CSCI 4560/6560 Computational Geometry

https://www.cs.rpi.edu/~cutler/classes/computationalgeometry/F23/

Lecture 8: Orthogonal Range Searching

Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

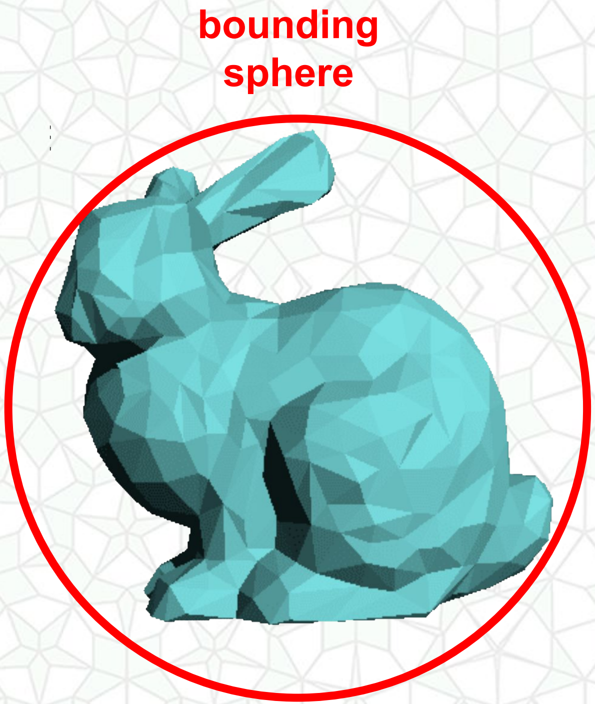
Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Problem: Minimal Bounding ~~Sphere~~ Circle

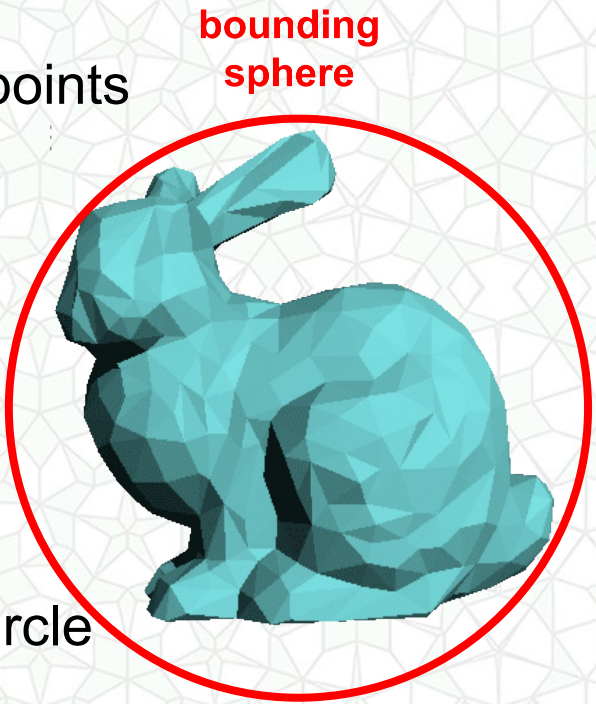
- Input: *n* vertices in ~~3D~~ 2D
- Assume (for convenience):
  "General Position"
  - No 3 points are collinear
  - No 4 points lie on the same circle
- Output: 3 of those vertices uniquely define a circle such that all other points lie inside of that circle

Note: In 3D, we would output 4 vertices (4 vertices uniquely define a sphere)



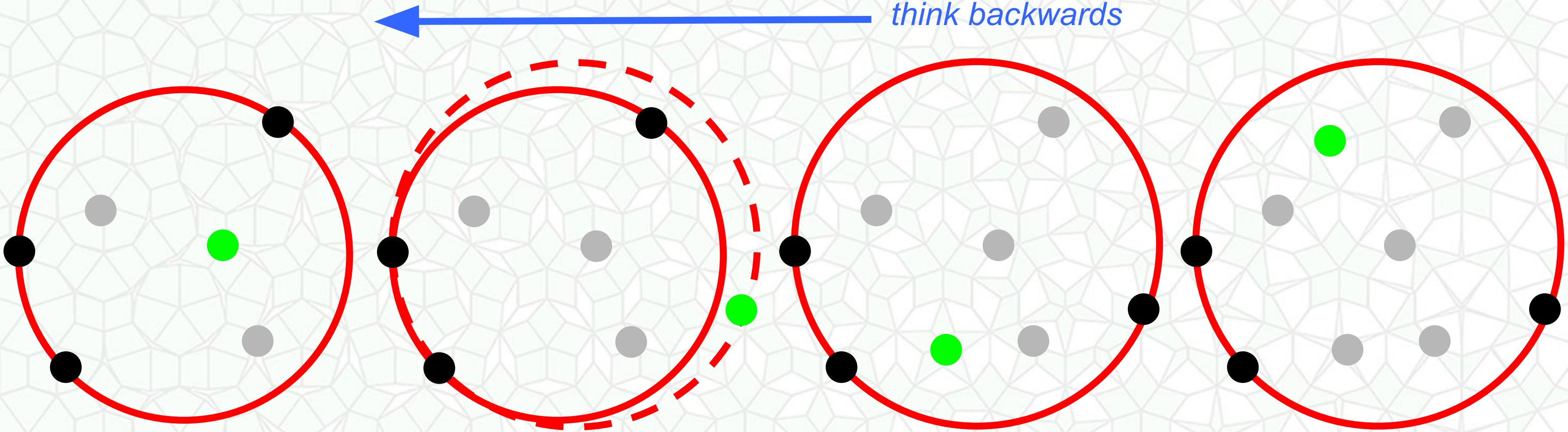
Problem: Minimal Bounding ~~Sphere~~ Circle

- Brute Force: $O(n^4)$
  - Try ALL triples, check against all other points
- Best Case: $O(n)$
  - Fit circle to first 3 points
  - Check all other points
  - Be lucky!
- Worst Case: $O(n^3)$
  - Fit circle to first 3 points
  - Unfortunately, find a point outside the circle
  - ***Try again… but we know that point MUST be on the solution circle***



Randomized Incremental Construction

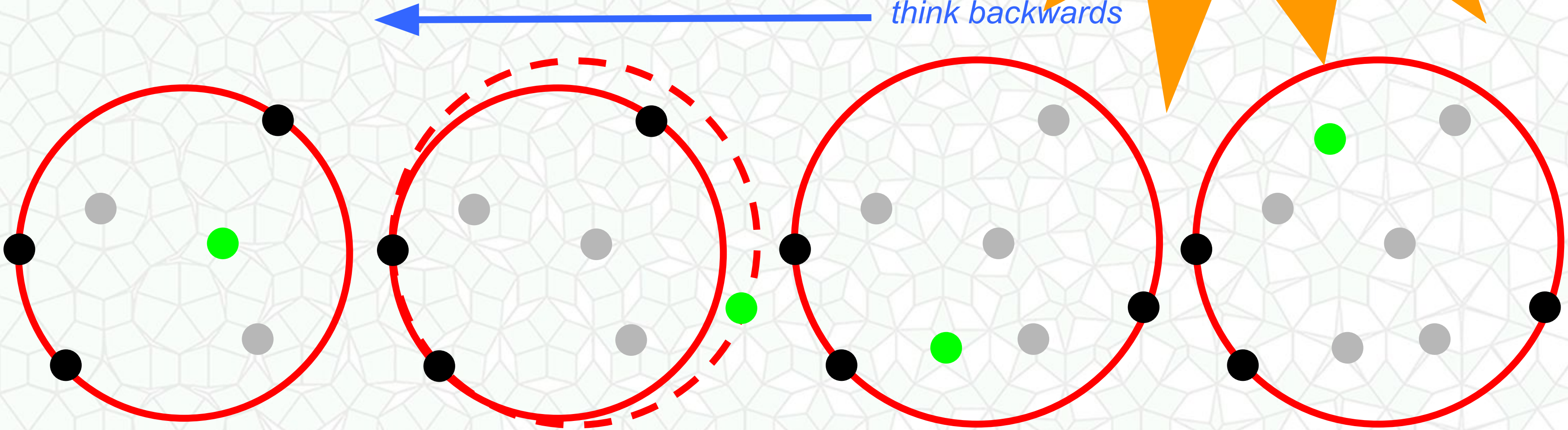
- We start with all *n* points and the optimal minimal bounding circle, which is defined by 3 of those points.
- Each step, we randomly choose one of *n* points to remove.



Randomized Incremental Construction

- We start with all *n* points and the optimal mini
  which is defined by 3 of those points.
- Each step, we randomly choose one of *n* po

The probability that the removed point defined the circle is 3/n each step. We expect to recompute the circle O(1) times.

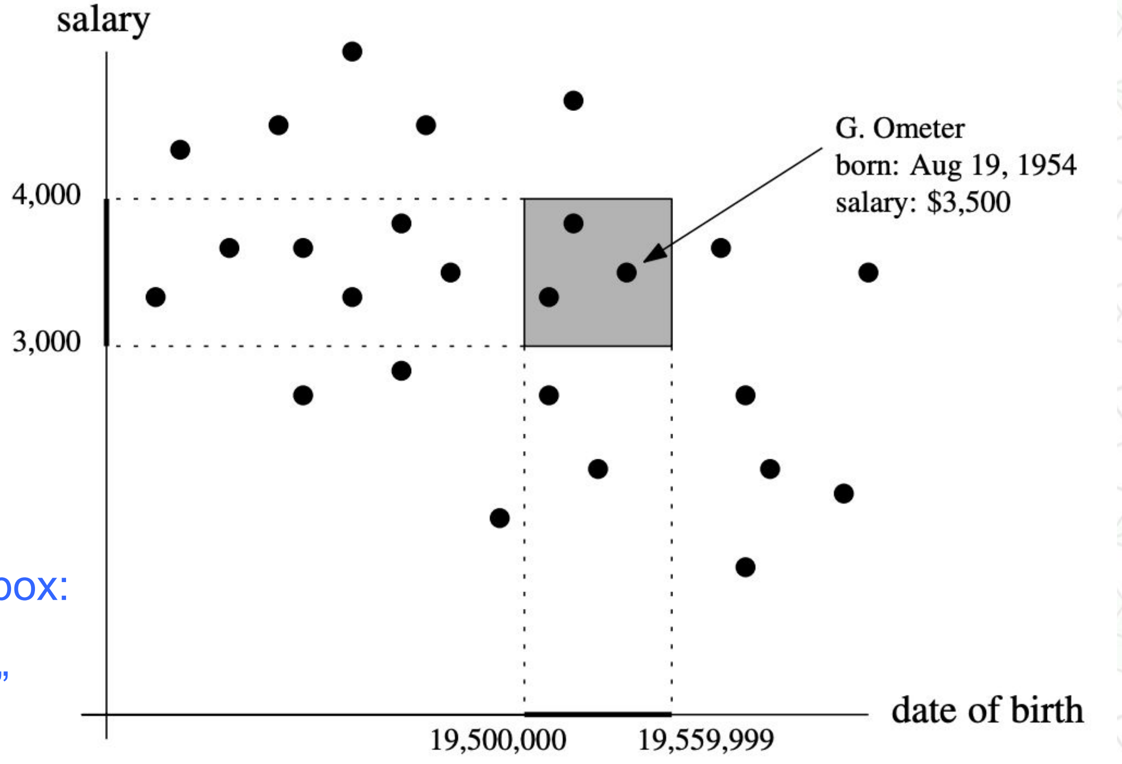


Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Motivating Application: 2D Database Queries

- Return all data points with *x* value in range $[x_0, x_1]$ and *y* value in range $[y_0, y_1]$



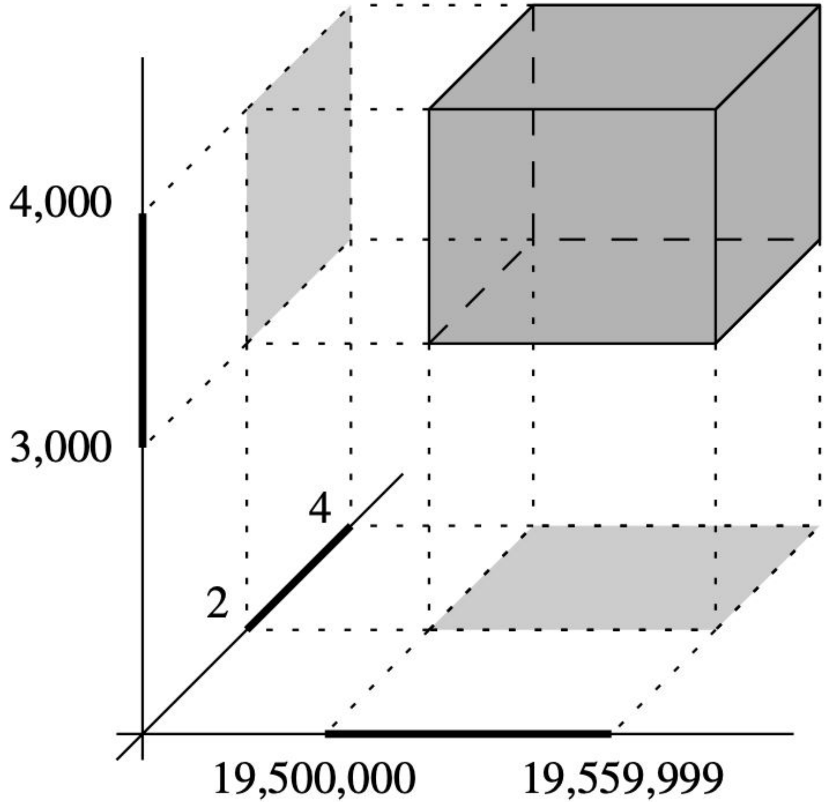
Find all values in an axis parallel box:
a "*rectangular range query*"
a.k.a. "*orthogonal range query*"

Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

Higher Dimensional Database Queries

- Return all data points with x value in range $[x_0, x_1]$ and y value in range $[y_0, y_1]$ and z value in range $[z_0, z_1]$ and …

Select all people born 1950-1960, with salary 3,000-4,000, who have 2-4 children



Find all values in an axis parallel box:
a "*rectangular range query*"
a.k.a. "*orthogonal range query*"

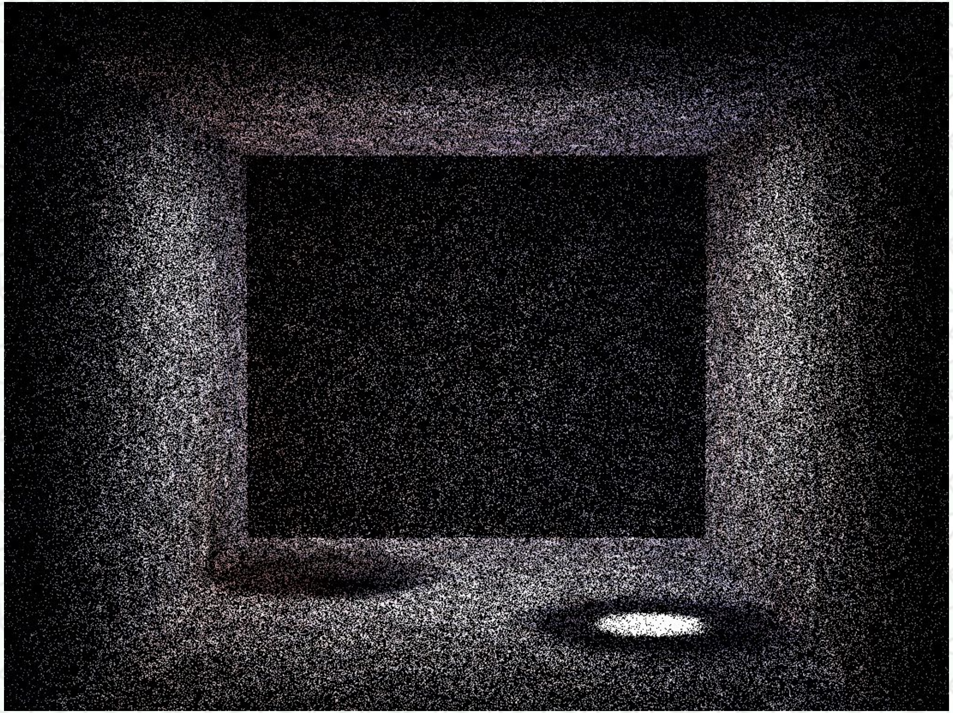
Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 5

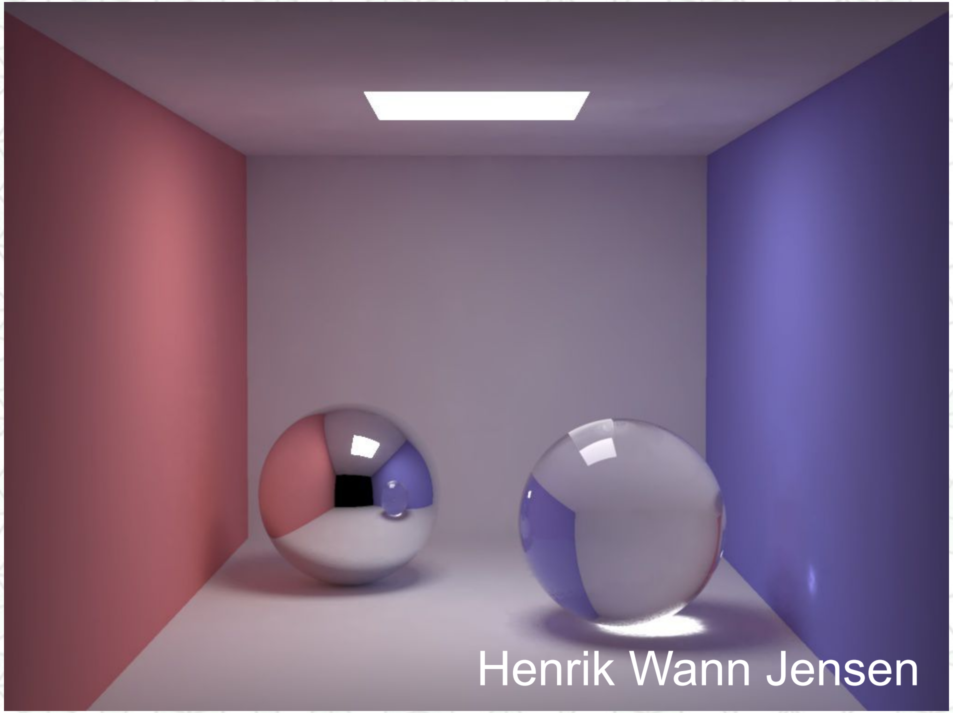
Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Motivating Application: Photon Mapping

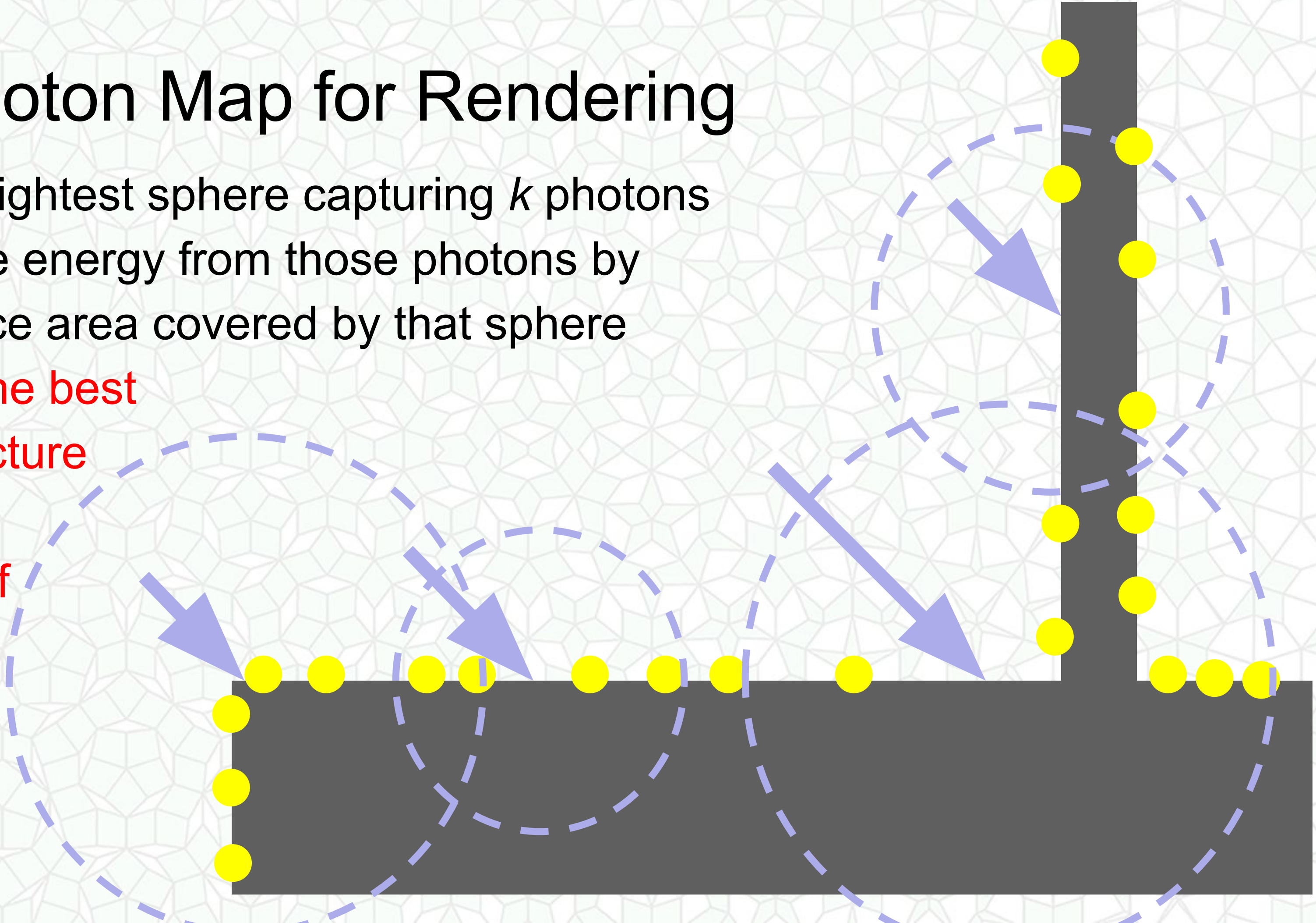
- Photons bounce around room and stored on each surface they hit





Using Photon Map for Rendering

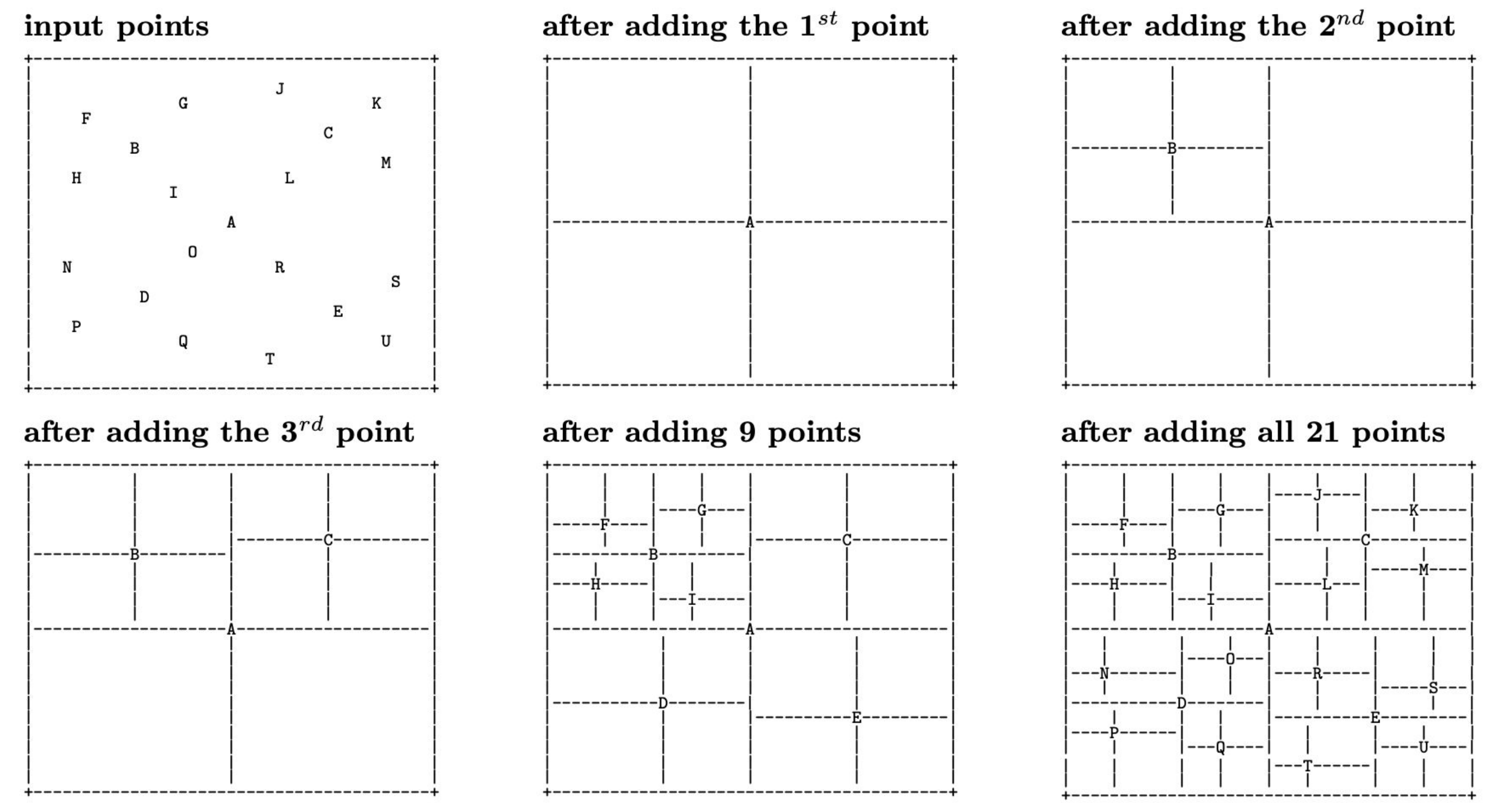
- Find the tightest sphere capturing *k* photons
- Divide the energy from those photons by the surface area covered by that sphere
- What is the best data structure to store millions of photons?



Outline for Today

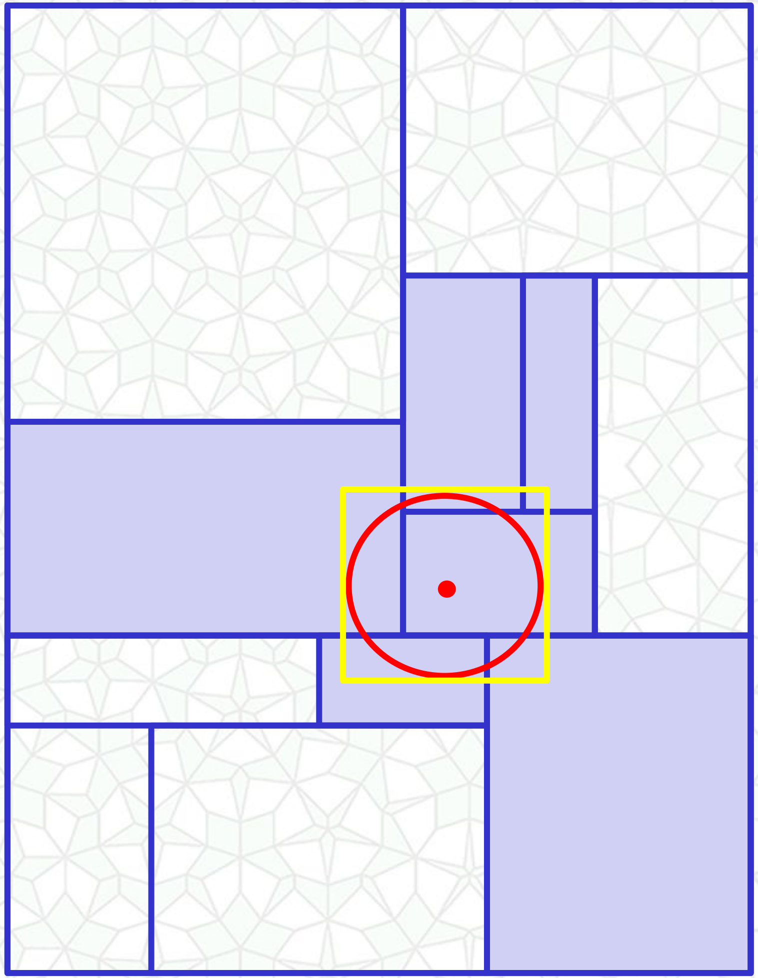
- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Data Structures Homework 8: Quad Tree



Collecting Photons from a *k*d tree

- Query point, and initial guess for radius (red)
- Make a rectangular/orthogonal query to the kD tree (yellow)
- kD tree returns all cells that overlap with query box (blue)
- Further processing necessary to filter points inside red circle and find smallest circle capturing exactly *k* photons

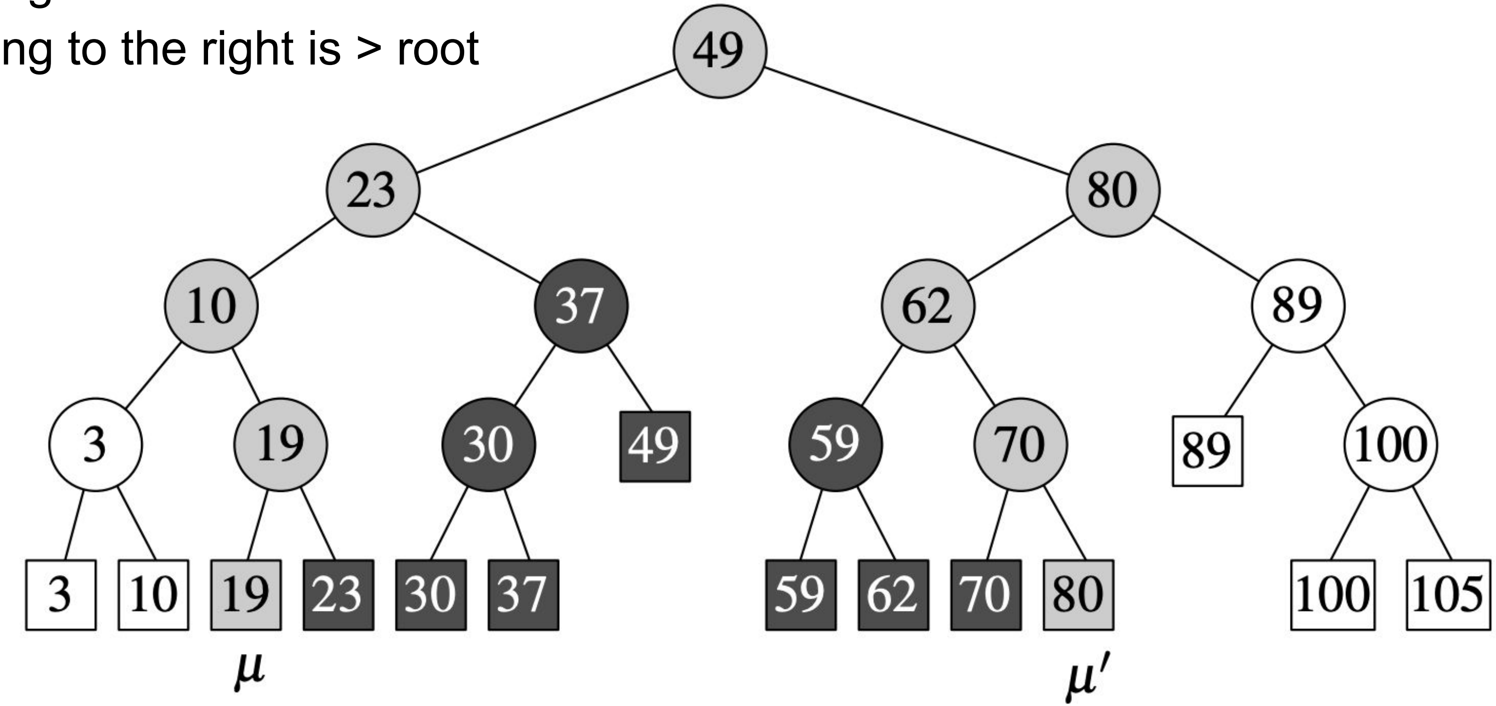


Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
    - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Review: 1 Dimensional Binary Search Trees

- Everything to the left is ≤ root
- Everything to the right is > root



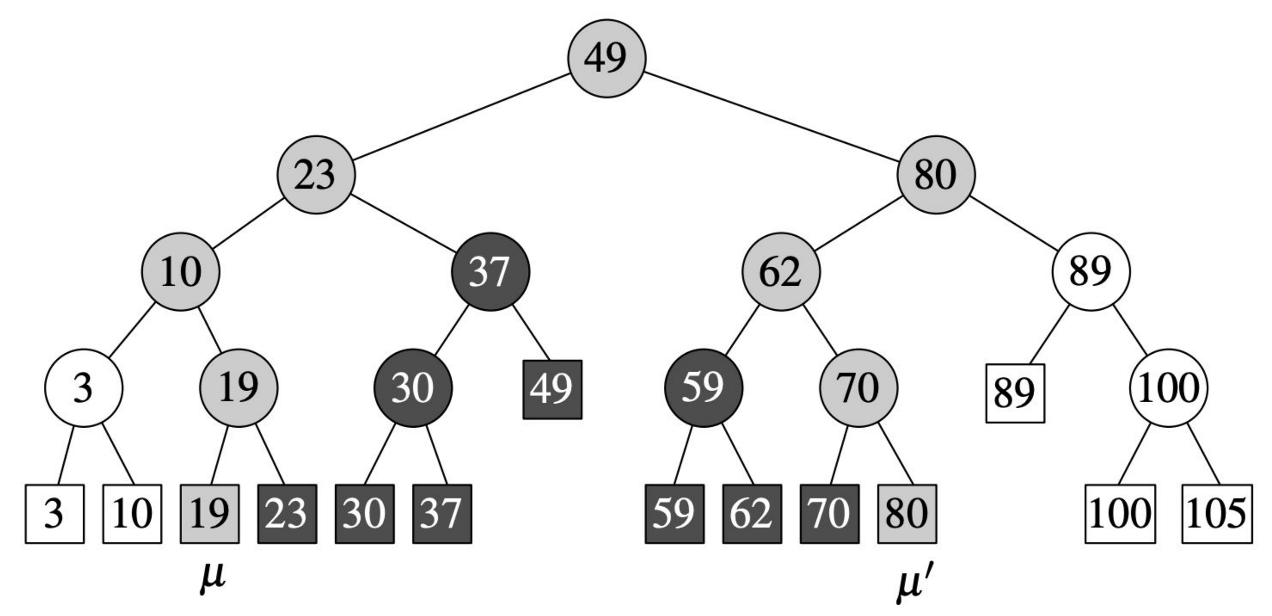
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

Assumptions

- No 2 data points have the same value in any dimension
  Only for algorithm presentation & analysis convenience,
  there are straightforward workarounds…

- We are given all of the data points at the start,
  allowing us to sort the data and construct well-balanced trees

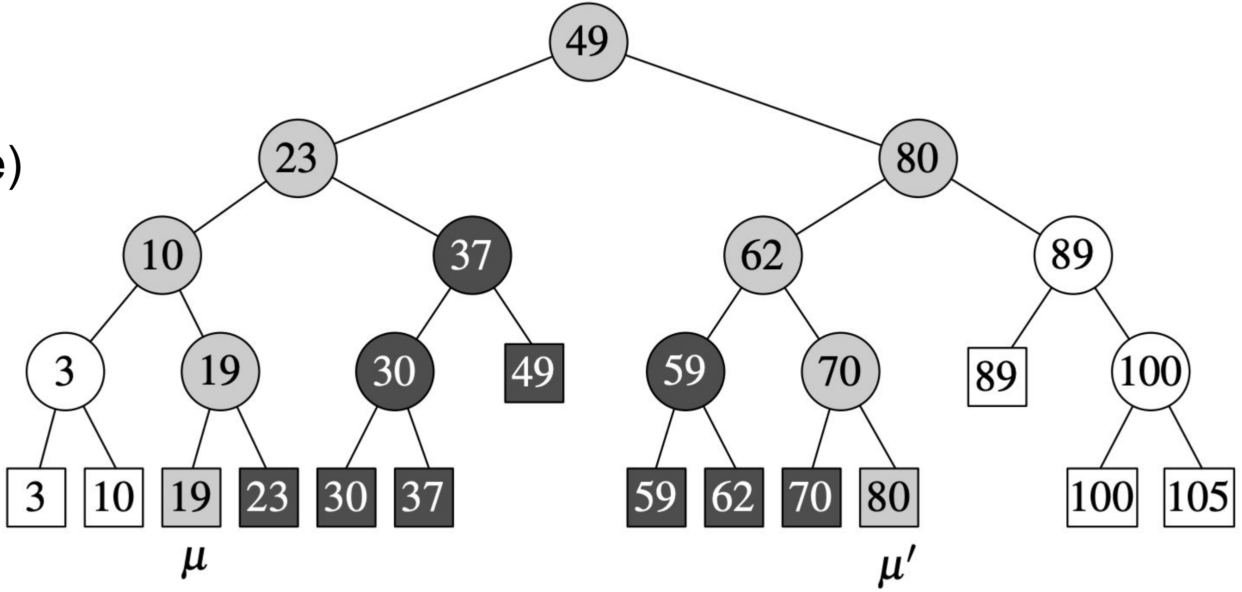
1D BST Construction Algorithm



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

1D BST Construction Algorithm

- Sort the data by x value
- Put the median (middle) value at the root
- Create 2 sublists for left & right
- Recurse

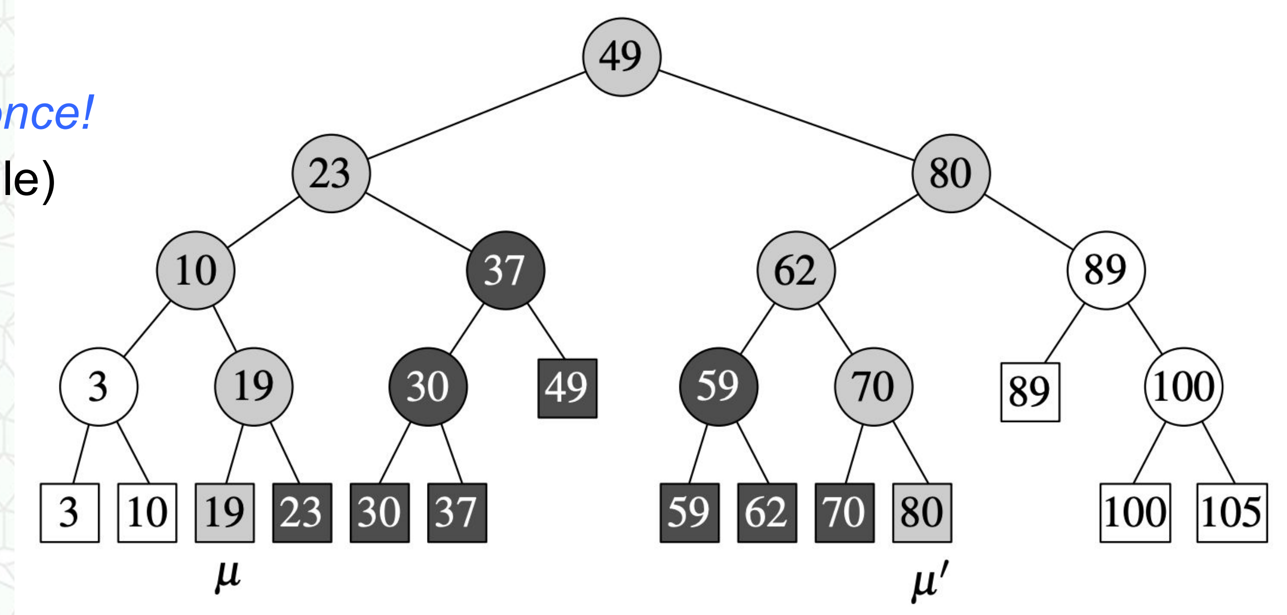


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

1D BST Construction Algorithm

- Sort the data by x value
  → *O(n log n)*
  only need to do this once!
- Put the median (middle) value at the root
  → *O(1)*
- Create 2 sublists for left & right
  → *O(n) for copy*
- Recurse

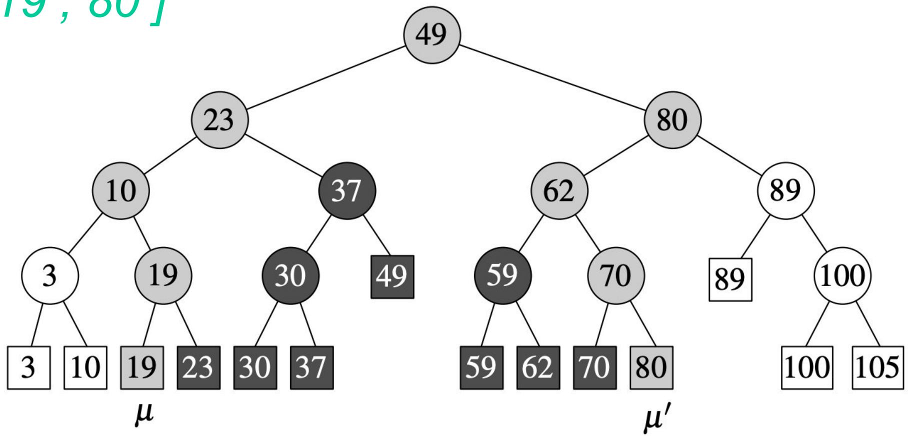
→ Overall *O(n log n)*



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

1D BST Query Algorithm

- Given a desired range [μ , μ']   *e.g., [19 , 80]*



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

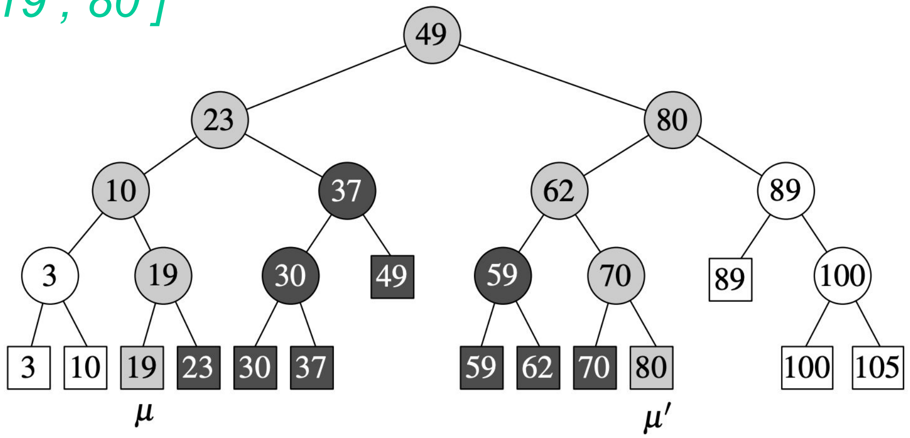
1D BST Query Algorithm

n values in the structure
k is the # of elements returned
an *output-sensitive* algorithm

- Given a desired range [μ , μ']   *e.g., [19 , 80]*

- Locate the leaf storing μ
- Locate the leaf storing μ'
- Increment from $\mu \rightarrow \mu'$
    - `Operator++`

    - `Operator++` from $\mu \rightarrow \mu'$



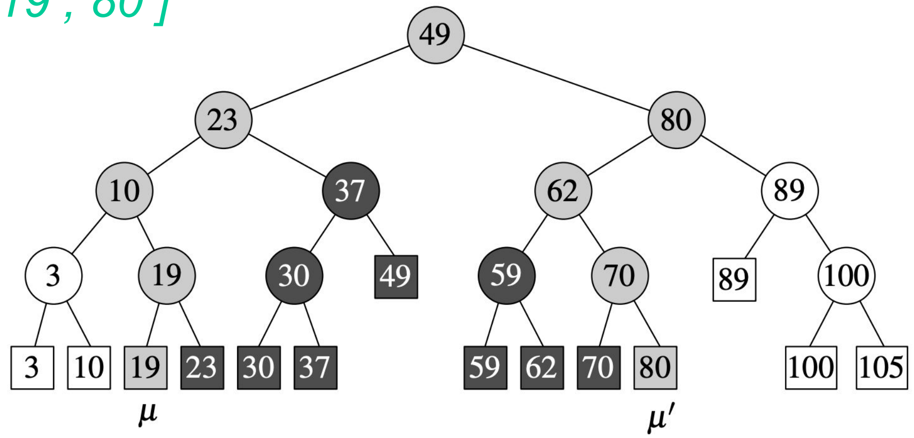
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

1D BST Query Algorithm

n values in the structure
k is the # of elements returned
an *output-sensitive* algorithm

- Given a desired range [μ , μ']   *e.g., [19 , 80]*

- Locate the leaf storing μ → O(log n)
- Locate the leaf storing μ' → O(log n)
- Increment from $\mu \rightarrow \mu'$
  - `Operator++`
    → *O(1)* expected time
  - `Operator++` from $\mu \rightarrow \mu'$
    → *k * O(1) = O(k) expected*



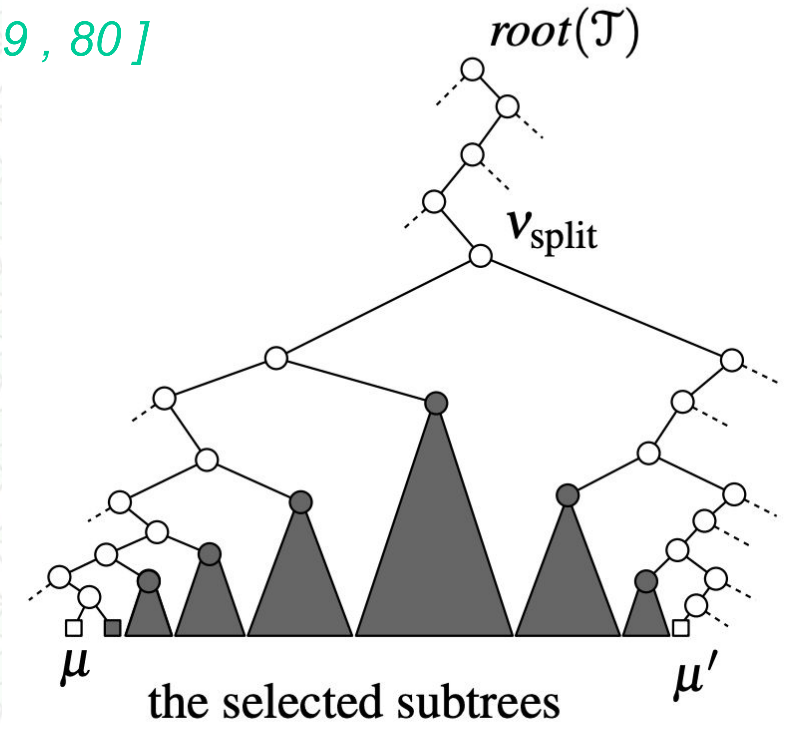
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

1D BST Query Algorithm

n values in the structure
k is the # of elements returned
an *output-sensitive* algorithm

- Given a desired range [μ , μ'] *e.g., [19 , 80]*

- Locate the leaf storing μ → O(log n)
- Locate the leaf storing μ' → O(log n)
- Increment from $\mu \rightarrow \mu'$
  - `Operator++`
    → *O(1)* expected time
  - `Operator++` from $\mu \rightarrow \mu'$
    → *k * O(1) = O(k) expected*
- *Equivalently:* Find all subtrees between the leaves, return all values in those subtrees



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

Analysis: 1D Binary Search Tree

n values in the structure
k is the # of elements returned
an *output-sensitive* algorithm

Starting with *n* values..

- Memory to store:
  - # of leaf nodes:
  - # of intermediate nodes:
  - Height of tree:
- Time to construct:
  - Sort the data:
  - Place middle value at root, recurse on left & right sublists:
- Time to query:
  - For search target / output returning *k* values

Analysis: 1D Binary Search Tree

n values in the structure
k is the # of elements returned
an *output-sensitive* algorithm

Starting with *n* values..

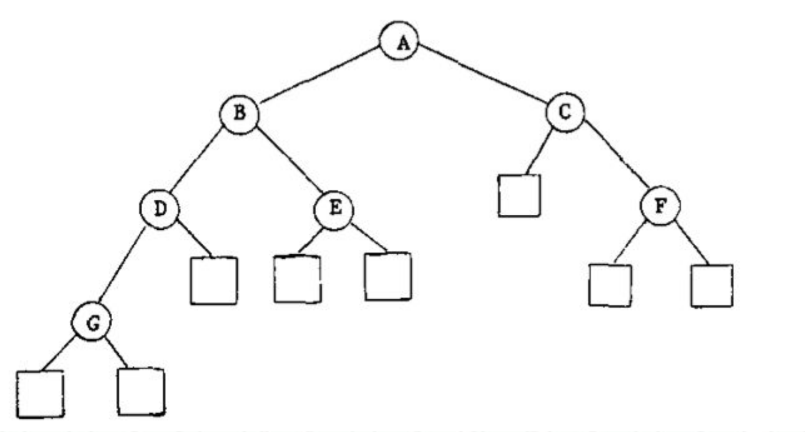
- Memory to store: → $O(n)$
  - # of leaf nodes: n
  - # of intermediate nodes: $n-1$
  - Height of tree: $\log n$
- Time to construct: → $O(n \log n)$
  - Sort the data: $O(n \log n)$
  - Place middle value at root, recurse on left & right sublists: $O(n)$
- Time to query: → $O(\log n + k)$
  - For search target / output returning *k* values

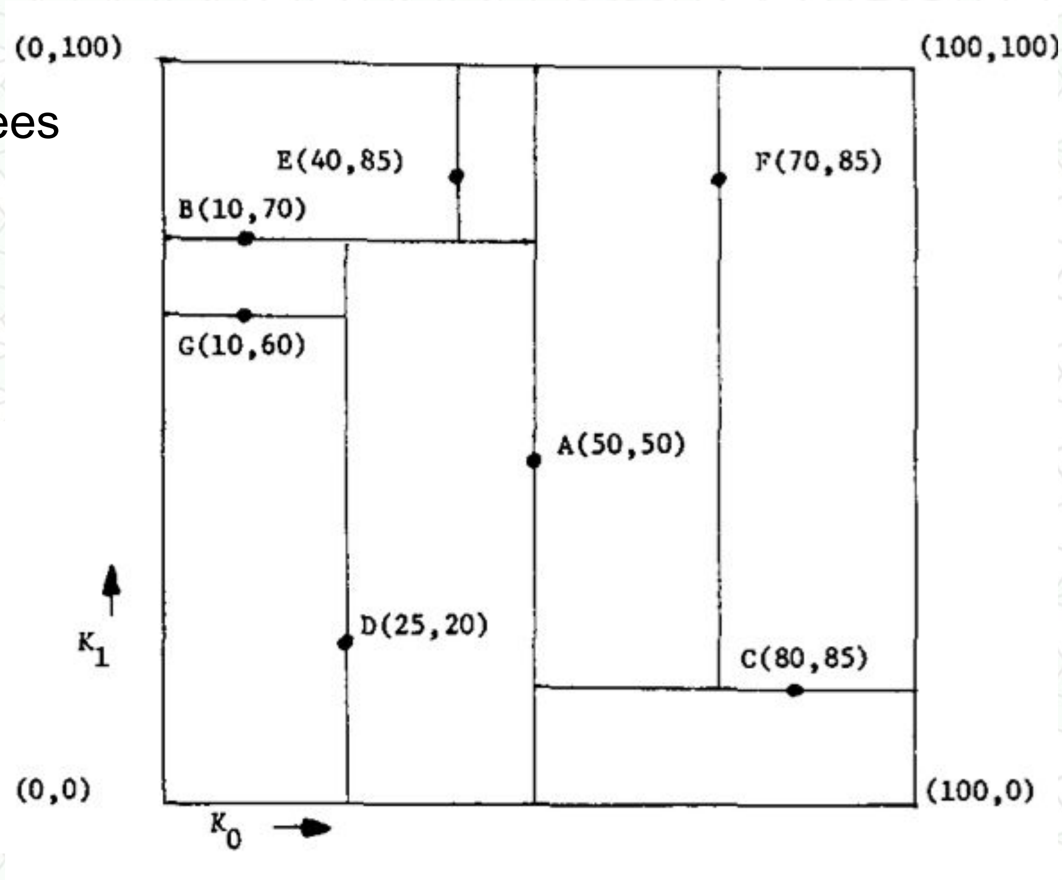
Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
    - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

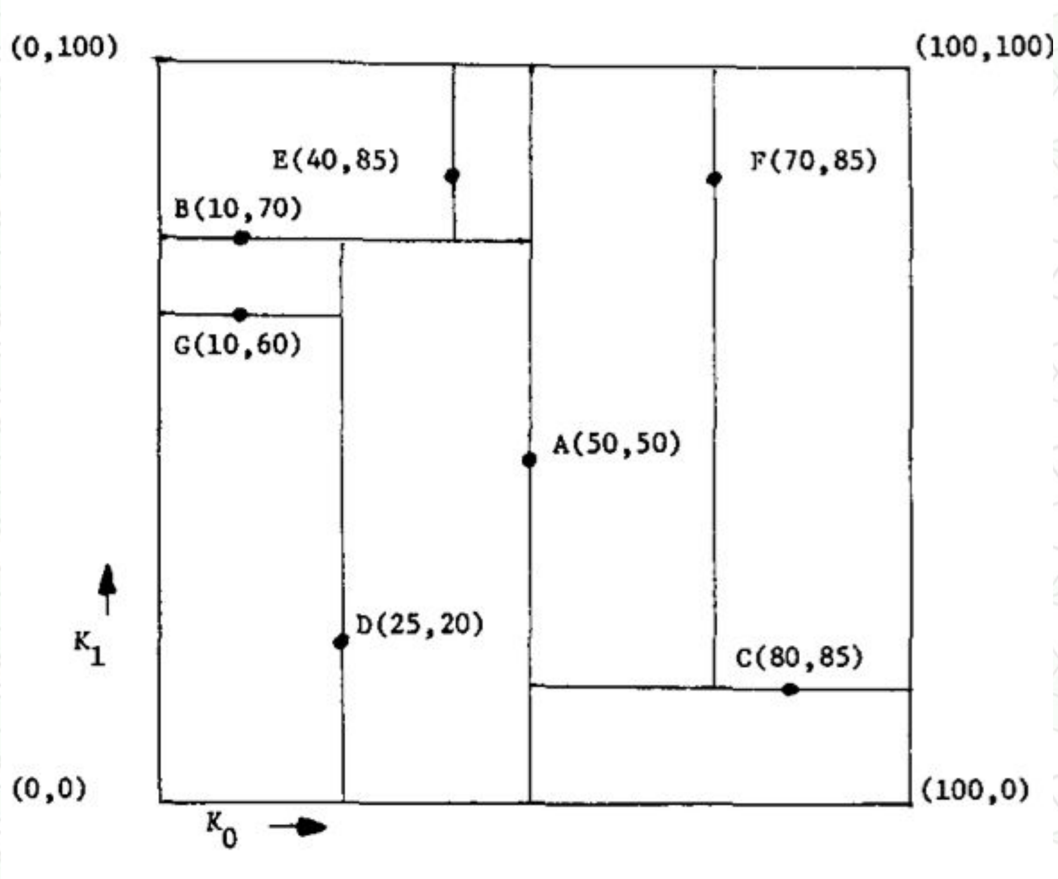
What is a *k*-d Tree?

"Multidimensional Binary Search Trees Used for Associative Searching", Communications of the ACM, Bentley 1975





2D kd Tree Construction

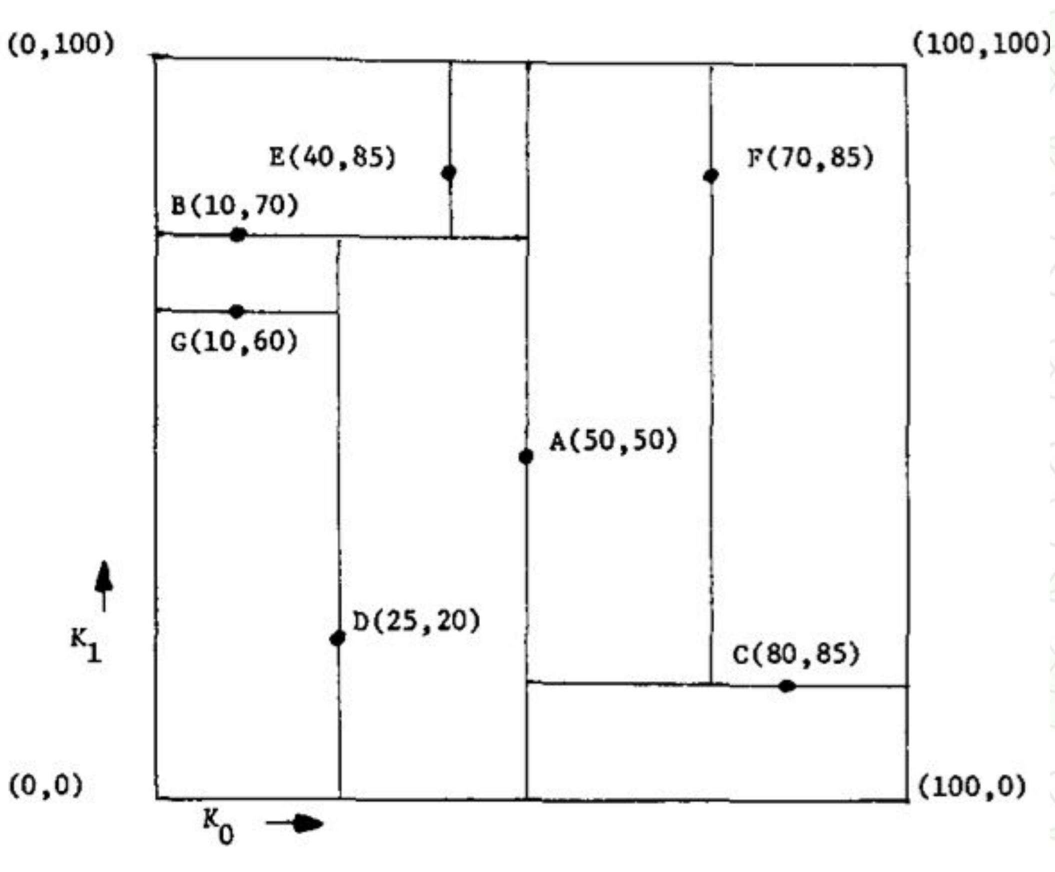


2D kd Tree Construction

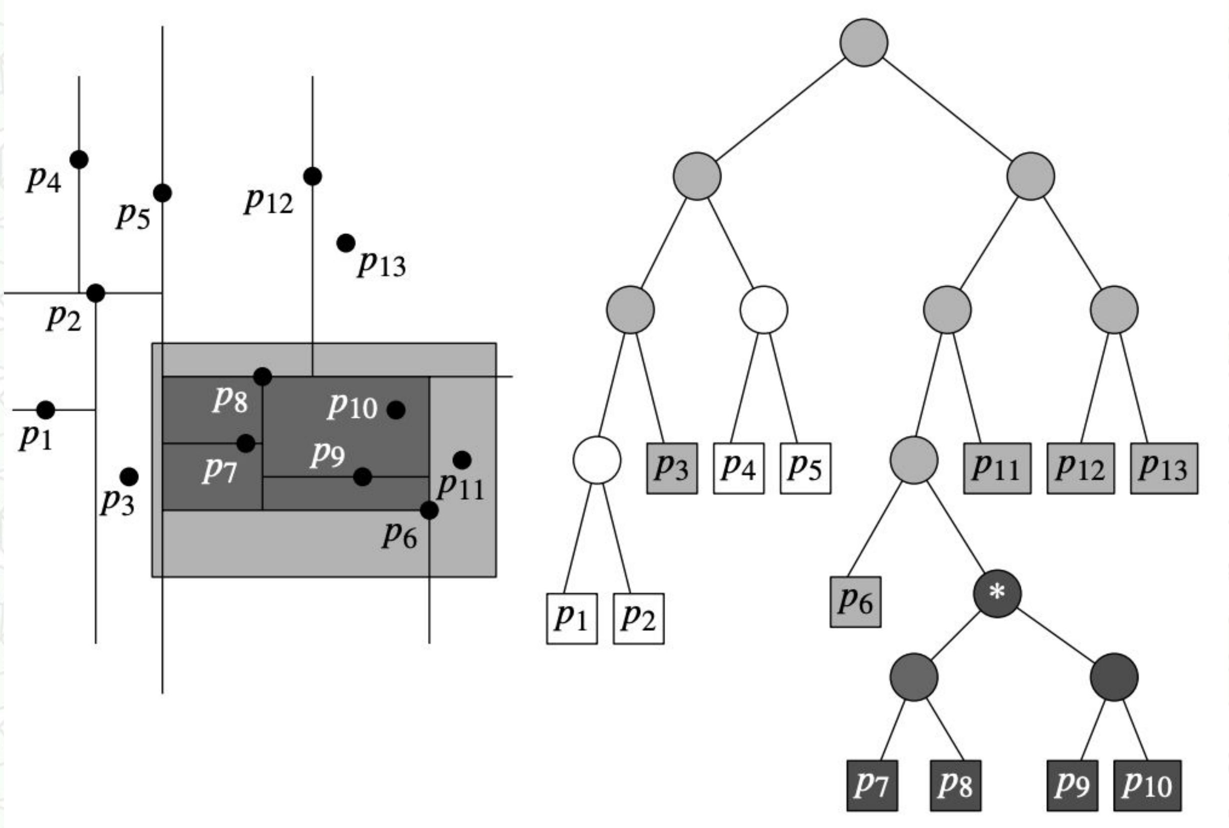
- Make 2 sorted lists,
  by *x* value and by y value

- Alternate dimensions
  (first split by x then by y)

- Find the median value
- Make a copy of the sorted lists, omitting values from the other side
- Recurse

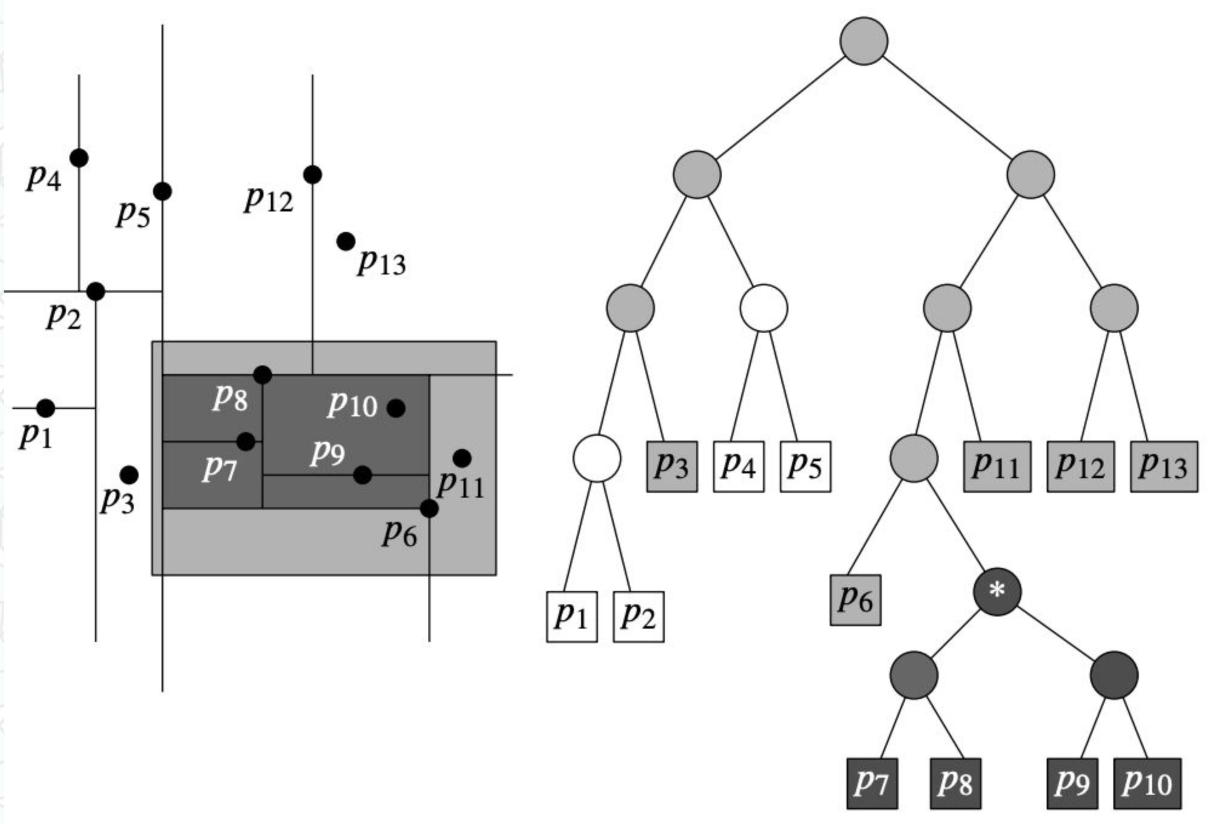


2D kd Tree Query Algorithm



2D kd Tree Query Algorithm

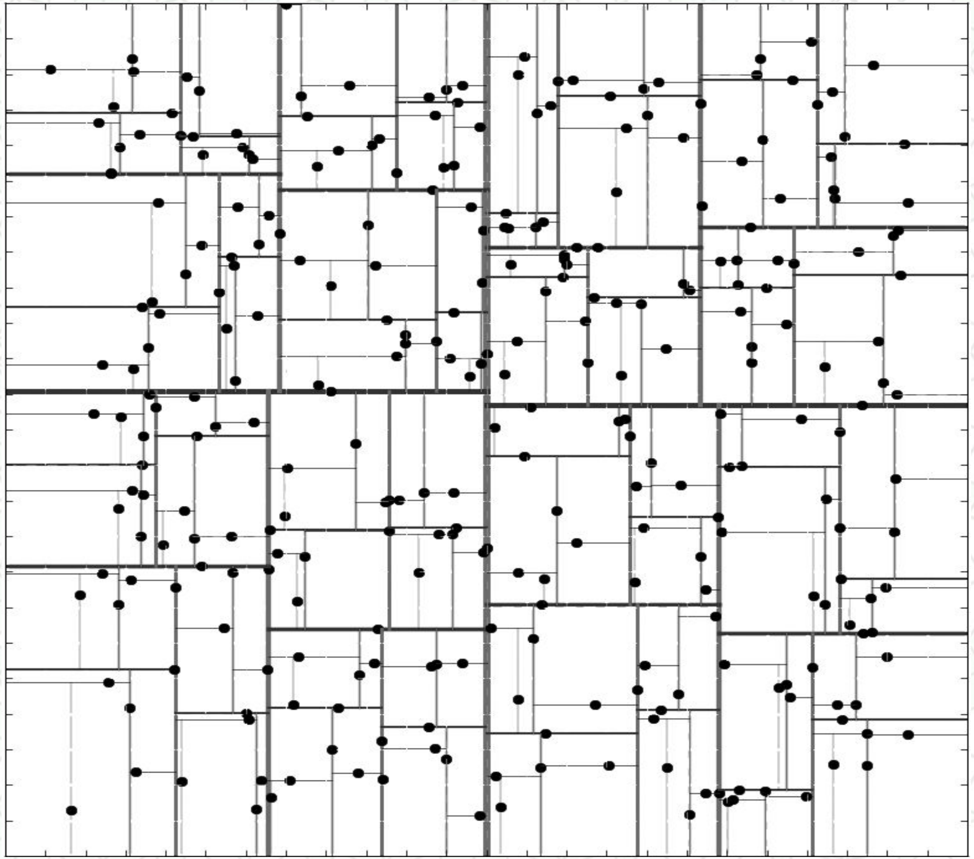
- At each split point
- Determine if the query box overlaps the split line
- Recurse down one or both branches
- If a subtree lies completely inside the box, return all items in that subtree
- Perform filtering in the leaves as necessary



2D kd Tree Query Analysis

https://salzis.wordpress.com/2014/06/28/kd-tree-and-nearest-neighbor-nn-search-2d-case/

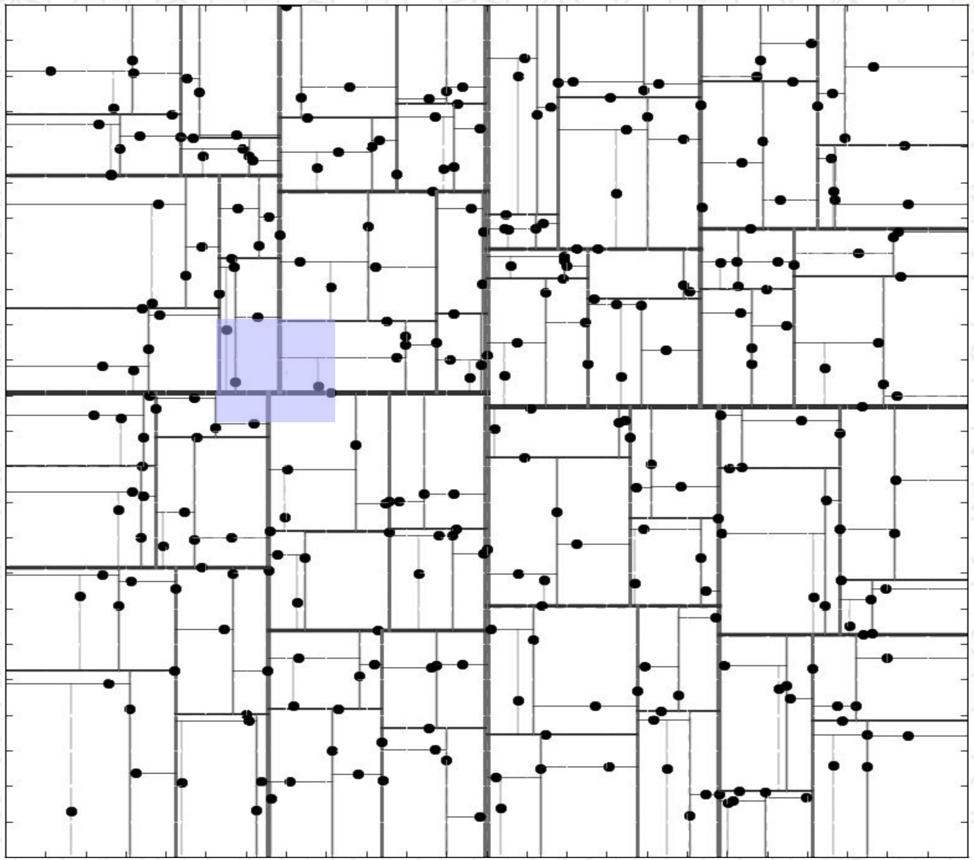
- 1 item is stored per leaf node
- For a query that will collect *k* items



2D kd Tree Query Analysis

https://salzis.wordpress.com/2014/06/28/kd-tree-and-nearest-neighbor-nn-search-2d-case/

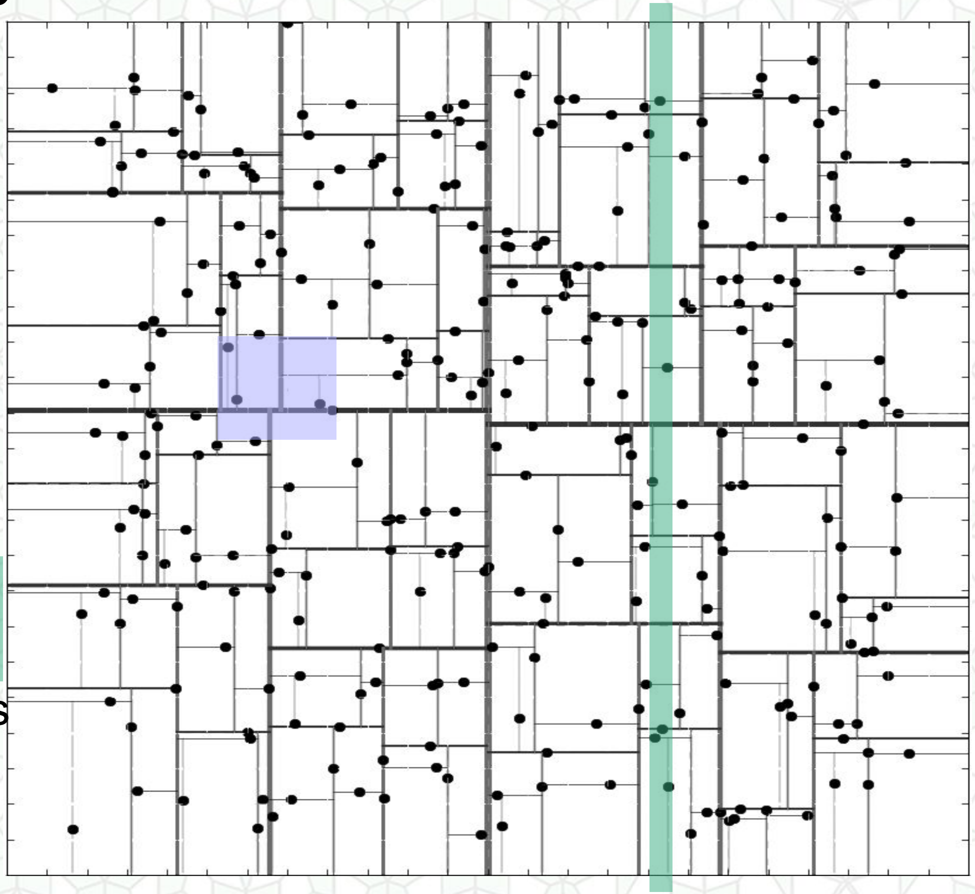
- 1 item is stored per leaf node
- For a query that will collect *k* items
- Best/Average(?) Case:
  An approximately square query
  (equal width & height)
  - touches/overlaps *O(k)* leaves
  - gathering leaves *O(log n + k)*
  - *Overall* → O(log n + k)



2D kd Tree Query Analysis

https://salzis.wordpress.com/2014/06/28/kd-tree-and-nearest-neighbor-nn-search-2d-case/

- 1 item is stored per leaf node
- For a query that will collect *k* items
- Best/Average(?) Case:
  An approximately square query
  (equal width & height)
  - touches/overlaps *O(k)* leaves
  - gathering leaves *O(log n + k)*
  - ***Overall* → O(log n + k)**
- Worst Case Query:
  For a skinny / lopsided query box
  - touches/overlaps - √*n* +*k* leaves
  - gathering leaves O(√*n* +*k*)
  - ***Overall* → O(√n + k)**



Analysis: 2D *k*d Tree

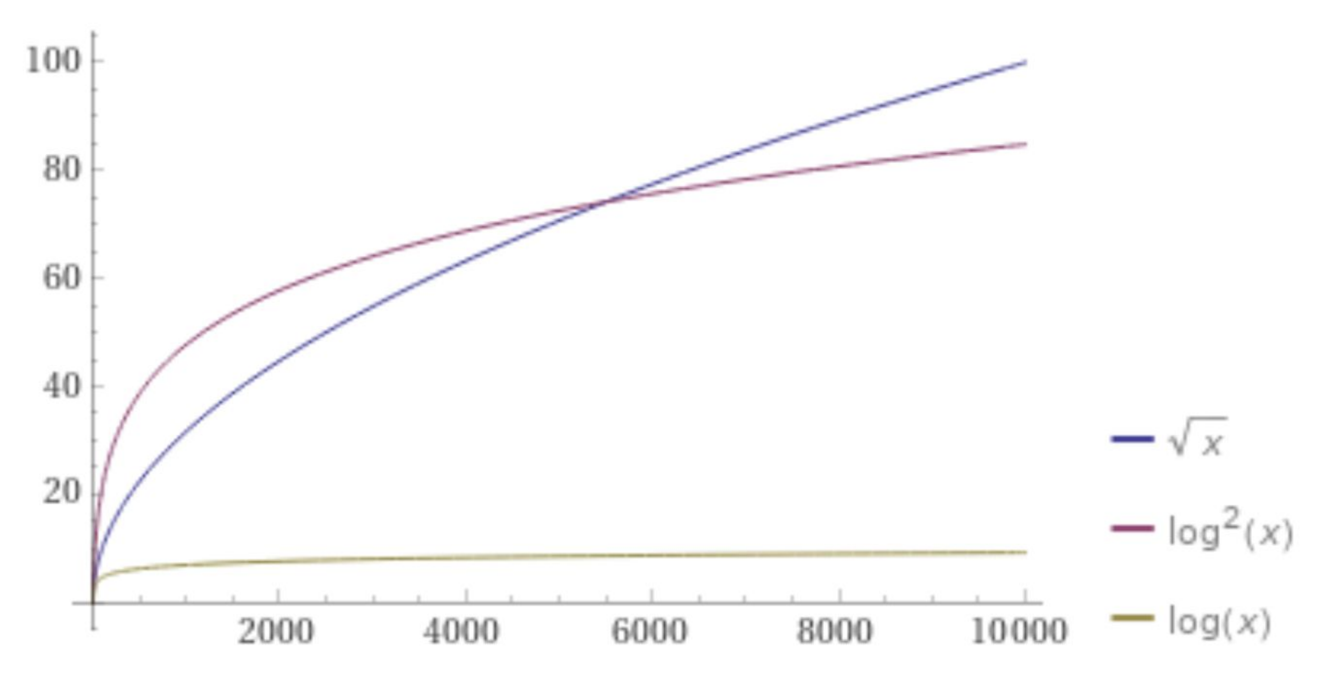
Starting with *n* values..

- Memory to store: → $O(n)$
  - # of leaf nodes: n
  - # of intermediate nodes: $n-1$
  - Height of tree: $\log n$
- Time to construct: → $O(n \log n)$
  - pre-sort the data, separately in x and in y: $O(n \log n)$
  - Alternate axes - place middle value at root, recurse on the two sublists: $O(n \log n)$
- Time to query: → $\mathbf{O(n^{1/2} + k) = O(\sqrt{n} + k)}$ *in worst case*
  - For search target / output returning k values

Is Query Time = $O(\sqrt{n} + k)$ *a problem?*

Is Query Time = $O(\sqrt{n} + k)$ a problem?

$O(1) < O(\log n) < O(\log^2 n) < O(\sqrt{n}) < O(n)$



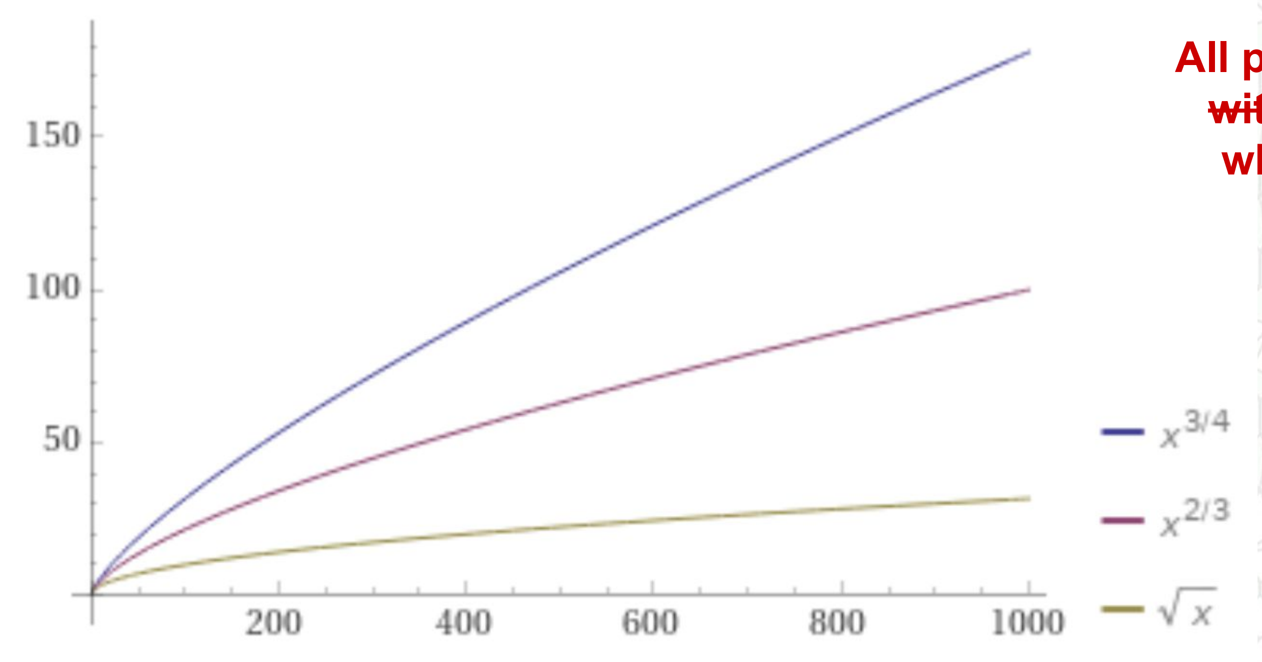
Analysis: 3D *k*d Tree and higher dimensions

Starting with n values..

- Memory to store: → $O(n)$
  - # of leaf nodes: n
  - # of intermediate nodes: $n-1$
  - Height of tree: $\log n$
- Time to construct: → $O(n \log n)$
  - pre-sort the data, separately in x and in y *and in* z: $O(n \log n)$
  - Rotate through axes (x, y, z, x, …) – place middle value at root, recurse on two sublists: $O(n \log n)$
- Time to query: → $\boldsymbol{O(n^{2/3} + k)}$ → $\boldsymbol{O(n^{(1-1/d)} + k)}$ *in worst case*
  - For search target / output returning k values

Is Query Time = $O(n^{(1-1/d)} + k)$ a problem?

- Yeah, this is a problem as dimensions increase
- ***Common for complex databases and typical, interesting queries***



All people born 1950-1960,
~~with salary 3,000-4,000,~~
who have 2-4 children

With *ANY* salary…

Becomes a skinny query box in one axis!

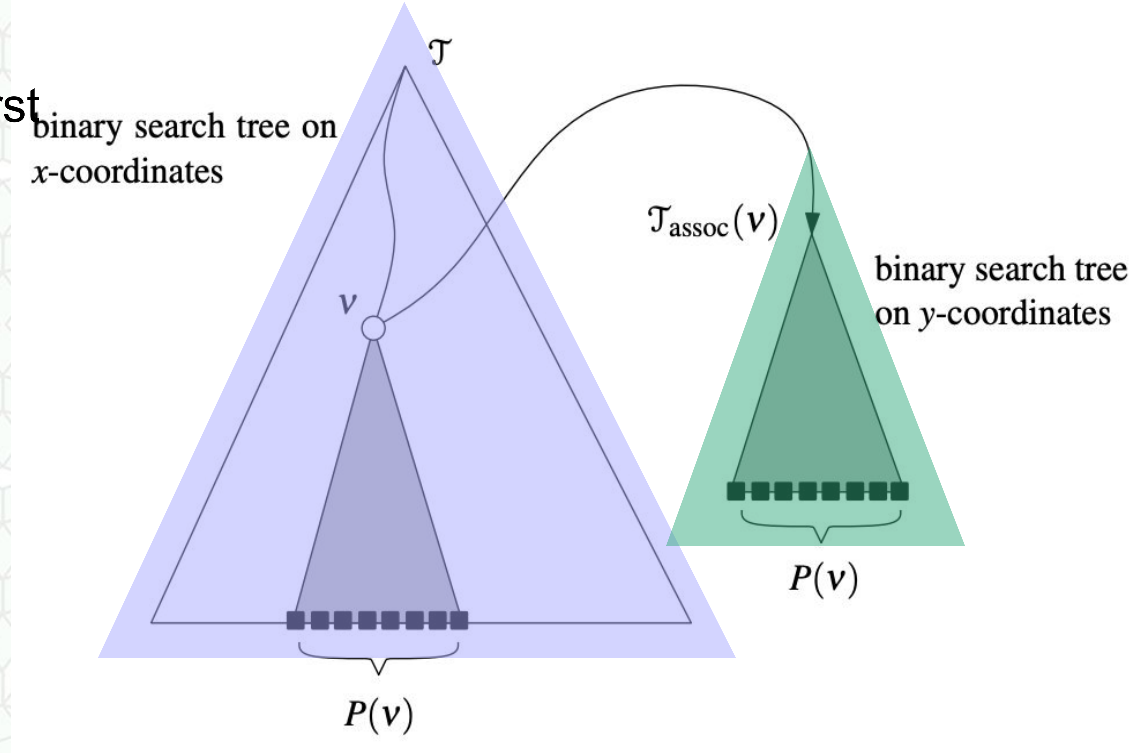
Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
  - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

What is a Range Tree?

- Idea: If we use more memory, can we reduce worst case query time of *k*D tree?

- *First we organize the data in a BST by x value*
- *At every node in the tree, we store a pointer to a BST with the same data, but organized by y value*

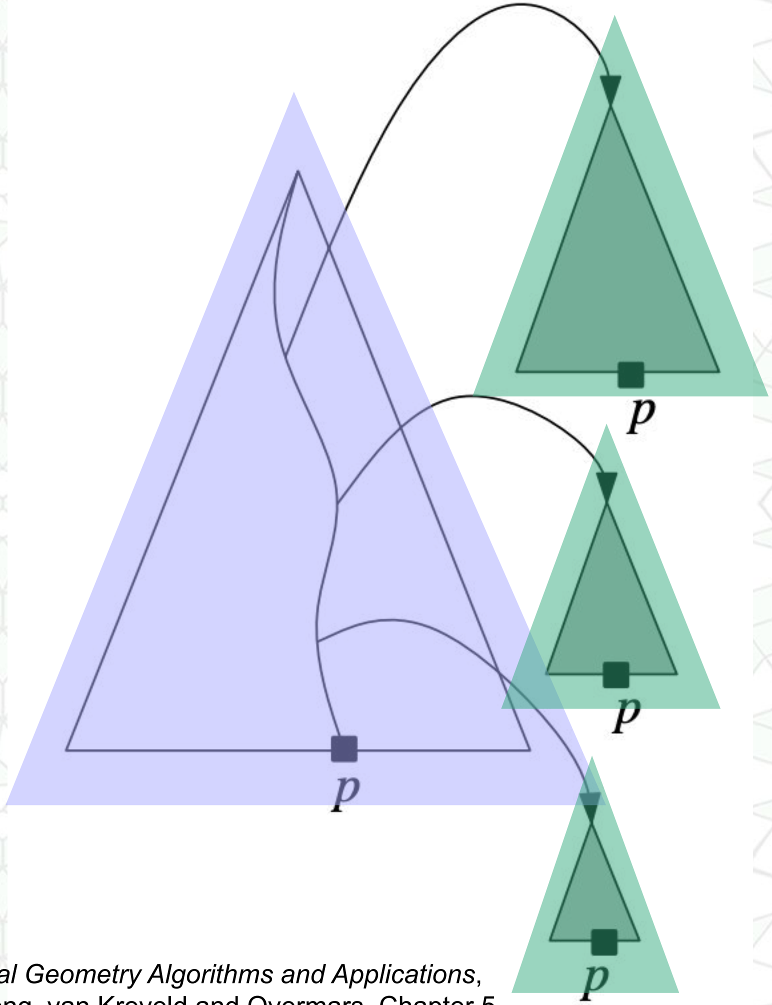


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

What is a Range Tree?

- Idea: If we use more memory, can we reduce worst case query time of *k*D tree?

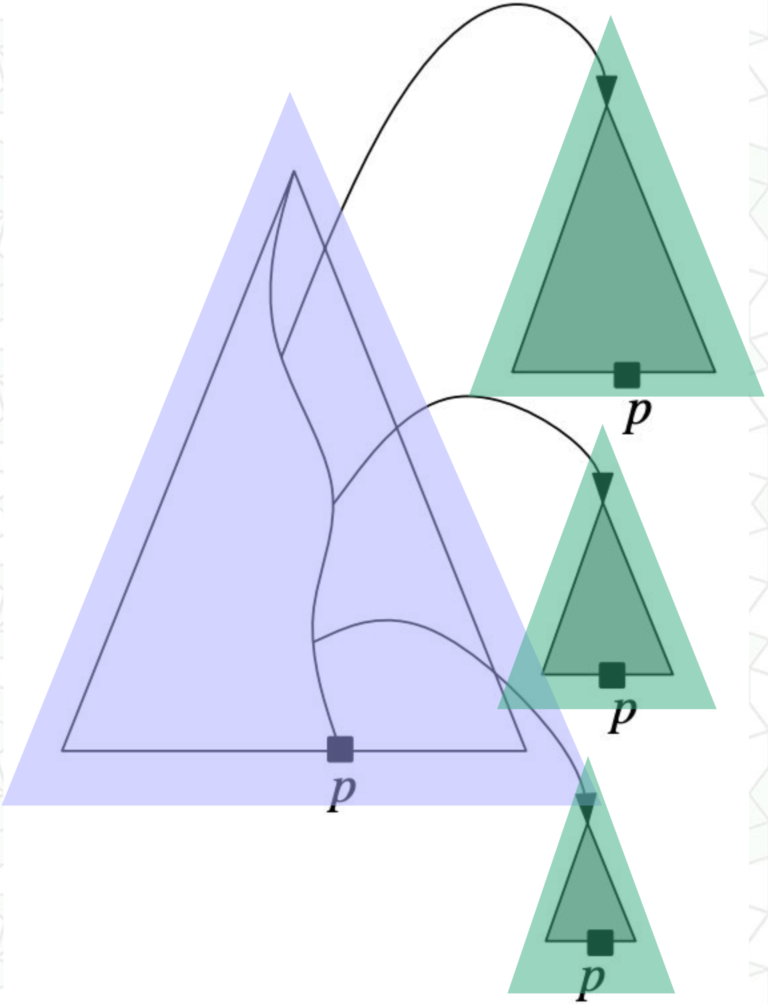
- *First we organize the data in a BST by x value*
- *At every node in the tree, we store a pointer to a BST with the same data, but organized by y value*



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5

How to Construct the 2D Range Tree?

How much memory does it use?



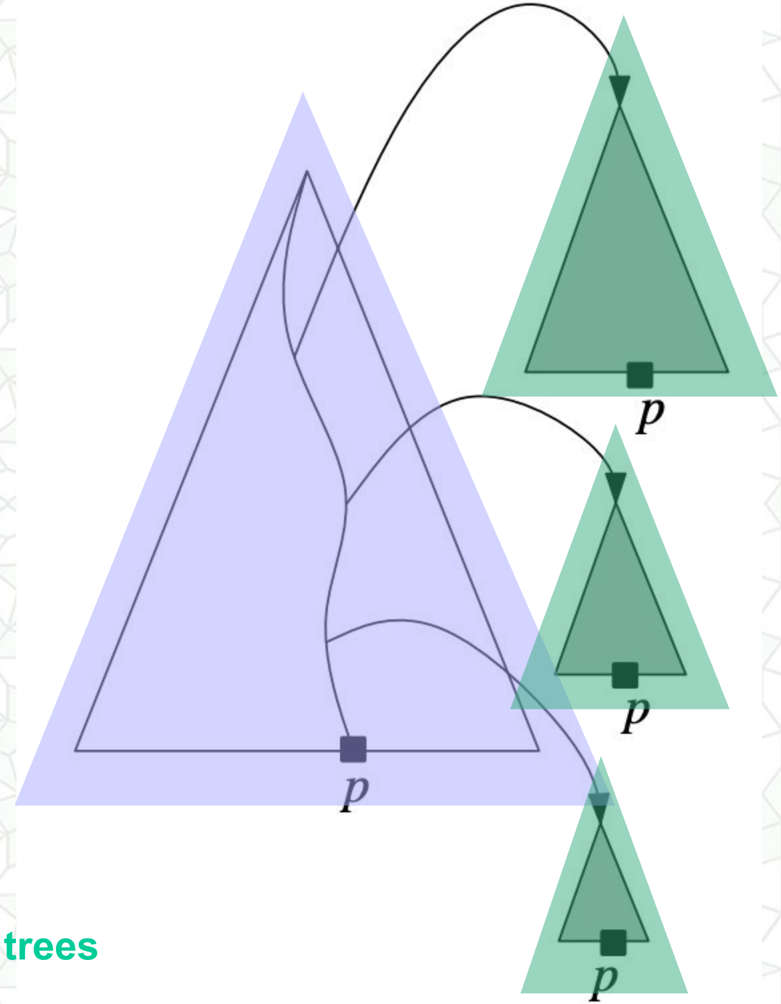
How to Construct the 2D Range Tree?

How much memory does it use?

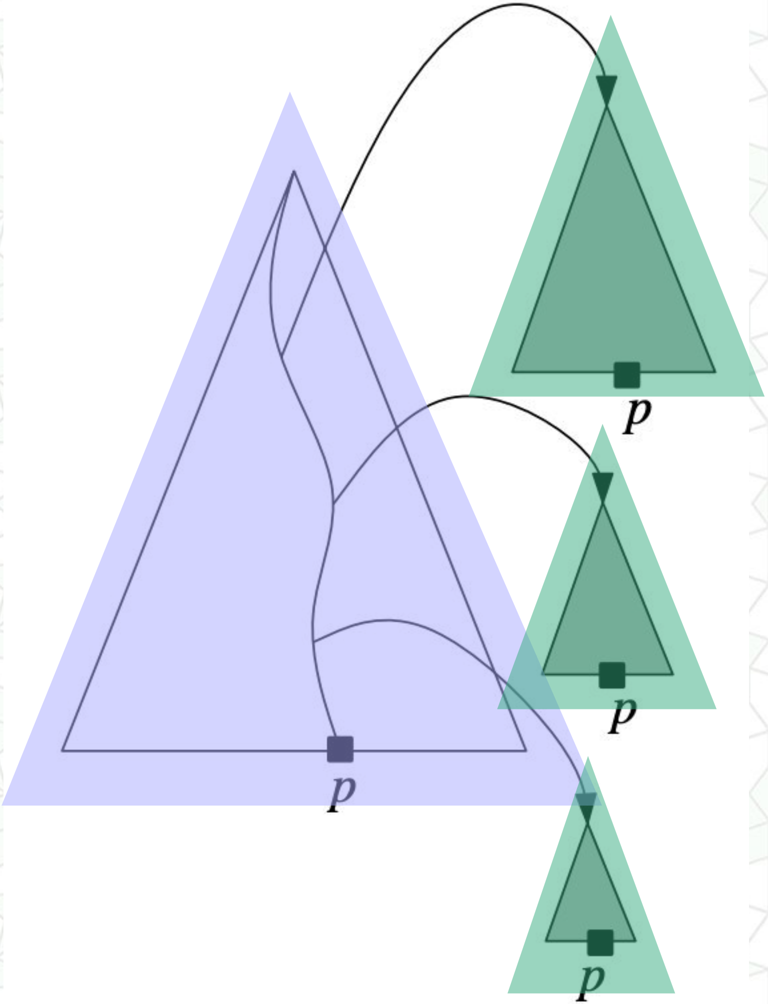
- Each point p is stored once in the level 1 (organized by x) tree
- And many times in level 2 (organized by y) trees
- How many level 2 trees? And how big are they?
  - 1 tree with n values
  - 2 trees with n/2 values
  - 4 trees with n/4 values
  - …
  - *n* trees with 1 values

→ ***O(n)* memory for the level 1 tree**

→ ***O(n log n)* memory in total for all of the level 2 trees**



How to Query 2D Range Tree?



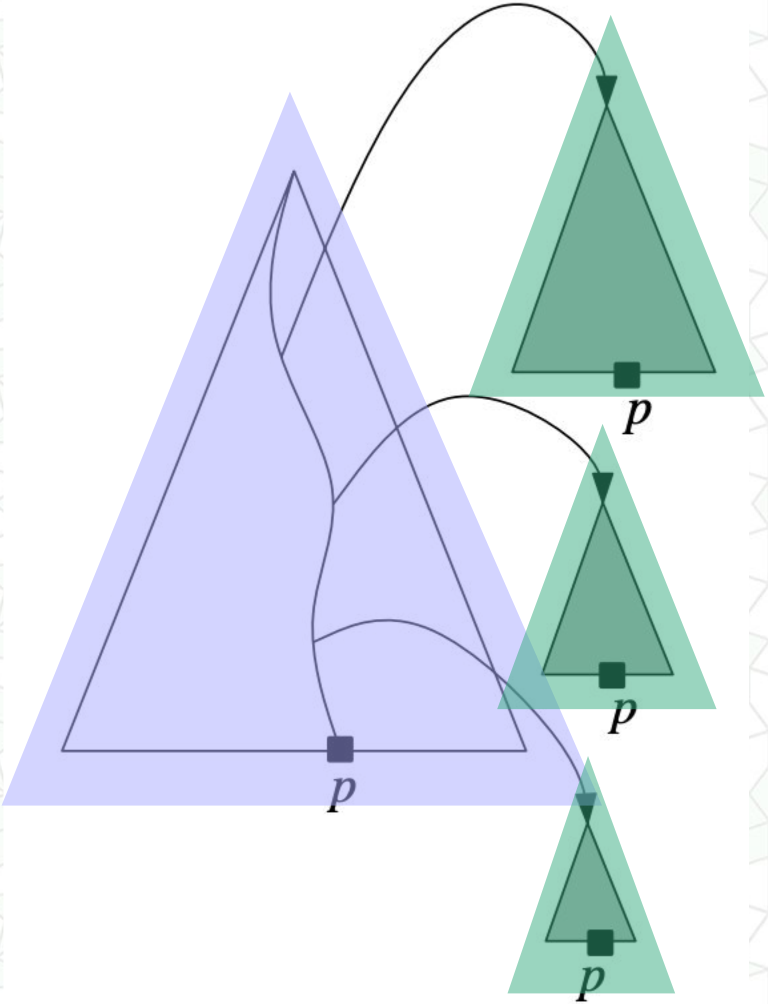
How to Query 2D Range Tree?

- Search through level 1 (blue) tree for all intermediate nodes that fit completely inside the query's *x* range

  For each matched intermediate blue node

  - Search through the corresponding level 2 (green) trees for all nodes and leaves that fit completely inside the query's y range

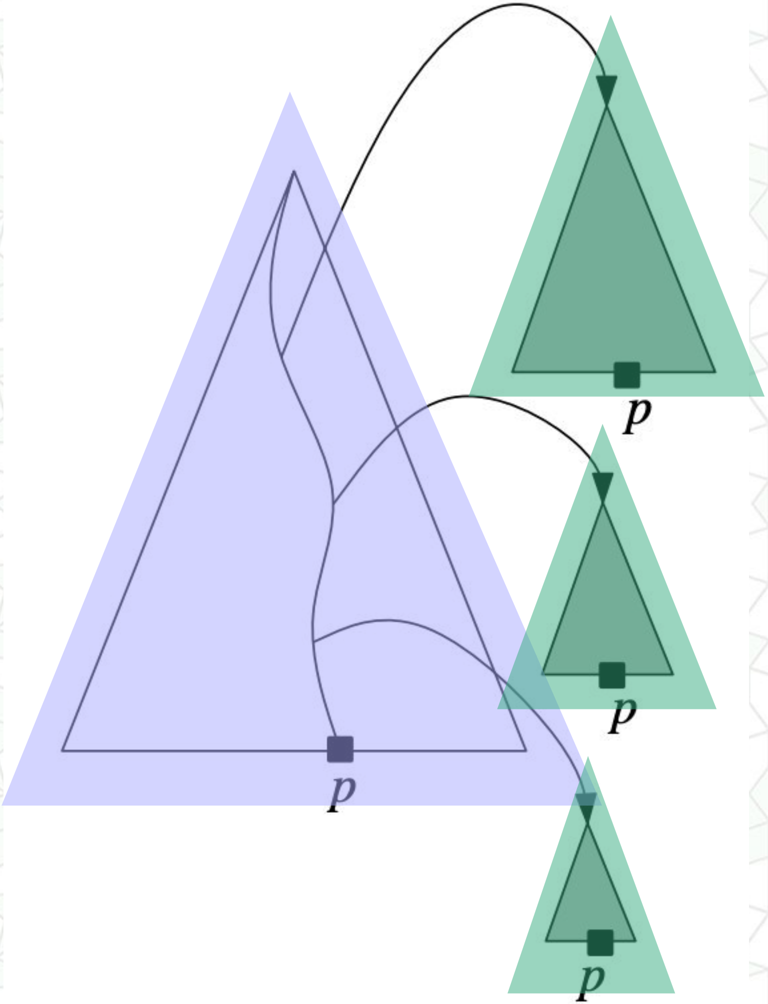
    Return all matching data!



Analysis: 2D Range Tree

Starting with *n* values..

- Memory to store:
- Time to construct:
- Time to query:



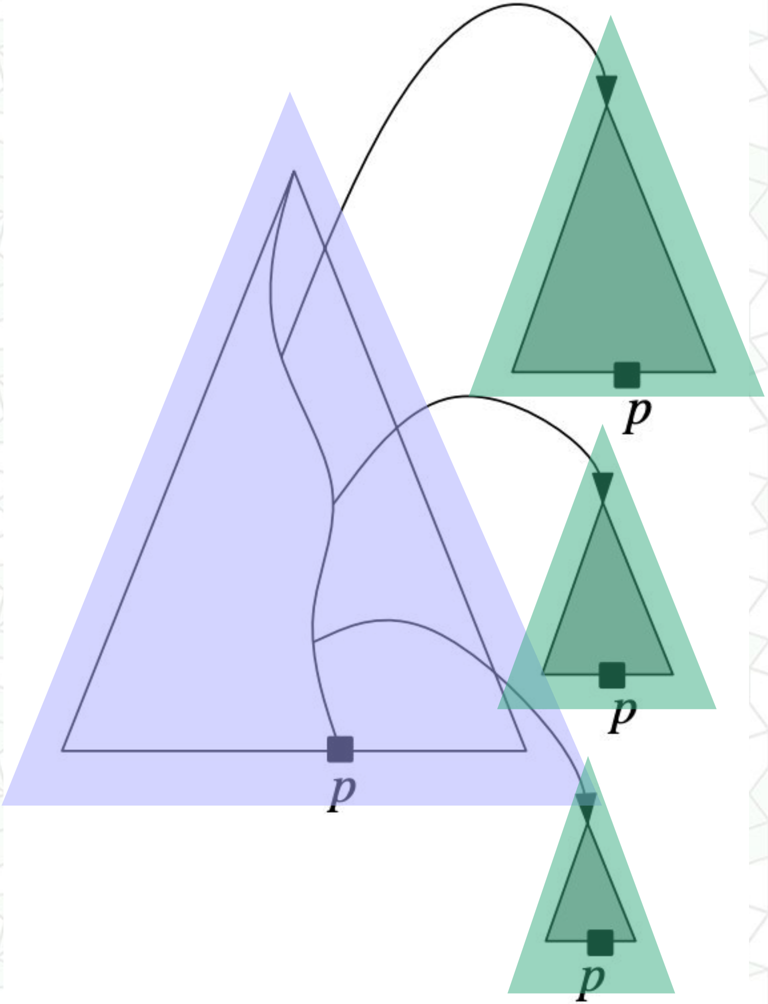
Analysis: 2D Range Tree

Starting with *n* values..

- Memory to store: → $O(n \log n)$

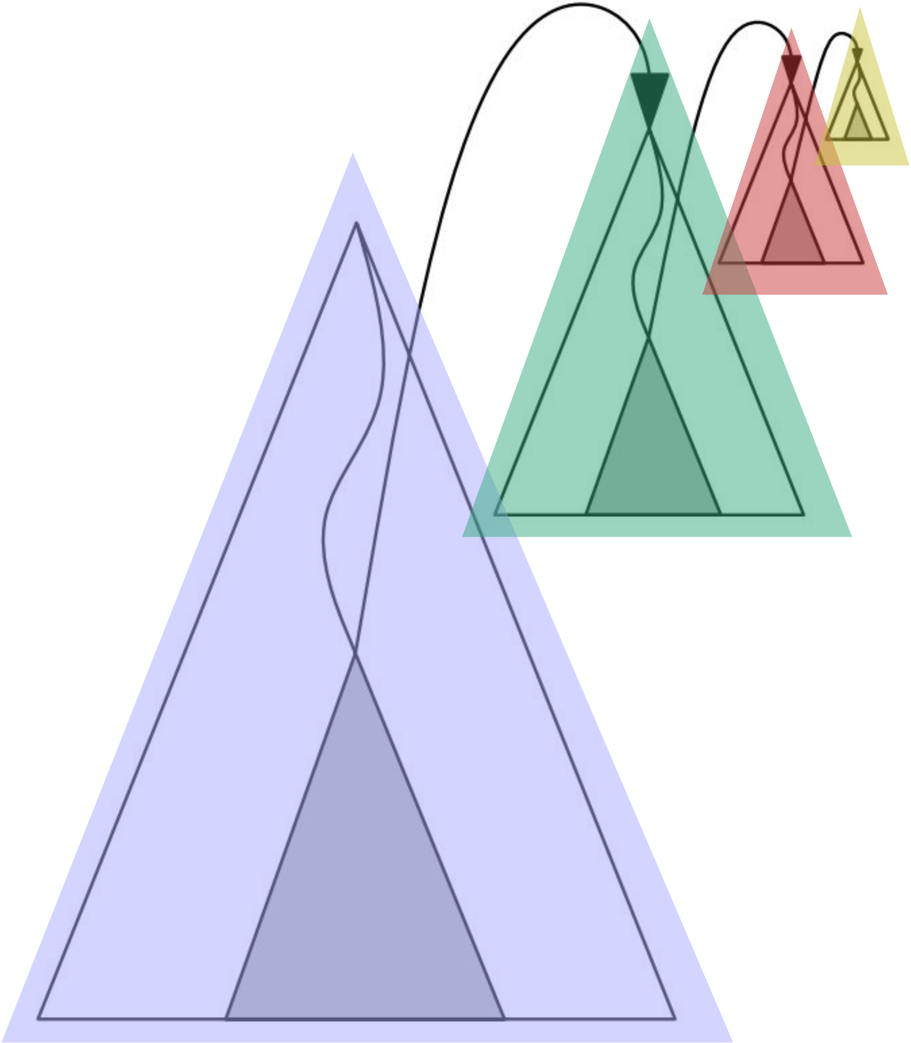
- Time to construct: → $O(n \log n)$

- Time to query: → $O(\log^2 n + k)$



Higher Dimensional Range Tree

- … and can be extended to arbitrarily higher dimensions

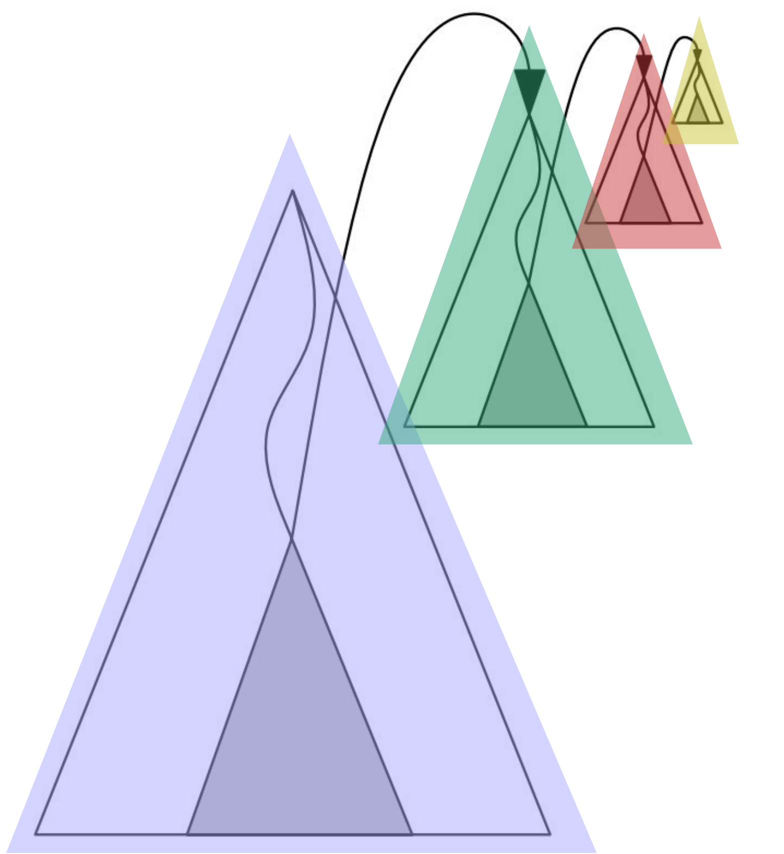


Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 5

Analysis: 3D *k*d Tree and higher dimensions

Starting with n values..

- Memory to store:
- Time to construct:
- Time to query:



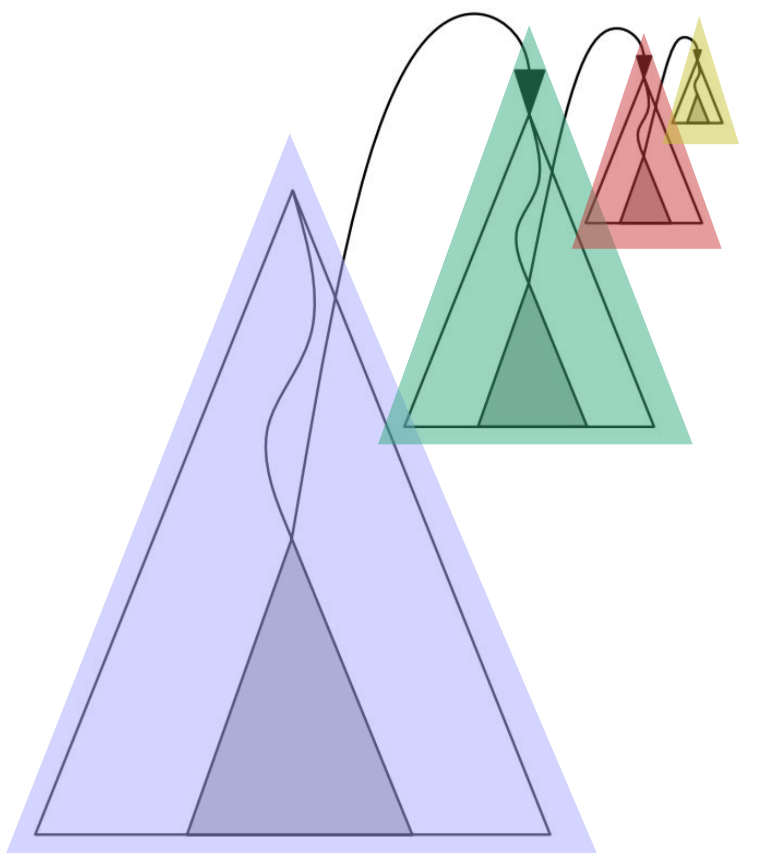
Analysis: 3D *k*d Tree and higher dimensions

Starting with *n* values..

- Memory to store: → $O(n \log^{d-1} n)$

- Time to construct: → $O(n \log^{d-1} n)$

- Time to query: → $O(\log^{d} n + k)$



Summary Comparison

- For n points, dimension d, with query to collect k items
- kd tree
    - Construction time: $\rightarrow O(n \log n)$
    - Memory: $\rightarrow O(n)$
    - Query time
        - Square(ish) box: $\rightarrow O(\log n + k)$
        - Worst case (long, skinny box): $\rightarrow O(n^{(1-1/d)} + k)$
- Range tree
    - Construction time $\rightarrow O(n \log^{d-1} n)$
    - Memory $\rightarrow O(n \log^{d-1} n)$
    - Query time $\rightarrow O(\log^{d} n + k)$

Tradeoff:
Use more memory
Faster runtime

Outline for Today

- Homework 4 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices - Evaluation Criteria
    - cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees

Next Lecture: GPS Point Localization

- Given a 2D coordinate, e.g., a latitude & longitude
- What region of the ocean contains this point?
  - Access currents, weather, etc.

NASA Scientific Visualization Studio
https://svs.gsfc.nasa.gov/