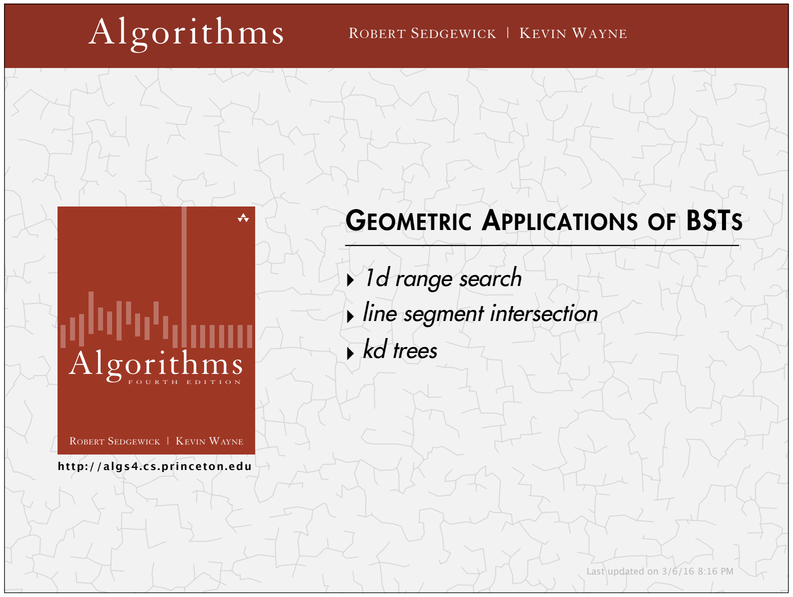


Algorithms

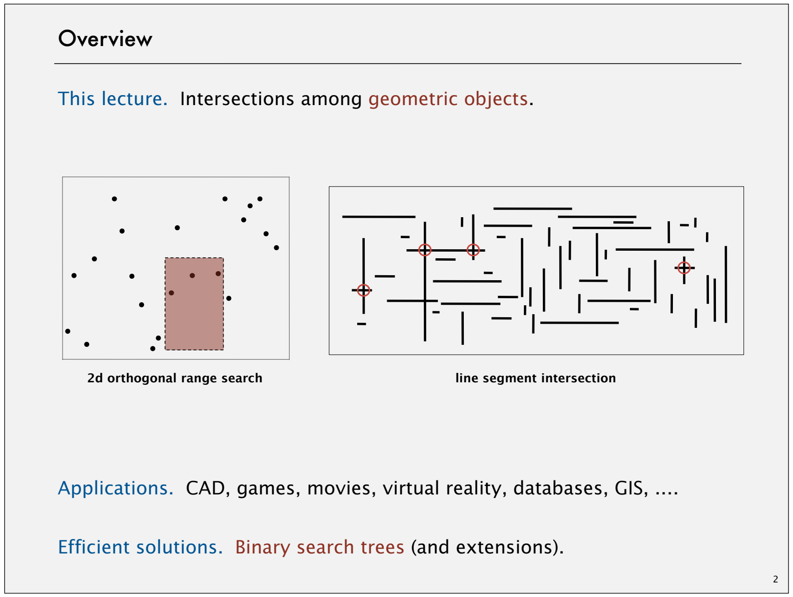
Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

- *1d range search*
- *line segment intersection*
- *kd trees*

Last updated on 3/6/16 8:16 PM



Overview

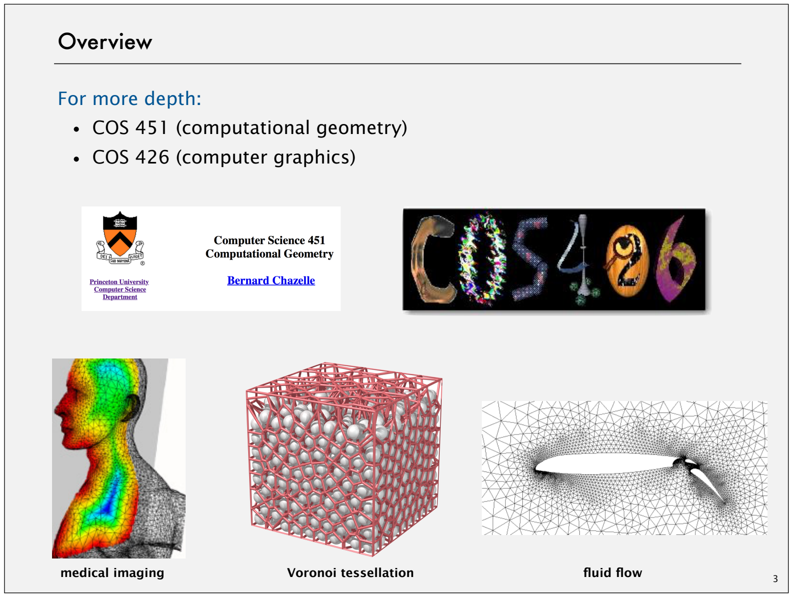
This lecture. Intersections among geometric objects.

2d orthogonal range search

line segment intersection

Applications. CAD, games, movies, virtual reality, databases, GIS, ….

Efficient solutions. Binary search trees (and extensions).



Overview

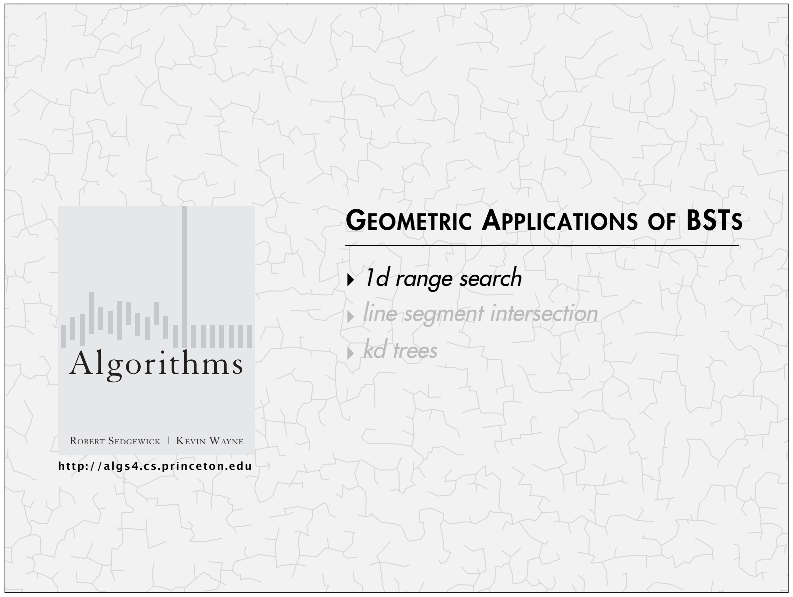
For more depth:

- COS 451 (computational geometry)
- COS 426 (computer graphics)

medical imaging

Voronoi tessellation

fluid flow



http://algs4.cs.princeton.edu

Geometric Applications of BSTs

- *1d range search*
- *line segment intersection*
- *kd trees*

1d range search

Extension of ordered symbol table.

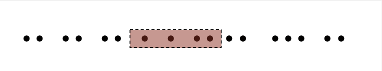
- Insert key-value pair.
- Search for key k.
- Delete key k.
- Range search: find all keys between k_1 and k_2.
- Range count: number of keys between k_1 and k_2.

Application. Database queries.

insert B	B
insert D	B D
insert A	A B D
insert I	A B D I
insert H	A B D H I
insert F	A B D F H I
insert P	A B D F H I P
search G to K	H I
count G to K	2

Geometric interpretation.

- Keys are point on a line.
- Find/count points in a given 1d interval.



Quiz 1

Suppose that the keys are stored in a sorted array. What is the order of growth of the running time to perform range count as a function of N and R?

N = number of keys
R = number of matching keys

A. $\log R$

B. $\log N$

C. $\log N + R$

D. $N + R$

E. *I don't know.*

Quiz 2

Suppose that the keys are stored in a sorted array. What is the order of growth of the running time to perform range search as a function of N and R?

N = number of keys
R = number of matching keys

A. $\log R$

B. $\log N$

C. $\log N + R$

D. $N + R$

E. *I don't know.*

1d range search: elementary implementations

Ordered array. Slow insert; fast range search.
Unordered list. Slow insert; slow range search.

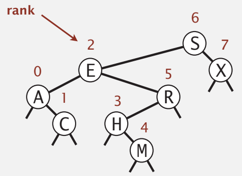
order of growth of running time for 1d range search

data structure	insert	range count	range search
ordered array	N	$\log N$	$R + \log N$
unordered list	N	N	N
goal	$\log N$	$\log N$	$R + \log N$

N = number of keys
R = number of keys that match

1d range count: BST implementation

1d range count. How many keys between `lo` and `hi` ?



rangeCount(E, S)
- rank(S) = 6
- rank(E) = 2
- 5 keys between E and S

```
public int size(Key lo, Key hi)
{
   if (contains(hi)) return rank(hi) - rank(lo) + 1;
   else              return rank(hi) - rank(lo);
}
```

number of keys < hi

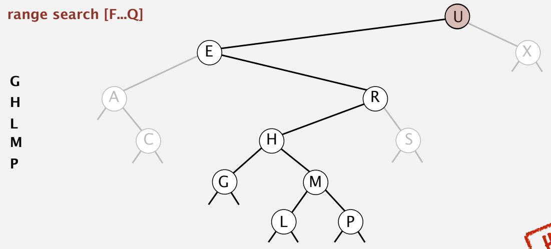
Proposition. Running time proportional to $\log N$. ← assuming BST is balanced

Pf. Nodes examined = search path to `lo` + search path to `hi`.

1d range search: BST implementation

1d range search. Find all keys between `lo` and `hi`.
- Recursively find all keys in left subtree (if any could fall in range).
- Check key in current node.
- Recursively find all keys in right subtree (if any could fall in range).

range search [F...Q]



Homework: verify proof

Proposition. Running time proportional to $R + \log N$.

Pf. Nodes examined = search path to `lo` + search path to `hi` + matches.

1d range search: summary of performance

Ordered array. Slow insert; fast range search.

Unordered list. Slow insert; slow range search.

BST. Fast insert; fast range search.

order of growth of running time for 1d range search

data structure	insert	range count	range search
ordered array	N	$\log N$	$R + \log N$
unordered list	N	N	N
goal	$\log N$	$\log N$	$R + \log N$

N = number of keys

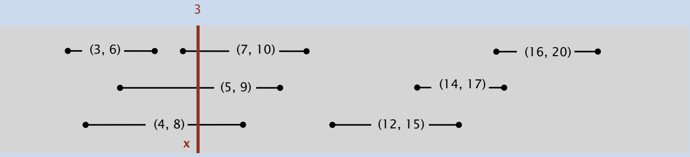
R = number of keys that match

Interval Stabbing Query

Goal. Insert intervals `(left, right)` and support queries of the form "how many intervals contain x ?"

`public class IntervalStab`

`IntervalStab()`	*create an empty data structure*
`void insert(double left, double right)`	*insert the interval (left, right) into the data structure*
`int count(double x)`	*number of intervals that contain x*



http://algs4.cs.princeton.edu

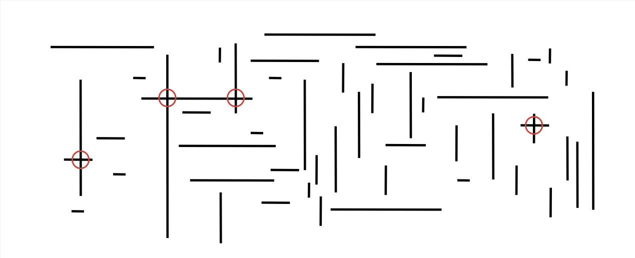
Geometric Applications of BSTs

- 1d range search
- *line segment intersection*
- kd trees

Skipped in class

Orthogonal line segment intersection

Given N horizontal and vertical line segments, find all intersections.



Quadratic algorithm. Check all pairs of line segments for intersection.

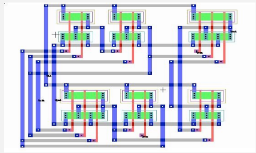
Microprocessors and geometry

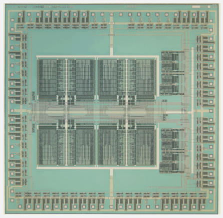
Early 1970s. microprocessor design became a geometric problem.

- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).

Design-rule checking.

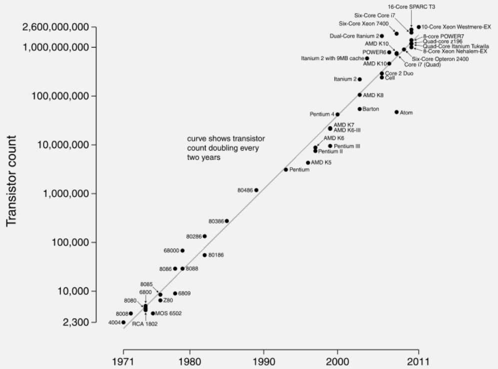
- Certain wires cannot intersect.
- Certain spacing needed between different types of wires.
- Debugging = line segment (or rectangle) intersection.

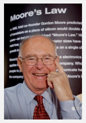




Algorithms and Moore's law

Moore's law (1965). Transistor count doubles every 2 years.





Gordon Moore

http://commons.wikimedia.org/wiki/File%3ATransistor_Count_and_Moore's_Law_-_2011.svg

Algorithms and Moore's law

Sustaining Moore's law.

- Problem size doubles every 2 years. ← problem size = transistor count
- Processing power doubles every 2 years. ← get to use faster computer
- How much $ do I need to get the job done with a quadratic algorithm?

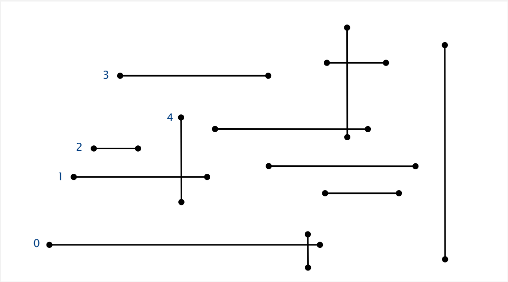
$$
\begin{aligned}
T_N &= a N^2 && \text{running time today} \\
T_{2N} &= (a/2)(2N)^2 && \text{running time in 2 years} \\
&= 2 T_N
\end{aligned}
$$

running time	1970	1972	1974	2000
N	$\$ x$	$\$ x$	$\$ x$	$\$ x$
$N \log N$	$\$ x$	$\$ x$	$\$ x$	$\$ x$
N^2	$\$ x$	$\$ 2x$	$\$ 4x$	$\$ 2^{15} x$

Bottom line. Linearithmic algorithm is necessary to sustain Moore's Law.

Orthogonal line segment intersection: sweep-line algorithm

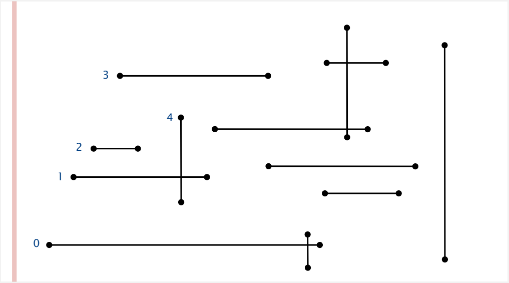
Nondegeneracy assumption. All *x*- and y-coordinates are distinct.



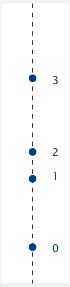
Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.



nondegeneracy assumption: all x- and y-coordinates are distinct

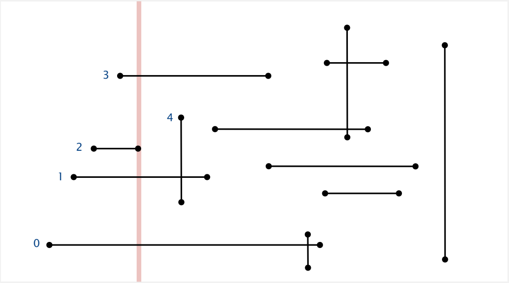


y-coordinates

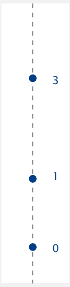
Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.
- *h*-segment (right endpoint): remove *y*-coordinate from BST.



nondegeneracy assumption: all x- and y-coordinates are distinct

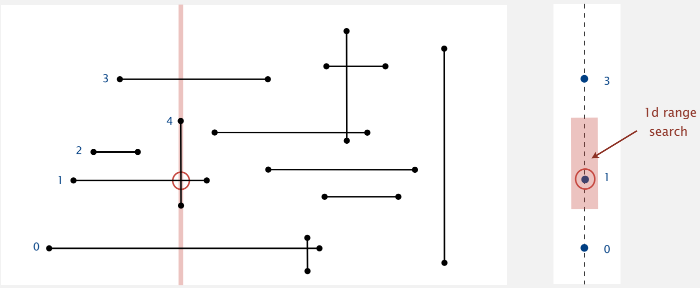


y-coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.
- h-segment (right endpoint): remove y-coordinate from BST.
- v-segment: range search for interval of y-endpoints.



nondegeneracy assumption: all x- and y-coordinates are distinct

y-coordinates

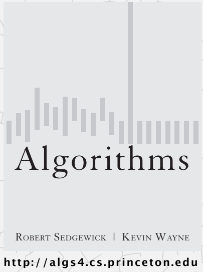
Orthogonal line segment intersection: sweep-line analysis

Proposition. The sweep-line algorithm takes time proportional to $N \log N + R$ to find all R intersections among N orthogonal line segments.

Pf.

- Put x-coordinates on a PQ (or sort). ← N log N
- Insert y-coordinates into BST. ← N log N
- Delete y-coordinates from BST. ← N log N
- Range searches in BST. ← N log N + R

Bottom line. Sweep line reduces 2d orthogonal line segment intersection search to 1d range search.



Geometric Applications of BSTs

- 1d range search
- line segment intersection
- kd trees

2-d orthogonal range search

Extension of ordered symbol-table to 2d keys.

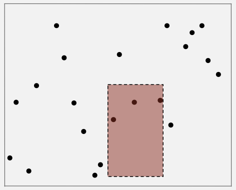
- Insert a 2d key.
- Search for a 2d key.
- Delete a 2d key.
- Range search: find all keys that lie in a 2d range.
- Range count: number of keys that lie in a 2d range.

Applications. Networking, circuit design, databases, ...

Geometric interpretation.

- Keys are point in the plane.
- Find/count points in a given h–v rectangle

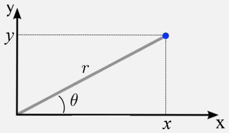
↑ rectangle is axis-aligned



Data representation

How to represent a point?

- Cartesian co-ordinates: (x, y)
- Polar co-ordinates: (r, θ)



How to represent a line segment?

- A pair of points

How to represent a line?

- (x-intercept, y-intercept)
- (x-intercept, slope)
- (y-intercept, slope)
- (distance from origin, slope)

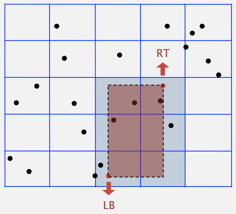
How to represent a rectangle?

- A pair of points
- (xmin, ymin, xmax, ymax)

2d orthogonal range search: grid implementation

Grid implementation.

- Divide space into M-by-M grid of squares.
- Create list of points contained in each square.
- Use 2d array to directly index relevant square.
- Insert: add (x, y) to list for corresponding square.
- Range search: examine only squares that intersect 2d range query.



2d orthogonal range search: grid implementation analysis

Space-time tradeoff.

- Space: $M^2 + N$.
- Time: $1 + N / M^2$ per square examined, on average.

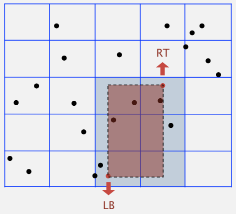
Choose grid square size to tune performance.

- Too small: wastes space.
- Too large: too many points per square.
- Rule of thumb: $\sqrt{N}$-by-$\sqrt{N}$ grid.

Running time. [if points are evenly distributed]

- Initialize data structure: N.
- Insert point: 1.
- Range search: 1 per point in range.

choose M ~ √N

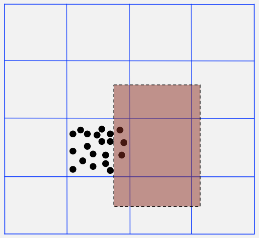


Clustering

Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

- Lists are too long, even though average length is short.
- Need data structure that adapts gracefully to data.

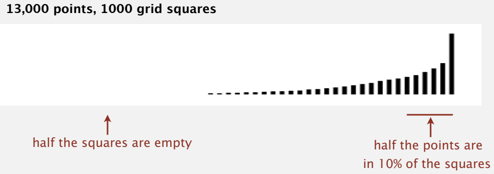


Clustering

Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

Ex. USA map data.



Space-partitioning trees

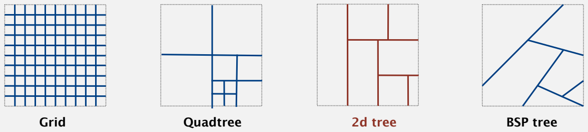
Use a tree to represent a recursive subdivision of 2d space.

Grid. Divide space uniformly into squares.

Quadtree. Recursively divide space into four quadrants.

2d tree. Recursively divide space into two halfplanes.

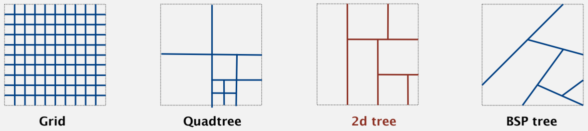
BSP tree. Recursively divide space into two regions.



Space-partitioning trees: applications

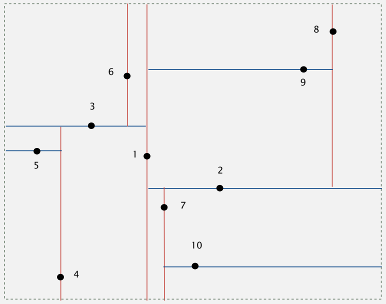
Applications.

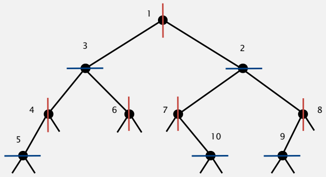
- Ray tracing.
- 2d range search.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Nearest neighbor search.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.



2d tree construction

Recursively partition plane into two halfplanes.





Quiz 3

Where would point 11 be inserted in the kd-tree below?

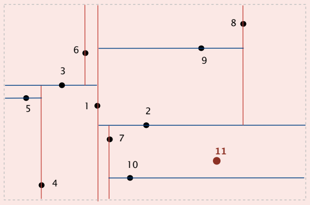
A. Right child of 6.

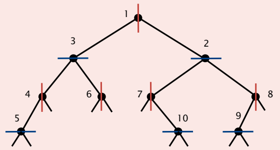
B. Left child of 7.

C. Left child of 10.

D. Right child of 10.

E. *I don't know.*

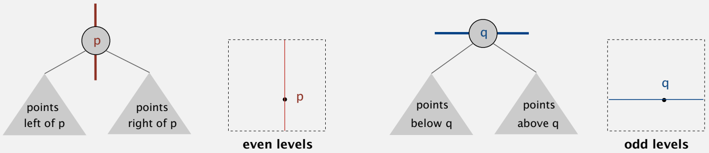


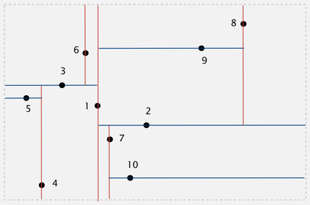


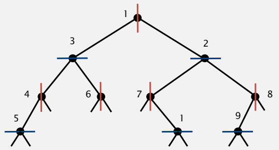
2d tree implementation

Data structure. BST, but alternate using x- and y-coordinates as key.

- Search gives rectangle containing point.
- Insert further subdivides the plane.



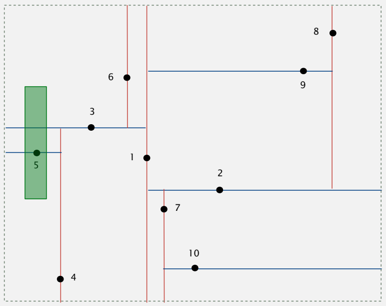


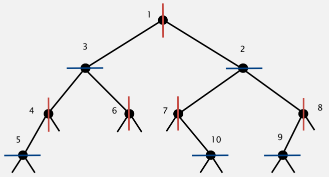


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

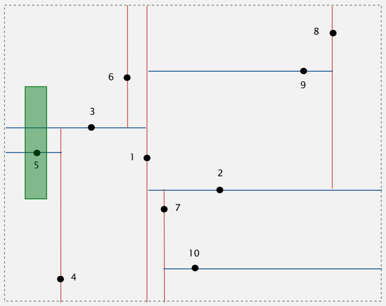


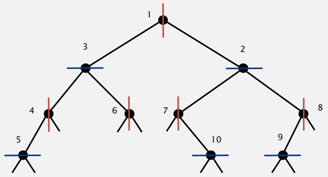


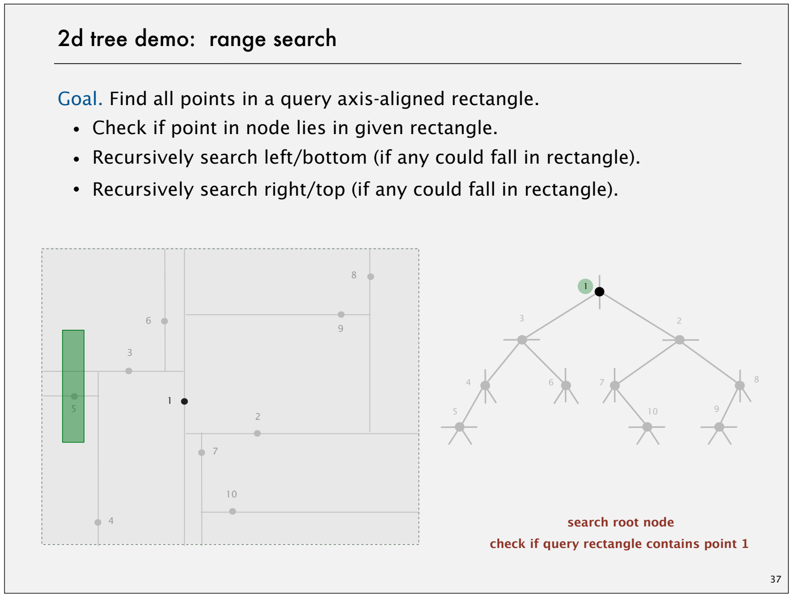
2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





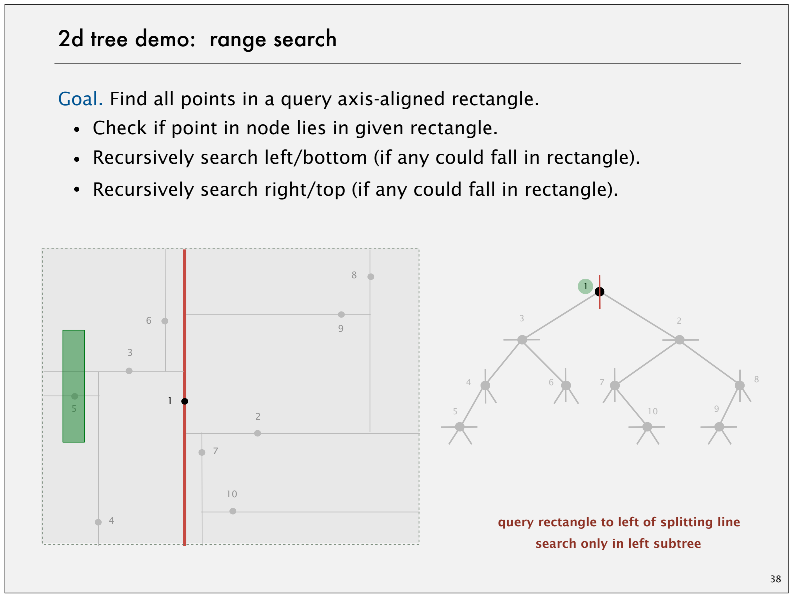


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

search root node
check if query rectangle contains point 1

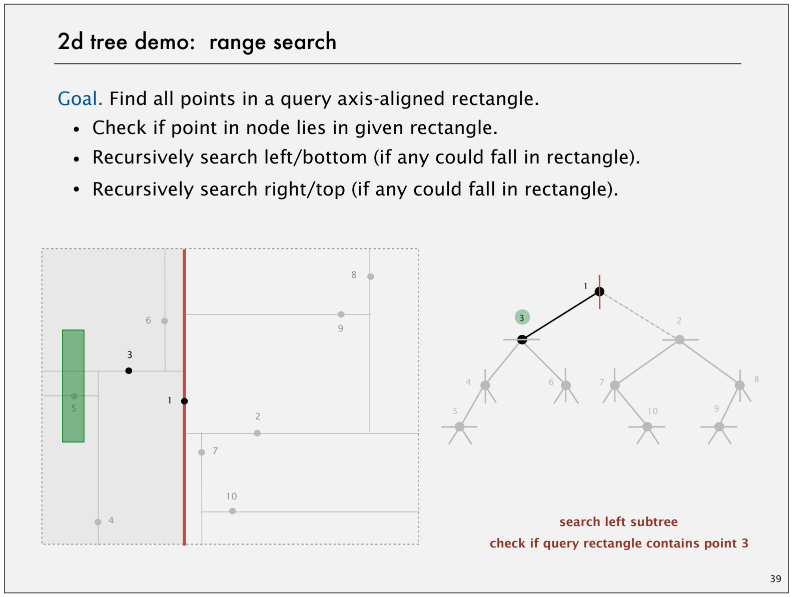


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

query rectangle to left of splitting line
search only in left subtree

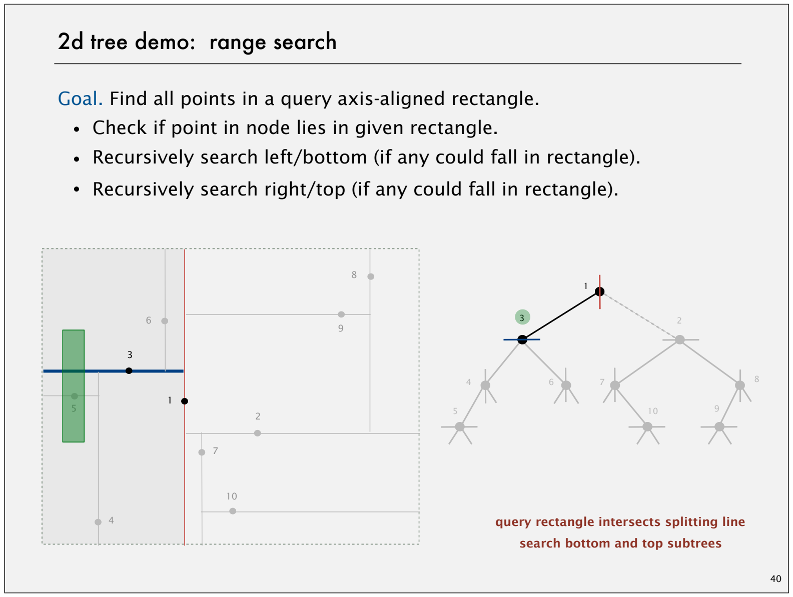


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

search left subtree
check if query rectangle contains point 3



2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

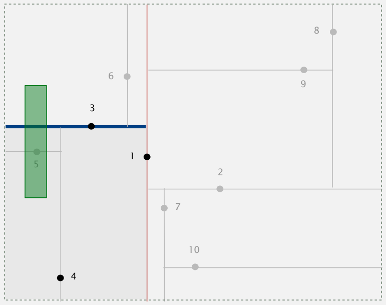
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

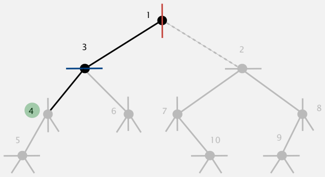
query rectangle intersects splitting line
search bottom and top subtrees

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



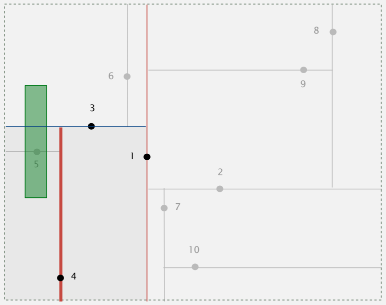


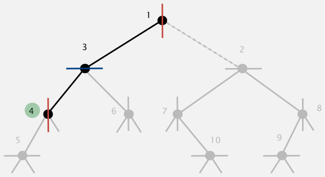
search left subtree
check if query rectangle contains point 4

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



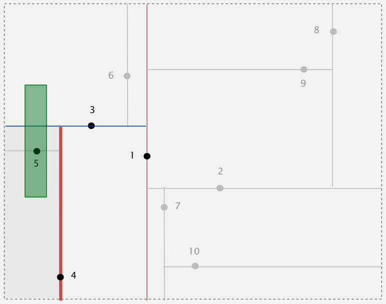


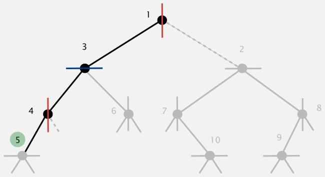
query rectangle to left of splitting line
search only in left subtree

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



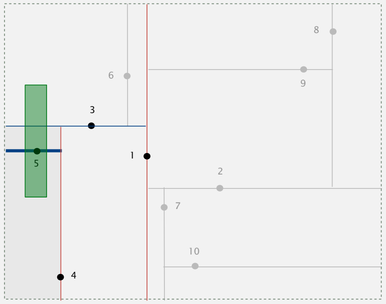


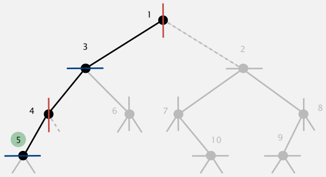
search left subtree
check if query rectangle contains point 5
(search hit)

2d tree demo: range search

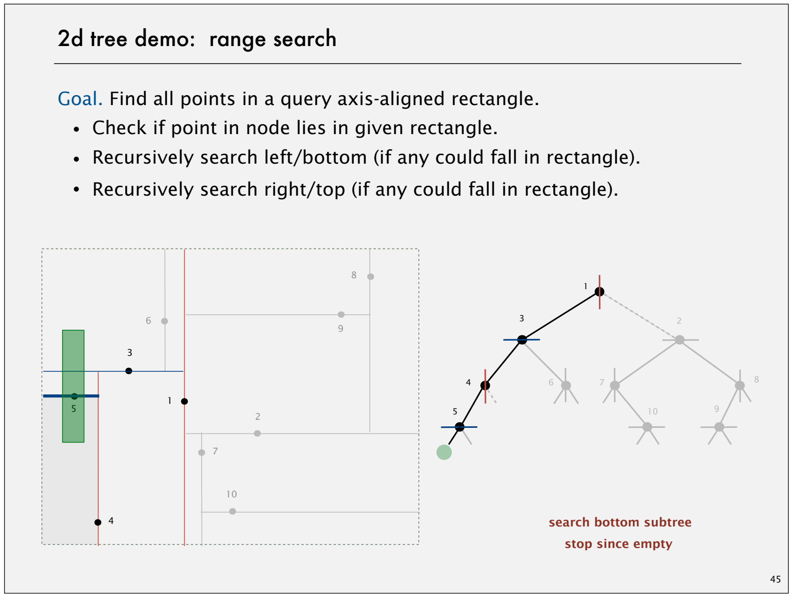
Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





query rectangle intersects splitting line
search bottom and top subtrees

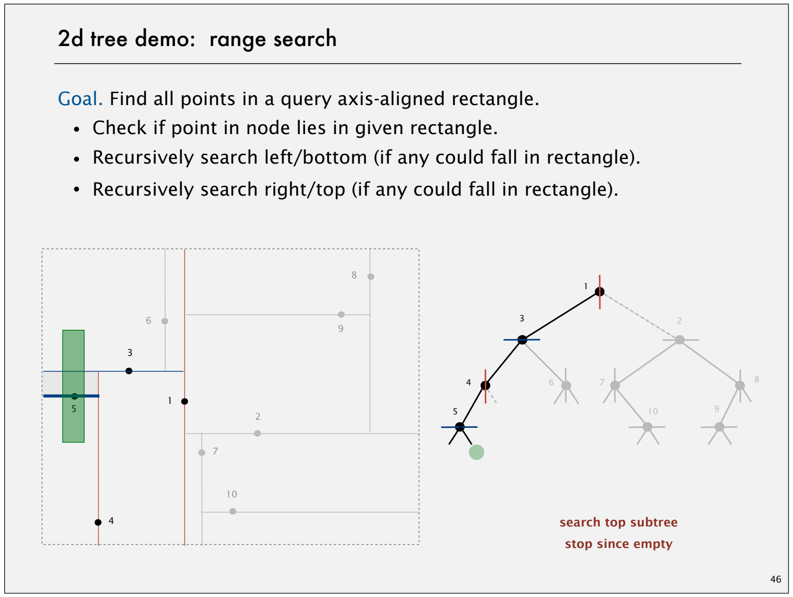


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

search bottom subtree
stop since empty

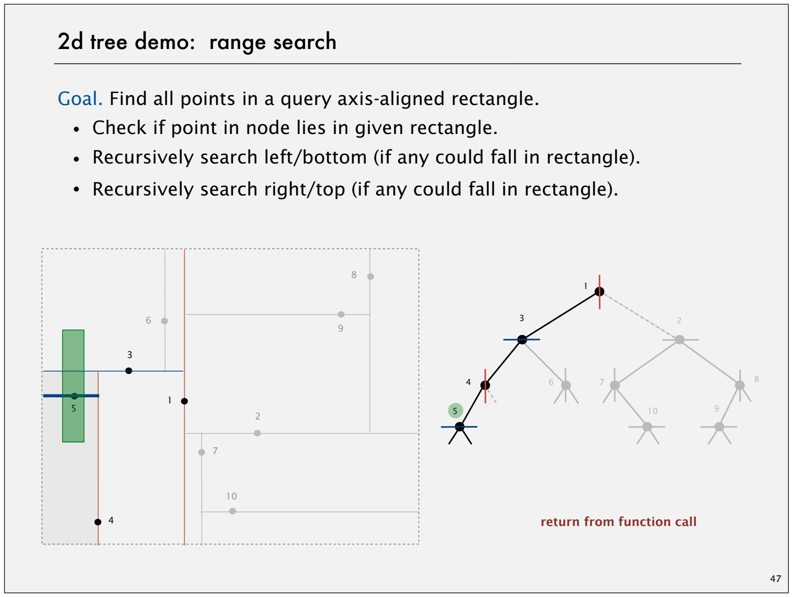


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

search top subtree
stop since empty

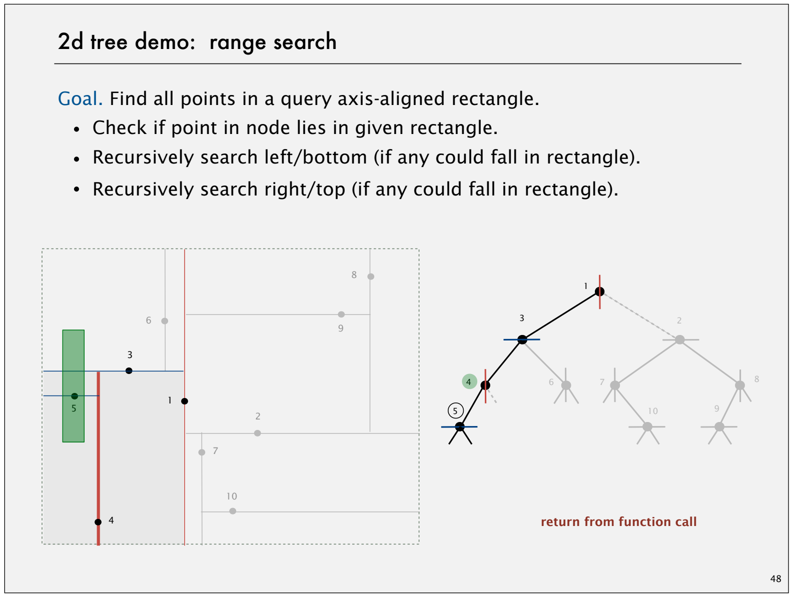


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

return from function call



2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

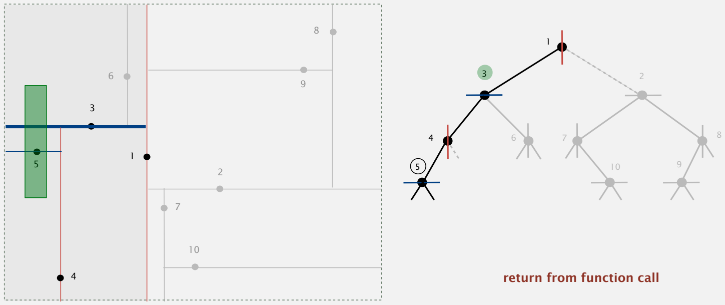
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

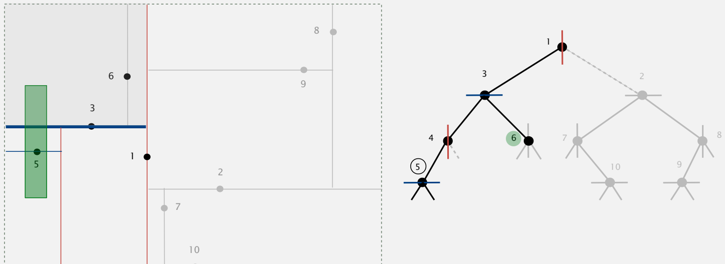


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



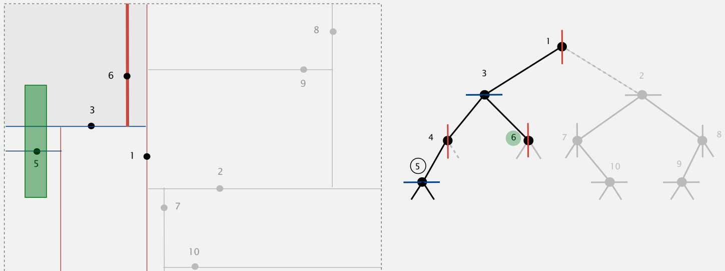
search top subtree

check if query rectangle contains point 6

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



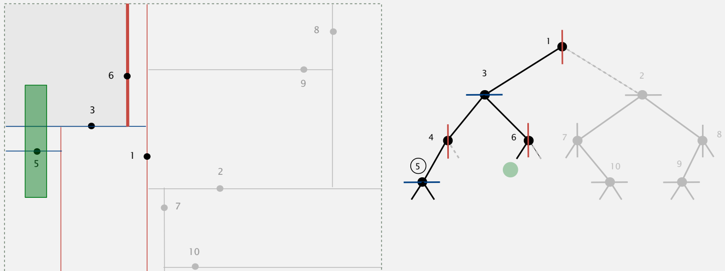
query rectangle to left of splitting line

search only in left subtree

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



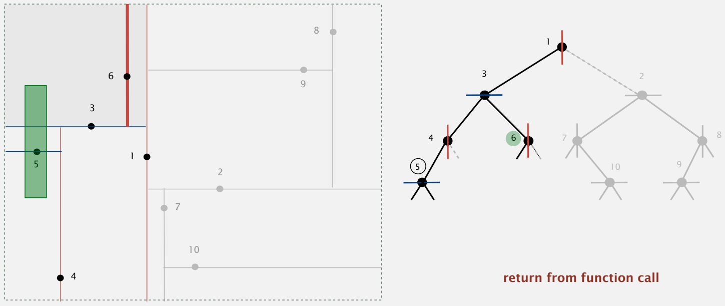
search left subtree

stop since empty

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

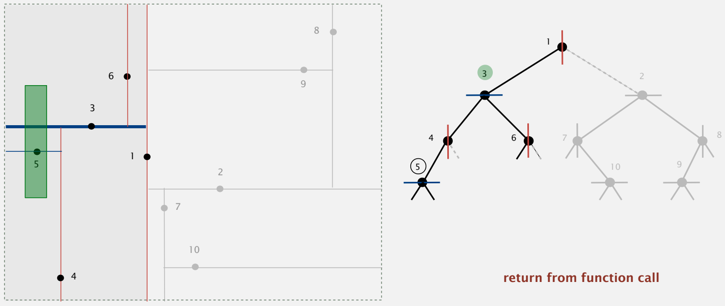


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

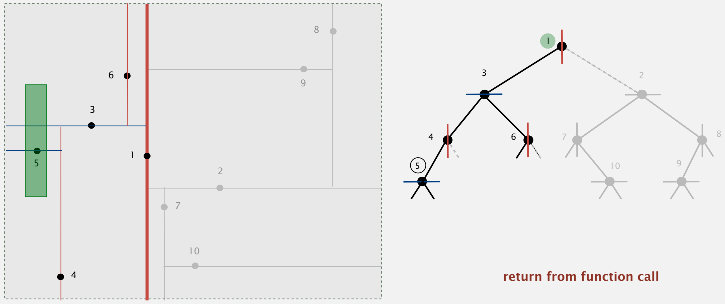


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

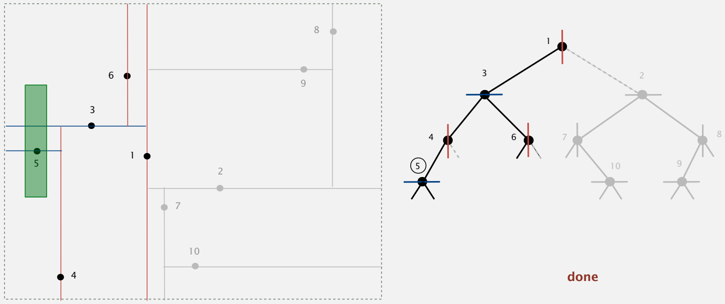


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

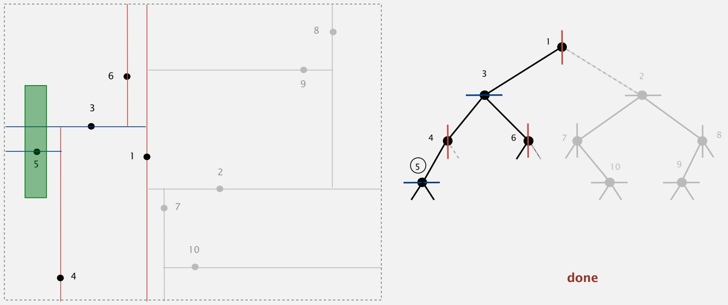


done

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

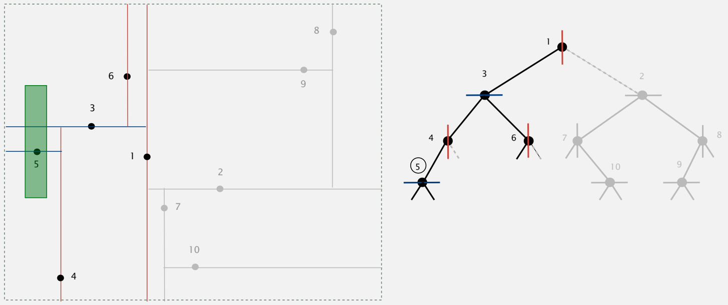
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



Range search in a 2d tree analysis

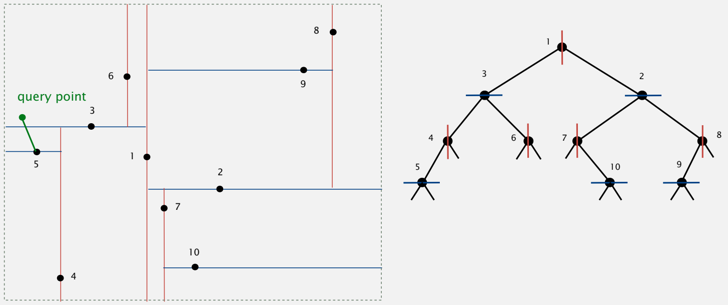
Typical case. $R + \log N$.

Worst case (assuming tree is balanced). $R + \sqrt{N}$.



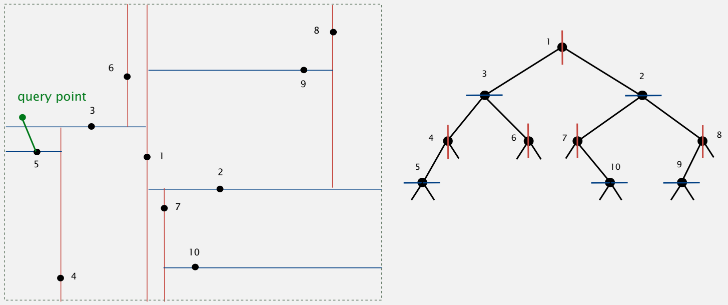
2d tree demo: nearest neighbor

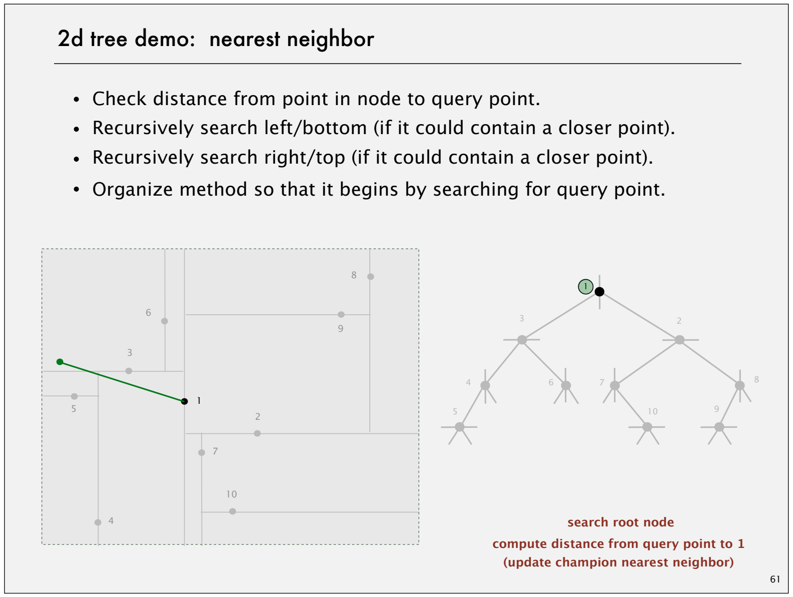
Goal. Find closest point to query point.



2d tree demo: nearest neighbor

Goal. Find closest point to query point.





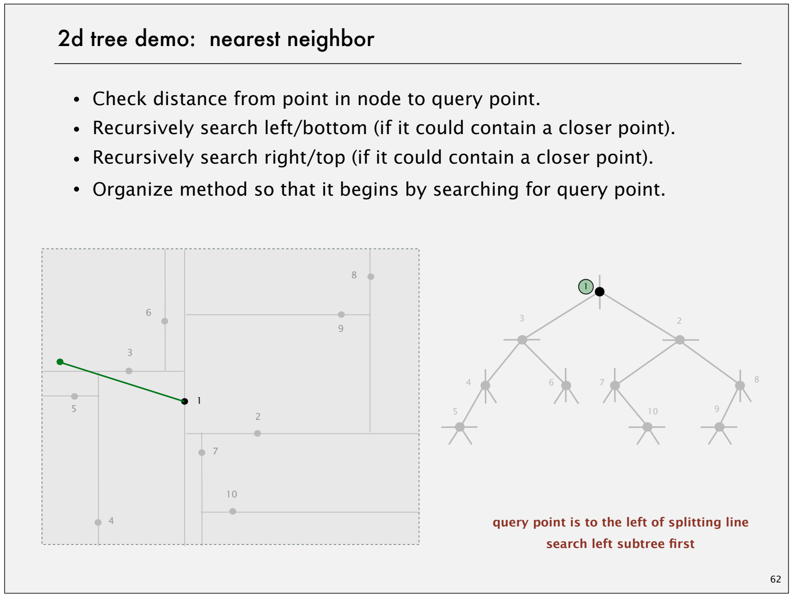
2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

search root node

compute distance from query point to 1

(update champion nearest neighbor)

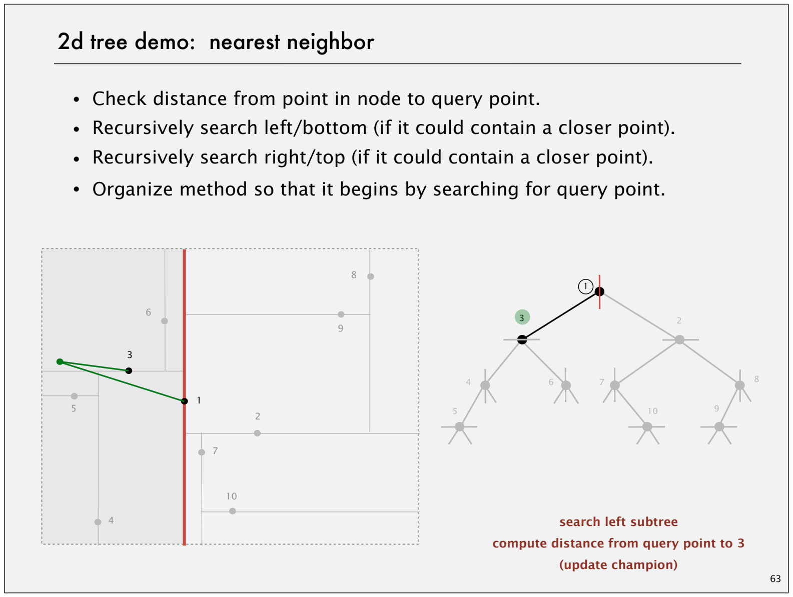


2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

query point is to the left of splitting line

search left subtree first



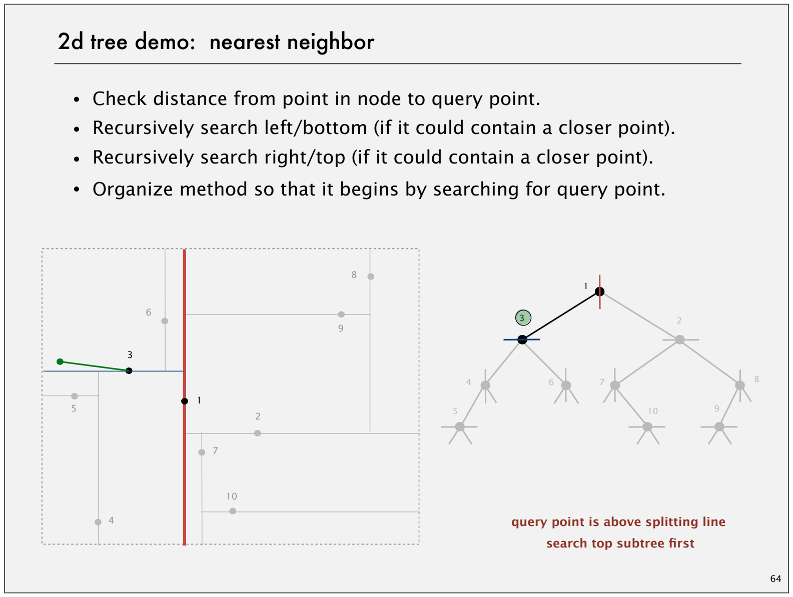
2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

search left subtree

compute distance from query point to 3

(update champion)



2d tree demo: nearest neighbor

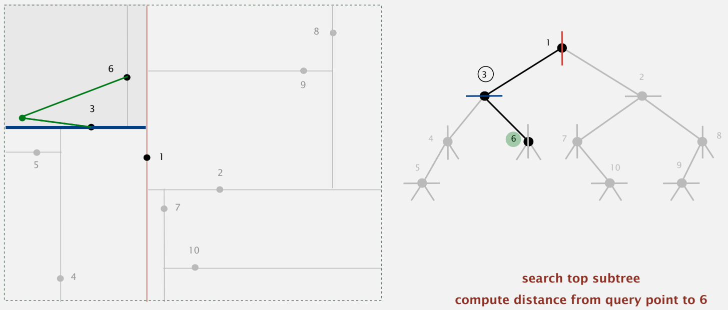
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

query point is above splitting line

search top subtree first

2d tree demo: nearest neighbor

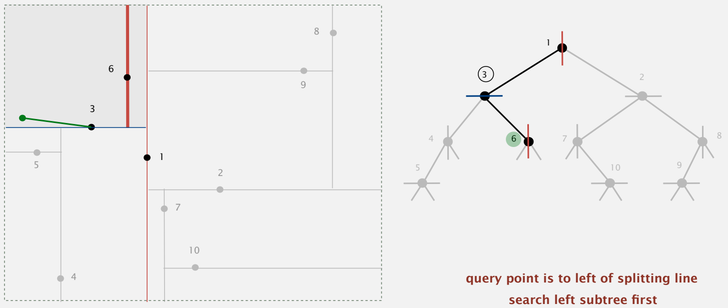
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search top subtree
compute distance from query point to 6

2d tree demo: nearest neighbor

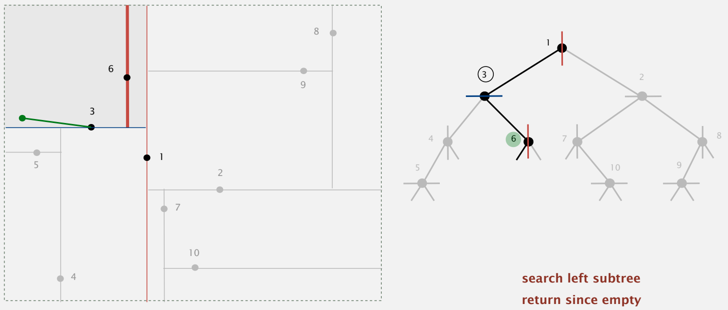
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



query point is to left of splitting line
search left subtree first

2d tree demo: nearest neighbor

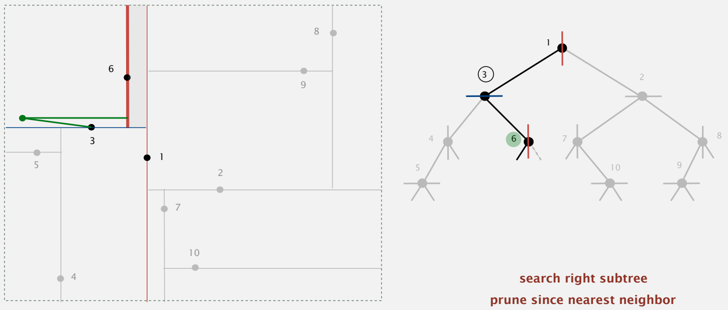
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search left subtree
return since empty

2d tree demo: nearest neighbor

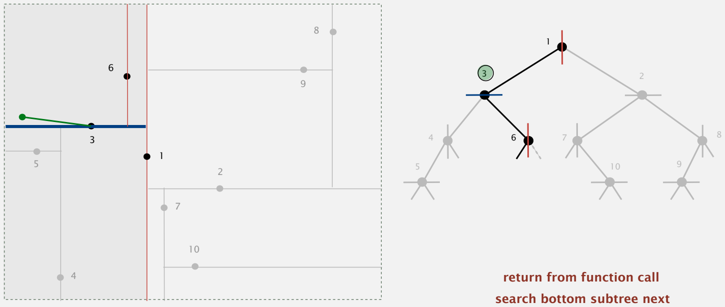
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search right subtree
prune since nearest neighbor
can't be here

2d tree demo: nearest neighbor

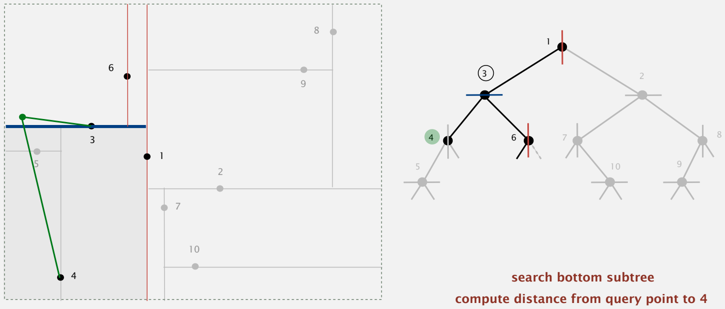
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



return from function call
search bottom subtree next

2d tree demo: nearest neighbor

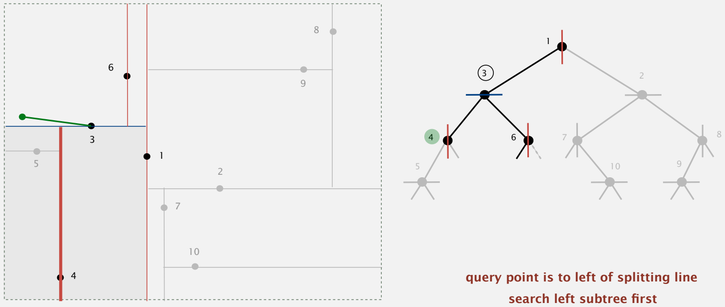
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search bottom subtree
compute distance from query point to 4

2d tree demo: nearest neighbor

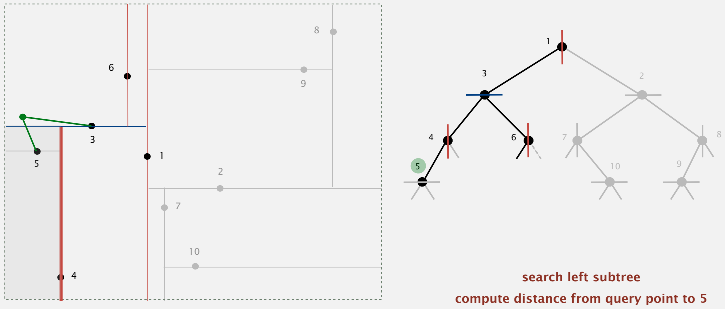
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



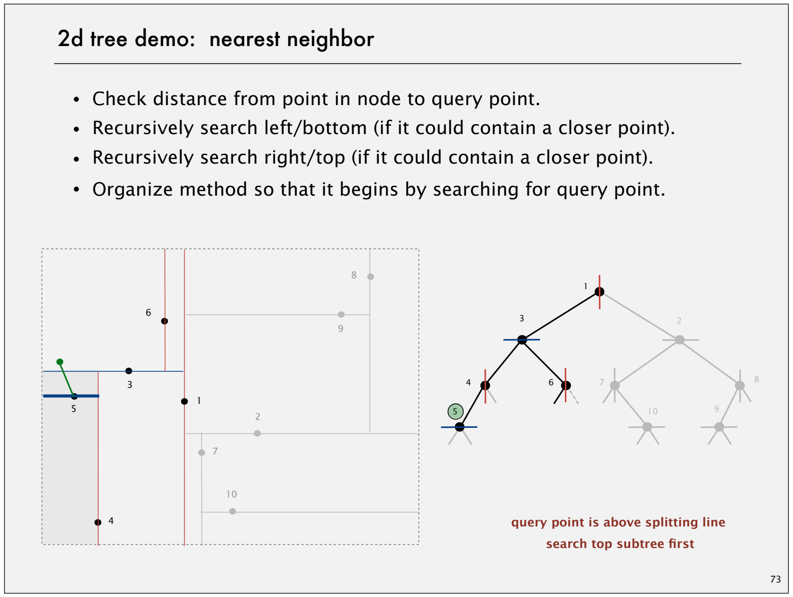
query point is to left of splitting line
search left subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



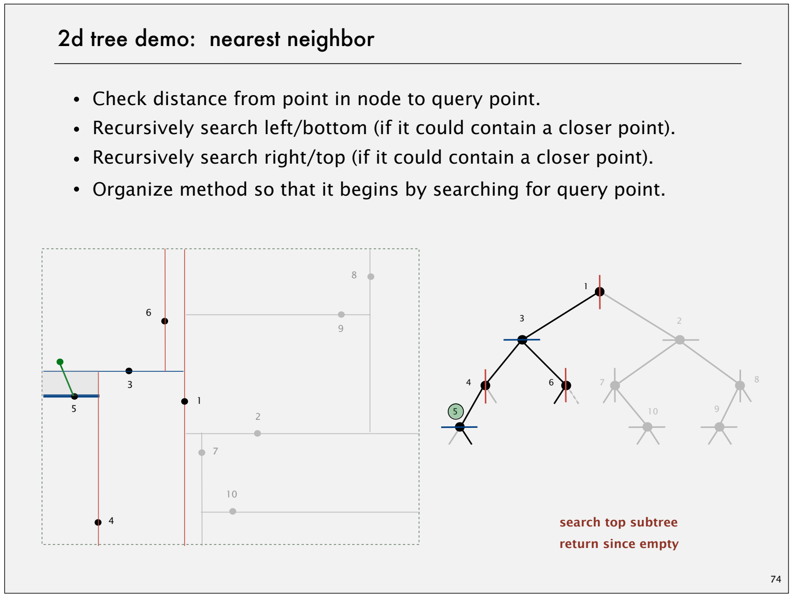
search left subtree
compute distance from query point to 5
(update champion)



2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

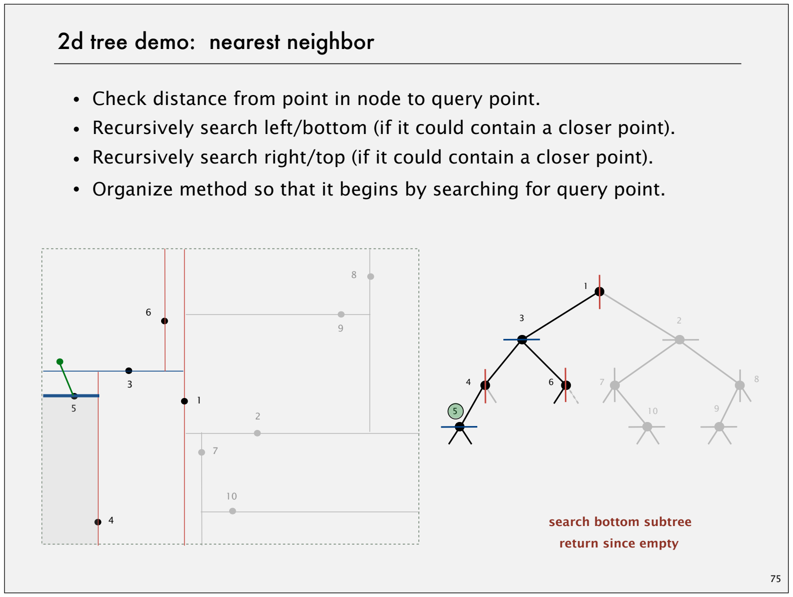
query point is above splitting line
search top subtree first



2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

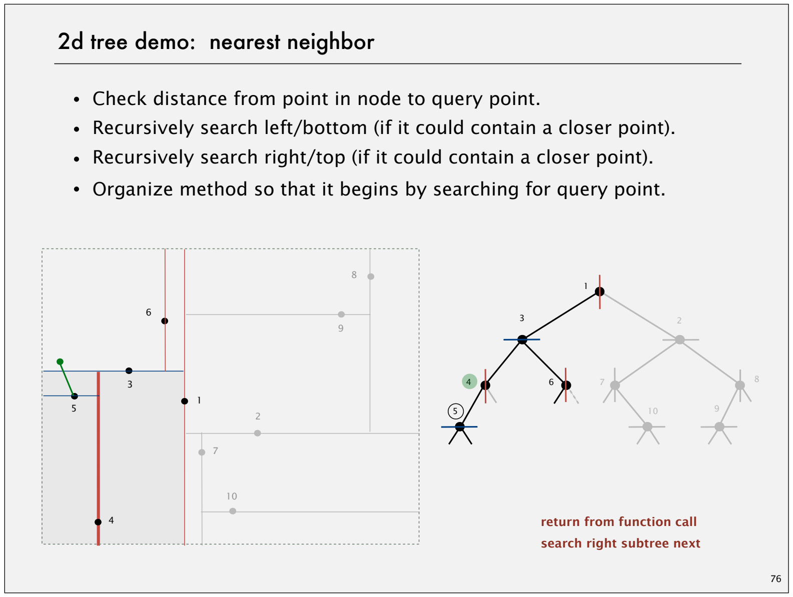
search top subtree
return since empty



2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

search bottom subtree
return since empty



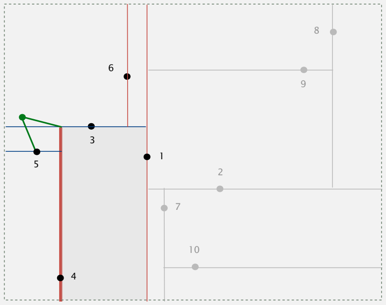
2d tree demo: nearest neighbor

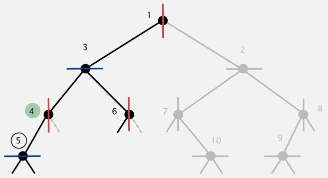
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

return from function call
search right subtree next

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

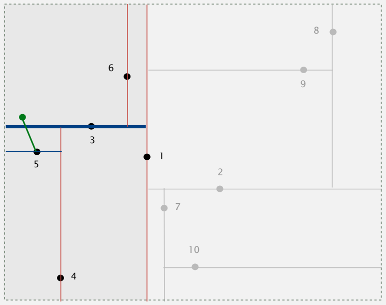


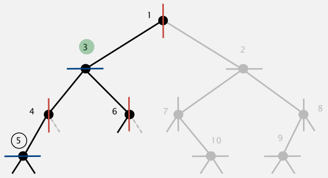


search right subtree
prune since nearest neighbor
can't be here
(drawing not quite to scale)

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

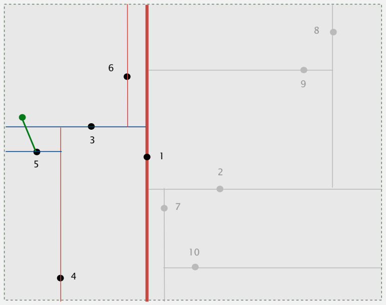


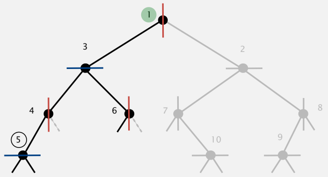


return from function call

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

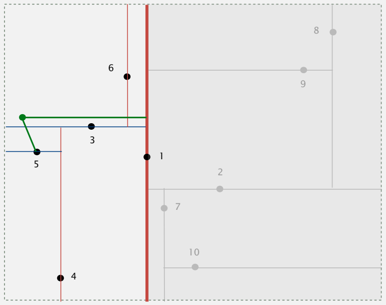


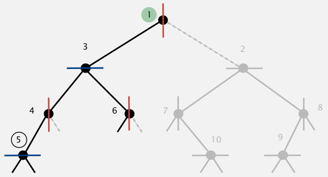


return from function call
search right subtree next

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

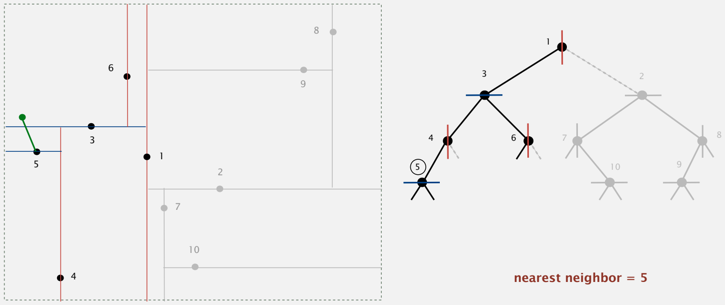




search right subtree
prune since nearest neighbor
can't be here

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

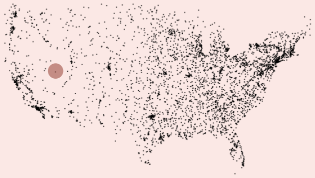


nearest neighbor = 5

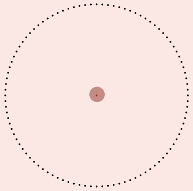
Quiz 4

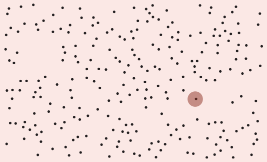
Which of the following is the worst case for nearest neighbor search?

A.



C.



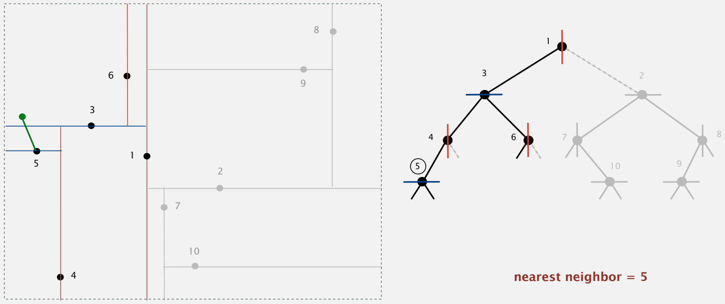


D. *I don't know.*

Nearest neighbor search in a 2d tree analysis

Typical case. $\log N$.

Worst case (even if tree is balanced). N.

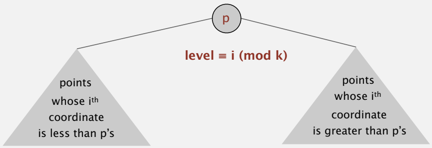


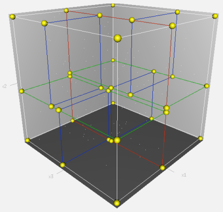
nearest neighbor = 5

Kd tree

Kd tree. Recursively partition k-dimensional space into 2 halfspaces.

Implementation. BST, but cycle through dimensions ala 2d trees.


p
level ≡ i (mod k)
points whose ith coordinate is less than p's
points whose ith coordinate is greater than p's



Efficient, simple data structure for processing k-dimensional data.

- Widely used.
- Adapts well to high-dimensional and clustered data.
- Discovered by an undergrad in an algorithms class!

Jon Bentley

Flocking birds

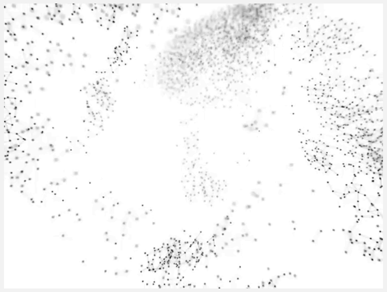
Q. What "natural algorithm" do starlings, migrating geese, starlings, cranes, bait balls of fish, and flashing fireflies use to flock?

http://www.youtube.com/watch?v=XH-groCeKbE

Flocking boids [Craig Reynolds, 1986]

Boids. Three simple rules lead to complex emergent flocking behavior:

- Collision avoidance: point away from k nearest boids.
- Flock centering: point towards the center of mass of k nearest boids.
- Velocity matching: update velocity to the average of k nearest boids.

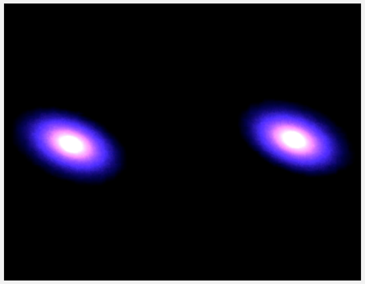


N-body simulation

Goal. Simulate the motion of N particles, mutually affected by gravity.

Brute force. For each pair of particles, compute force: $F = \frac{G\, m_1\, m_2}{r^2}$

Running time. Time per step is N^2.



http://www.youtube.com/watch?v=ua7YlN4eL_w

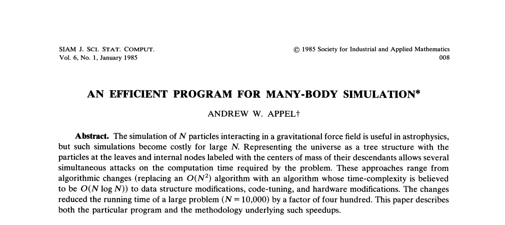
Appel's algorithm for N-body simulation

Key idea. Suppose particle is far, far away from cluster of particles.

- Treat cluster of particles as a single aggregate particle.
- Compute force between particle and center of mass of aggregate.

Appel's algorithm for N-body simulation

- Build 3d-tree with N particles as nodes.
- Store center-of-mass of subtree in each node.
- To compute total force acting on a particle, traverse tree, but stop as soon as distance from particle to subdivision is sufficiently large.



SIAM J. SCI. STAT. COMPUT.
Vol. 6, No. 1, January 1985

© 1985 Society for Industrial and Applied Mathematics
008

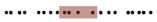
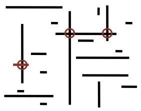
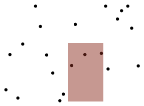
AN EFFICIENT PROGRAM FOR MANY-BODY SIMULATION*

ANDREW W. APPEL†

Abstract. The simulation of N particles interacting in a gravitational force field is useful in astrophysics, but such simulations become costly for large N. Representing the universe as a tree structure with the particles at the leaves and internal nodes labeled with the centers of mass of their descendants allows several simultaneous attacks on the computation time required by the problem. These approaches range from algorithmic changes (replacing an $O(N^2)$ algorithm with an algorithm whose time-complexity is believed to be $O(N \log N)$) to data structure modifications, code-tuning, and hardware modifications. The changes reduced the running time of a large problem ($N = 10{,}000$) by a factor of four hundred. This paper describes both the particular program and the methodology underlying such speedups.

Impact. Running time per step is $N \log N \Rightarrow$ enables new research.

Geometric applications of BSTs

problem	example	solution
1d range search		*binary search tree*
2d orthogonal line segment intersection		*sweep line reduces problem to 1d range search*
2d range search **kd range search**		*2d tree* *kd tree*