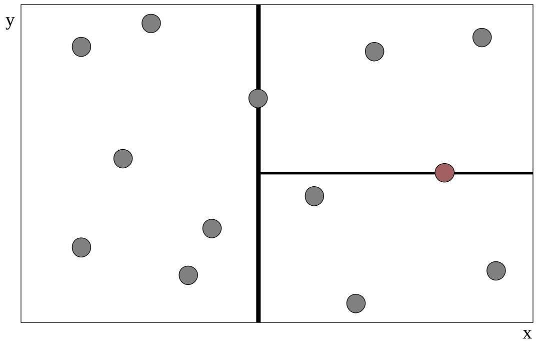
Building a 2-D Tree (0/4)



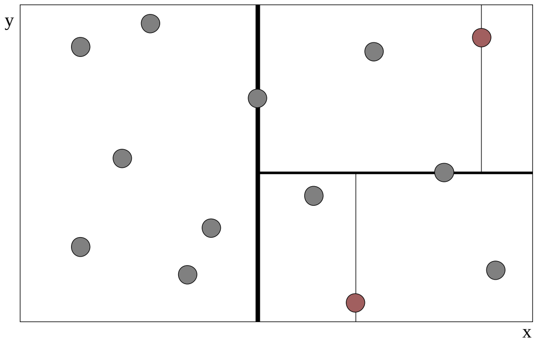
Building a 2-D Tree (1/4)



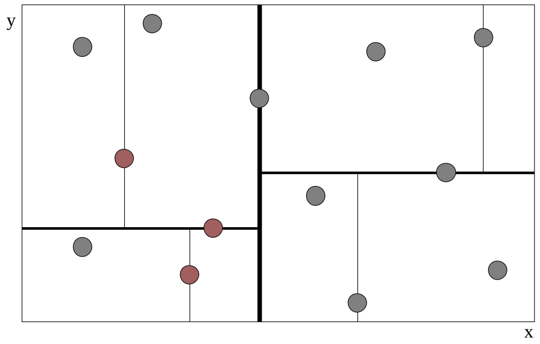
Building a 2-D Tree (2/4)



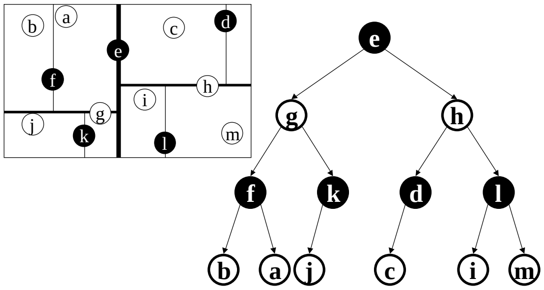
Building a 2-D Tree (3/4)



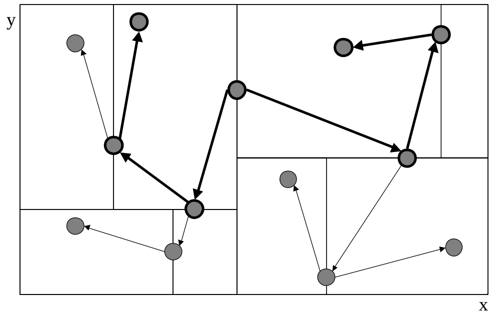
Building a 2-D Tree (4/4)



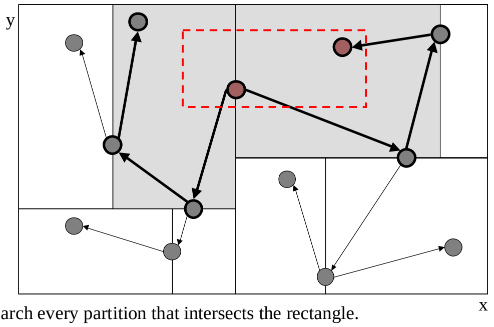
k-D Tree



2-D Tree

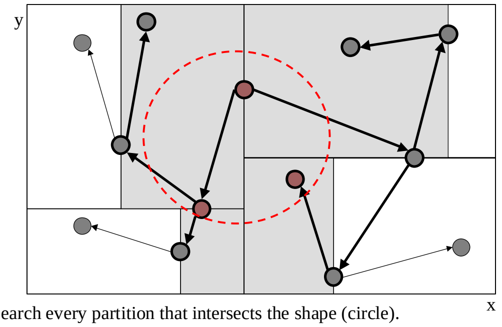


2-D Range Querying in 2-D Trees



Search every partition that intersects the rectangle.
Check whether each node (including leaves) falls into the range.

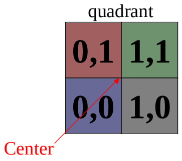
Other Shapes for Range Querying



Search every partition that intersects the shape (circle).
Check whether each node (including leaves) falls into the shape.

Quad Trees

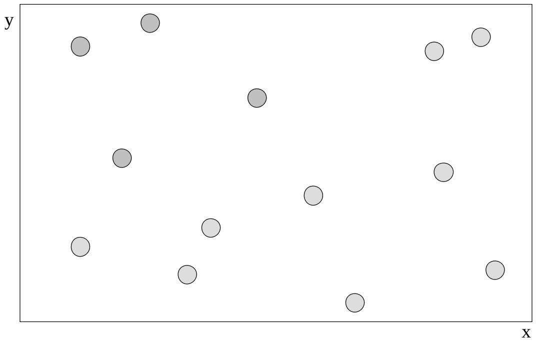
- Split on *all* (two) dimensions at each level
- Split key space into equal size partitions (quadrants)
- Add a new node by adding to a leaf, and, if the leaf is already occupied, split until only one node per leaf



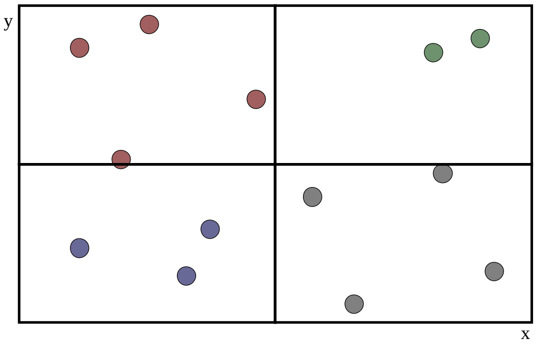
quad tree node

	keys		value	
Center:	x		y	
Quadrants:	0,0	1,0	0,1	1,1

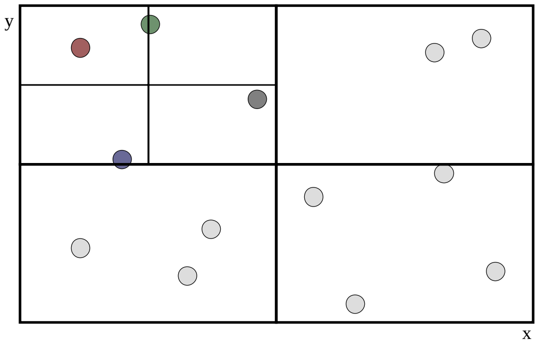
Building a Quad Tree (0/5)



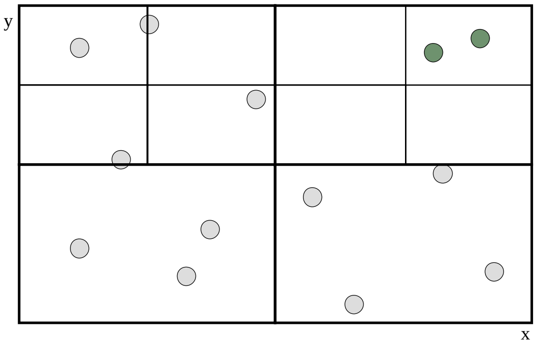
Building a Quad Tree (1/5)



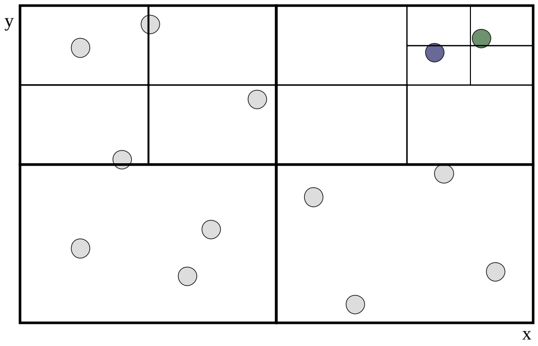
Building a Quad Tree (2/5)



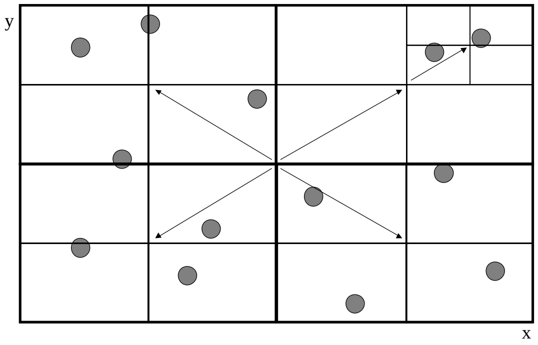
Building a Quad Tree (3/5)



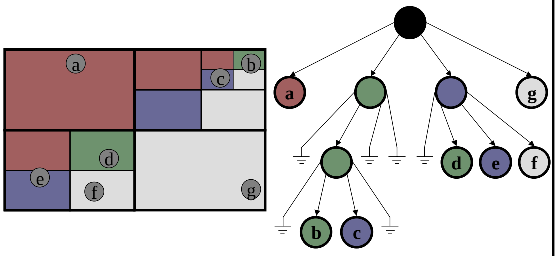
Building a Quad Tree (4/5)



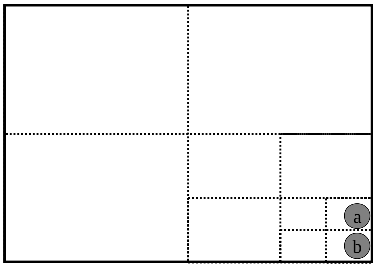
Building a Quad Tree (5/5)



Quad Tree Example



Quad Trees can be unbalanced



height:

Quad Trees vs. *k*-D Trees

- *k*-D Trees
  - Density balanced trees
  - Number of nodes is O(n) where *n* is the number of points
  - Height of the tree is O(log n) *with batch insertion*
  - Supports insert, find, nearest neighbor, range queries
- Quad Trees
  - Number of nodes is O(n(1+ log(Δ/n))) where *n* is the number of points and Δ is the ratio of the width (or height) of the key space and the smallest distance between two points
  - Height of the tree is O(log n + log Δ)
  - Supports insert, delete, find, nearest neighbor, range queries