http://algs4.cs.princeton.edu

3.5 SYMBOL TABLE APPLICATIONS

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

Set API

Mathematical set. A collection of distinct keys.

public class SET<Key extends Comparable<Key>>		
	SET()	*create an empty set*
void	add(Key key)	*add the key to the set*
boolean	contains(Key key)	*is the key in the set?*
void	remove(Key key)	*remove the key from the set*
int	size()	*return the number of keys in the set*
Iterator<Key>	iterator()	*iterator through keys in the set*

Q. How to implement?

Exception filter

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

```
% more list.txt
was it the of

% java WhiteList list.txt < tinyTale.txt
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of

% java BlackList list.txt < tinyTale.txt
best times worst times
age wisdom age foolishness
epoch belief epoch incredulity
season light season darkness
spring hope winter despair
```

list of exceptional words

Exception filter applications

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards

Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are in the list.

```
public class WhiteList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();

      In in = new In(args[0]);
      while (!in.isEmpty())
         set.add(in.readString());

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (set.contains(word))
            StdOut.println(word);
      }
   }
}
```

create empty set of strings

read in whitelist

print words not in list

Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are not in the list.

```
public class BlackList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();

      In in = new In(args[0]);
      while (!in.isEmpty())
         set.add(in.readString());

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (!set.contains(word))
            StdOut.println(word);
      }
   }
}
```

create empty set of strings

read in whitelist

print words not in list

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

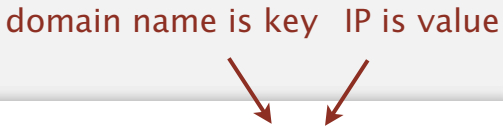
- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

Dictionary lookup

Command-line arguments.

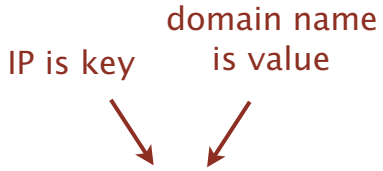
- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 1. DNS lookup.




```
% java LookupCSV ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found

% java LookupCSV ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
```




```
% more ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
www.yale.edu,130.132.51.8
www.econ.yale.edu,128.36.236.74
www.cs.yale.edu,128.36.229.30
espn.com,199.181.135.201
yahoo.com,66.94.234.13
msn.com,207.68.172.246
google.com,64.233.167.99
baidu.com,202.108.22.33
yahoo.co.jp,202.93.91.141
sina.com.cn,202.108.33.32
ebay.com,66.135.192.87
adobe.com,192.150.18.60
163.com,220.181.29.154
passport.net,65.54.179.226
tom.com,61.135.158.237
nate.com,203.226.253.11
cnn.com,64.236.16.20
daum.net,211.115.77.211
blogger.com,66.102.15.100
fastclick.com,205.180.86.4
wikipedia.org,66.230.200.100
rakuten.co.jp,202.72.51.22
...
```

Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 2. Amino acids.

codon is key name is value

```
% java LookupCSV amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

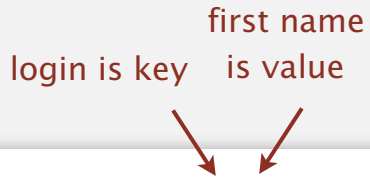
```
% more amino.csv
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
...
```

Dictionary lookup

Command-line arguments.

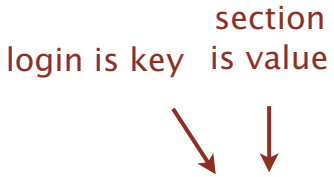
- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 3. Class list.




```
% java LookupCSV classlist.csv 4 1
eberl
Ethan
nwebb
Natalie

% java LookupCSV classlist.csv 4 3
dpan
P01
```




```
% more classlist.csv
13,Berl,Ethan Michael,P01,eberl
12,Cao,Phillips Minghua,P01,pcao
11,Chehoud,Christel,P01,cchehoud
10,Douglas,Malia Morioka,P01,malia
12,Haddock,Sara Lynn,P01,shaddock
12,Hantman,Nicole Samantha,P01,nhantman
11,Hesterberg,Adam Classen,P01,ahesterb
13,Hwang,Roland Lee,P01,rhwang
13,Hyde,Gregory Thomas,P01,ghyde
13,Kim,Hyunmoon,P01,hktwo
12,Korac,Damjan,P01,dkorac
11,MacDonald,Graham David,P01,gmacdona
10,Michal,Brian Thomas,P01,bmichal
12,Nam,Seung Hyeon,P01,seungnam
11,Nastasescu,Maria Monica,P01,mnastase
11,Pan,Di,P01,dpan
12,Partridge,Brenton Alan,P01,bpartrid
13,Rilee,Alexander,P01,arilee
13,Roopakalu,Ajay,P01,aroopaka
11,Sheng,Ben C,P01,bsheng
12,Webb,Natalie Sue,P01,nwebb
⋮
```

Dictionary lookup: Java implementation

```
public class LookupCSV
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      int keyField = Integer.parseInt(args[1]);
      int valField = Integer.parseInt(args[2]);

      ST<String, String> st = new ST<String, String>();
      while (!in.isEmpty())
      {
         String line = in.readLine();
         String[] tokens = line.split(",");
         String key = tokens[keyField];
         String val = tokens[valField];
         st.put(key, val);
      }

      while (!StdIn.isEmpty())
      {
         String s = StdIn.readString();
         if (!st.contains(s)) StdOut.println("Not found");
         else                 StdOut.println(st.get(s));
      }
   }
}
```

process input file

build symbol table

process lookups with standard I/O

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

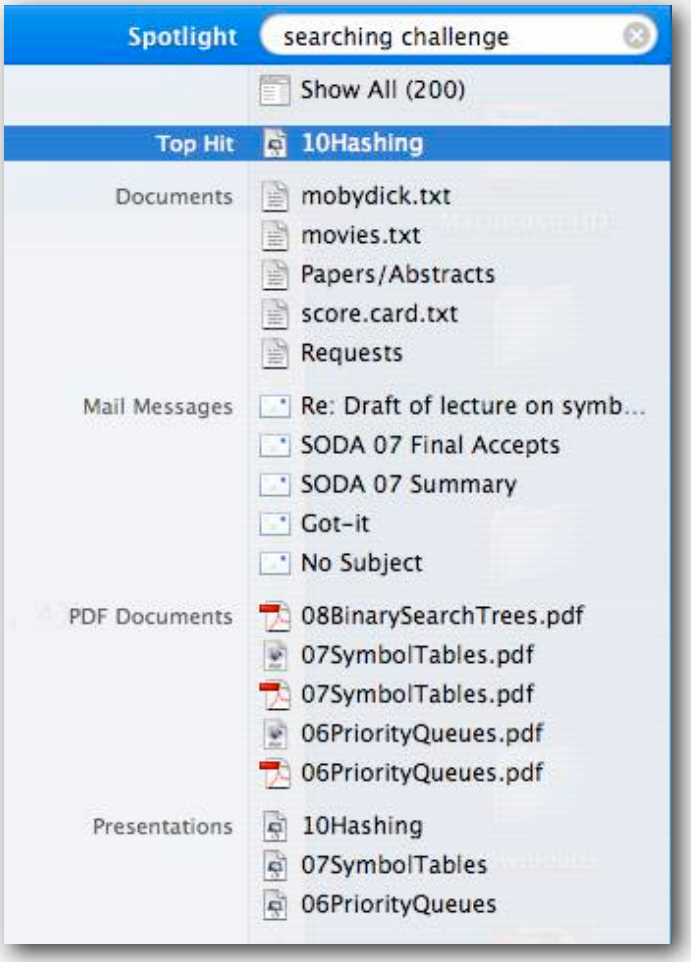
http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

File indexing

Goal. Index a PC (or the web).



File indexing

Goal. Given a list of files specified, create an index so that you can efficiently find all files containing a given query string.

```
% ls *.txt
aesop.txt magna.txt moby.txt
sawyer.txt tale.txt

% java FileIndex *.txt

freedom
magna.txt moby.txt tale.txt

whale
moby.txt

lamb
sawyer.txt aesop.txt
```

```
% ls *.java
BlackList.java Concordance.java
DeDup.java FileIndex.java ST.java
SET.java WhiteList.java

% java FileIndex *.java

import
FileIndex.java SET.java ST.java

Comparator
null
```

Solution. Key = query string; value = set of files containing that string.

File indexing

```
import java.io.File;
public class FileIndex
{
   public static void main(String[] args)
   {
      ST<String, SET<File>> st = new ST<String, SET<File>>();

      for (String filename : args) {
         File file = new File(filename);
         In in = new In(file);
         while (!in.isEmpty())
         {
            String key = in.readString();
            if (!st.contains(key))
               st.put(key, new SET<File>());
            SET<File> set = st.get(key);
            set.add(file);
         }
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         StdOut.println(st.get(query));
      }
   }
}
```

symbol table

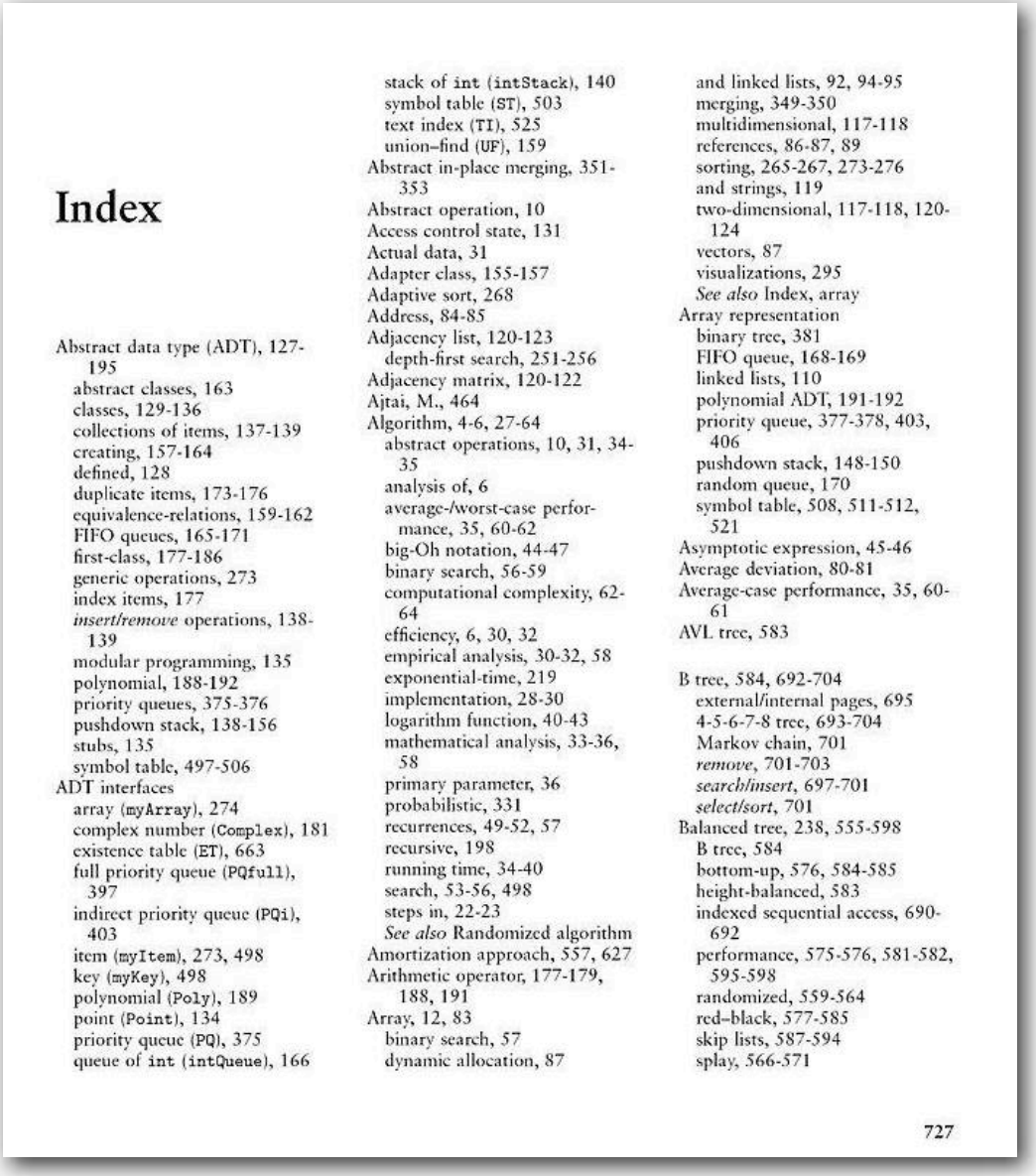
list of file names from command line

for each word in file, add file to corresponding set

process queries

Book index

Goal. Index for an e-book.



Index

Concordance

Goal. Preprocess a text corpus to support concordance queries: given a word, find all occurrences with their immediate contexts.

```
% java Concordance tale.txt
cities
tongues of the two *cities* that were blended in

majesty
their turnkeys and the *majesty* of the law fired
me treason against the *majesty* of the people in
of his most gracious *majesty* king george the third

princeton
no matches
```

Concordance

```
public class Concordance
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      String[] words = in.readAllStrings();
      ST<String, SET<Integer>> st = new ST<String, SET<Integer>>();
      for (int i = 0; i < words.length; i++)
      {
         String s = words[i];
         if (!st.contains(s))
            st.put(s, new SET<Integer>());
         SET<Integer> set = st.get(s);
         set.add(i);
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         SET<Integer> set = st.get(query);
         for (int k : set)
             // print words[k-4] to words[k+4]
      }
   }
}
```

read text and build index

process queries and print concordances

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

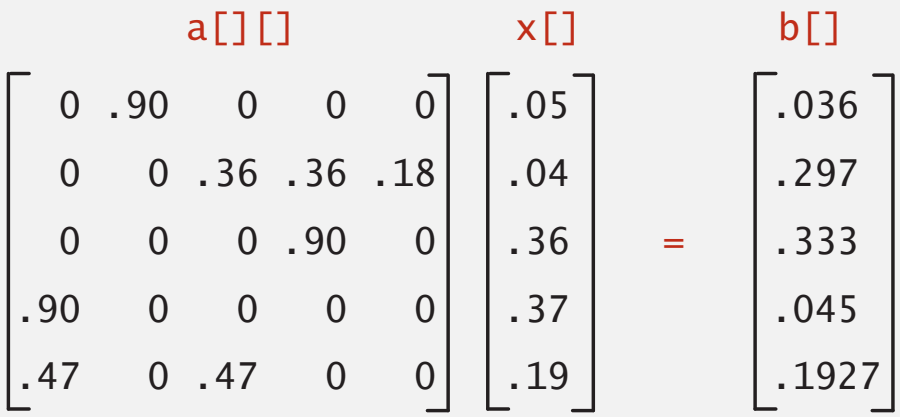
- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

Matrix-vector multiplication (standard implementation)



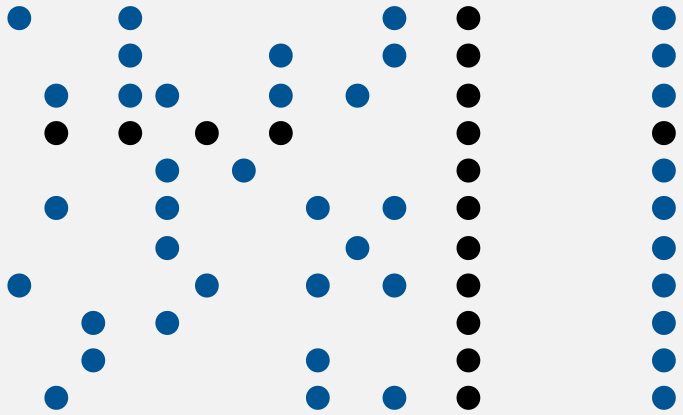

```
...
double[][] a = new double[N][N];
double[] x = new double[N];
double[] b = new double[N];
...
// initialize a[][] and x[]
...
for (int i = 0; i < N; i++)
{
   sum = 0.0;
   for (int j = 0; j < N; j++)
      sum += a[i][j]*x[j];
   b[i] = sum;
}
```

nested loops
(N^2 running time)

Sparse matrix-vector multiplication

Problem. Sparse matrix-vector multiplication.

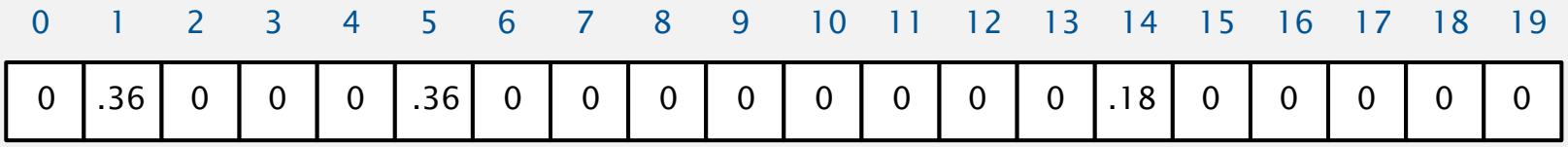
Assumptions. Matrix dimension is 10,000; average nonzeros per row ~ 10.



Vector representations

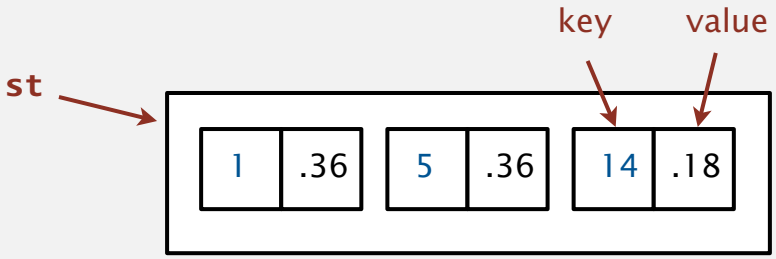
1d array (standard) representation.

- Constant time access to elements.
- Space proportional to N.



Symbol table representation.

- Key = index, value = entry.
- Efficient iterator.
- Space proportional to number of nonzeros.



Sparse vector data type

```
public class SparseVector
{
   private HashST<Integer, Double> v;

   public SparseVector()
   {  v = new HashST<Integer, Double>();  }

   public void put(int i, double x)
   {  v.put(i, x);  }

   public double get(int i)
   {
      if (!v.contains(i)) return 0.0;
      else return v.get(i);
   }

   public Iterable<Integer> indices()
   {  return v.keys();  }

   public double dot(double[] that)
   {
       double sum = 0.0;
       for (int i : indices())
           sum += that[i]*this.get(i);
       return sum;
   }
}
```

HashST because order not important

empty ST represents all 0s vector

a[i] = value

return a[i]

dot product is constant time for sparse vectors

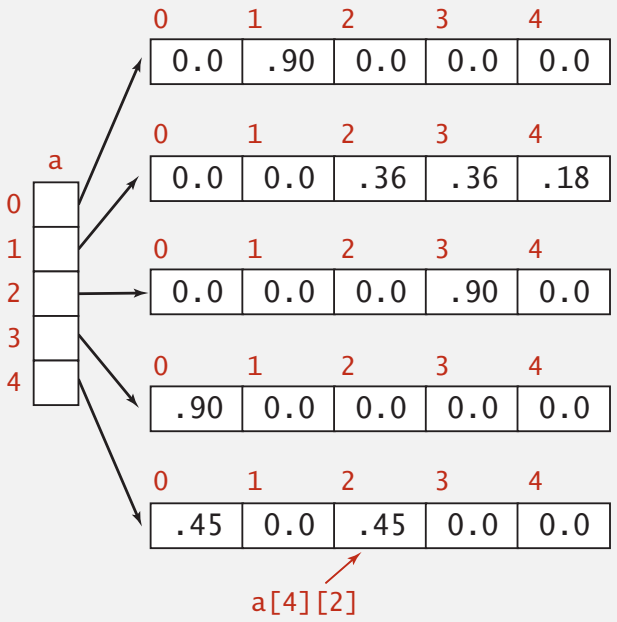
Matrix representations

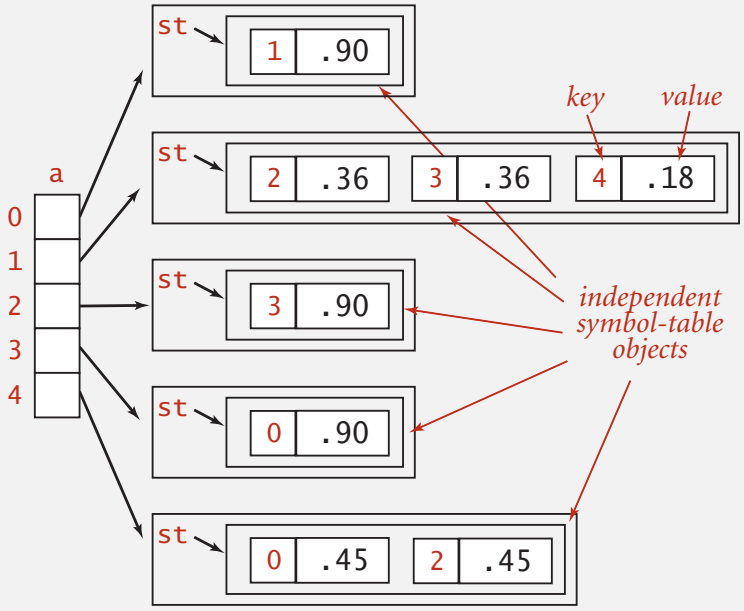
2D array (standard) matrix representation: Each row of matrix is an array.

- Constant time access to elements.
- Space proportional to N^2.

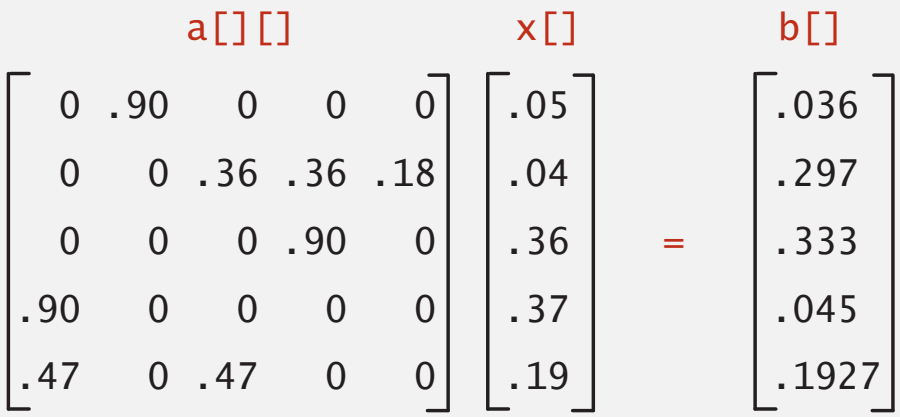
Sparse matrix representation: Each row of matrix is a sparse vector.

- Efficient access to elements.
- Space proportional to number of nonzeros (plus N).





Sparse matrix-vector multiplication




```
..
SparseVector[] a = new SparseVector[N];
double[] x = new double[N];
double[] b = new double[N];
...
// Initialize a[] and x[]
...
for (int i = 0; i < N; i++)
   b[i] = a[i].dot(x);
```

linear running time
for sparse matrix

http://algs4.cs.princeton.edu

3.5 Symbol Table Applications

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

http://algs4.cs.princeton.edu

3.5 SYMBOL TABLE APPLICATIONS

- *sets*
- *dictionary clients*
- *indexing clients*
- *sparse vectors*

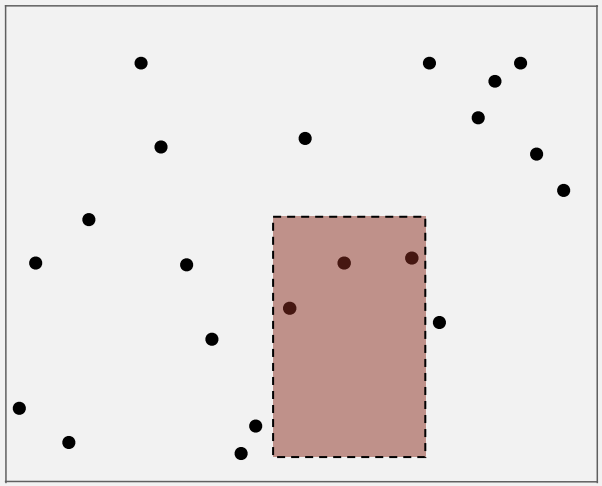
http://algs4.cs.princeton.edu

GEOMETRIC APPLICATIONS OF BSTS

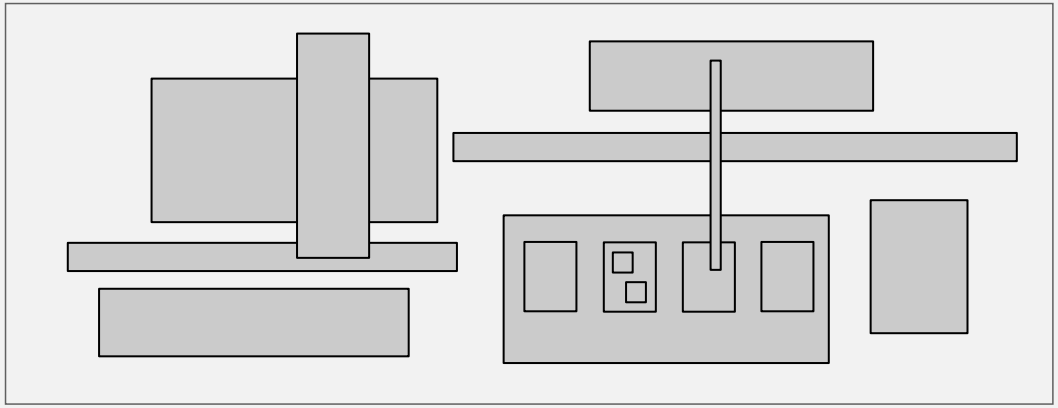
- *1d range search*
- *line segment intersection*
- *kd trees*
- *interval search trees*
- *rectangle intersection*

Overview

This lecture. Intersections among geometric objects.



2d orthogonal range search



orthogonal rectangle intersection

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

Efficient solutions. Binary search trees (and extensions).

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

1d range search

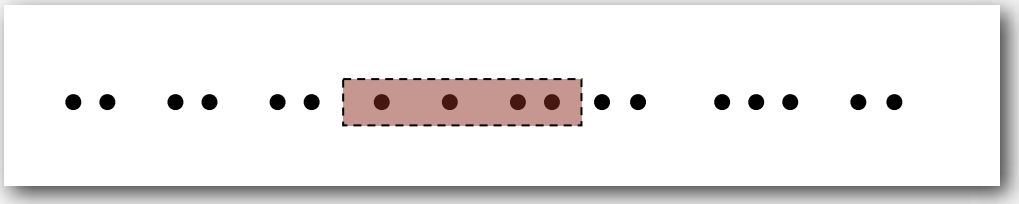
Extension of ordered symbol table.

- 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.

Geometric interpretation.

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



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
count G to K	2
search G to K	H I

1d range search: elementary implementations

Unordered list. Fast insert, slow range search.

Ordered array. Slow insert, binary search for k_1 and k_2 to do range search.

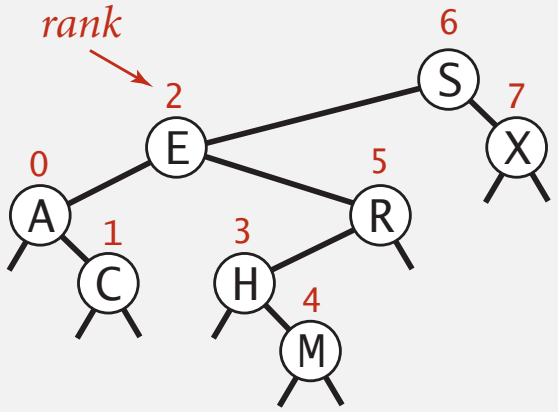
order of growth of running time for 1d range search

data structure	insert	range count	range search
unordered list	1	N	N
ordered array	N	log N	R + log 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` ?




```
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

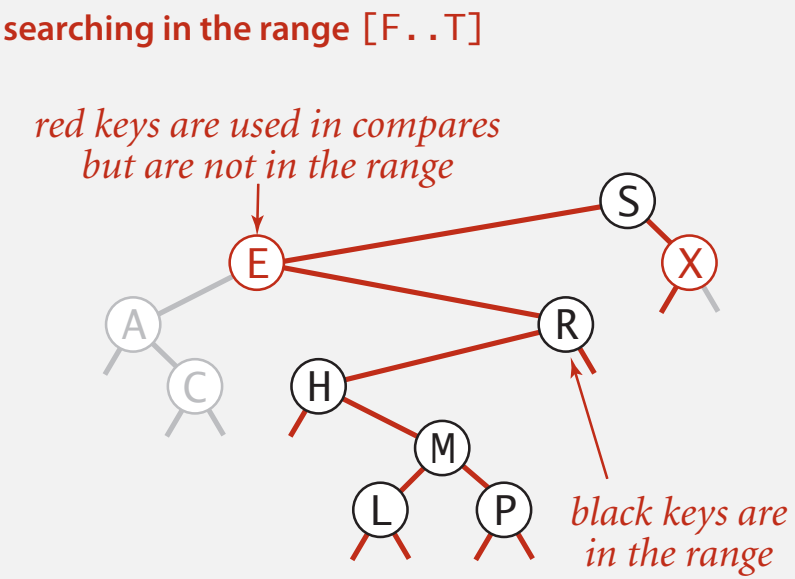
Proposition. Running time proportional to $\log N$.

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).



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

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

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

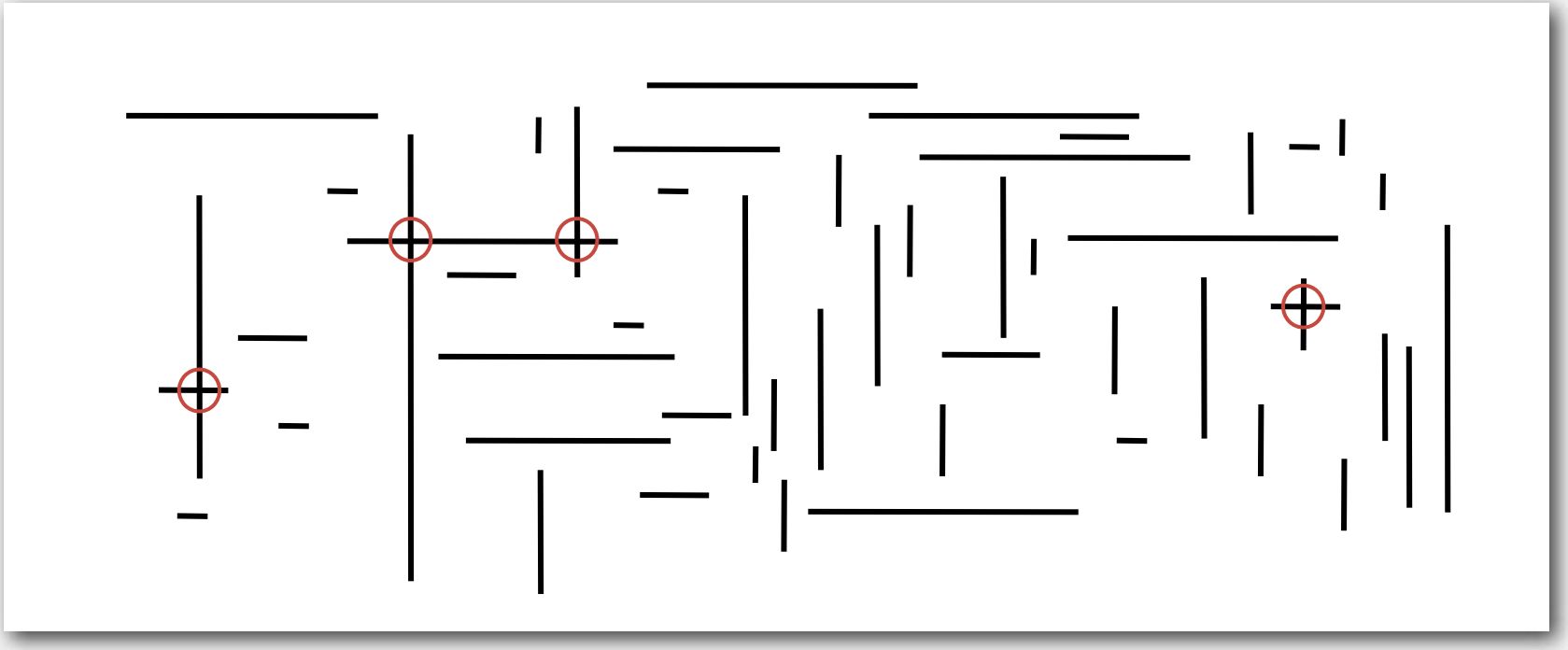
http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

Orthogonal line segment intersection

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



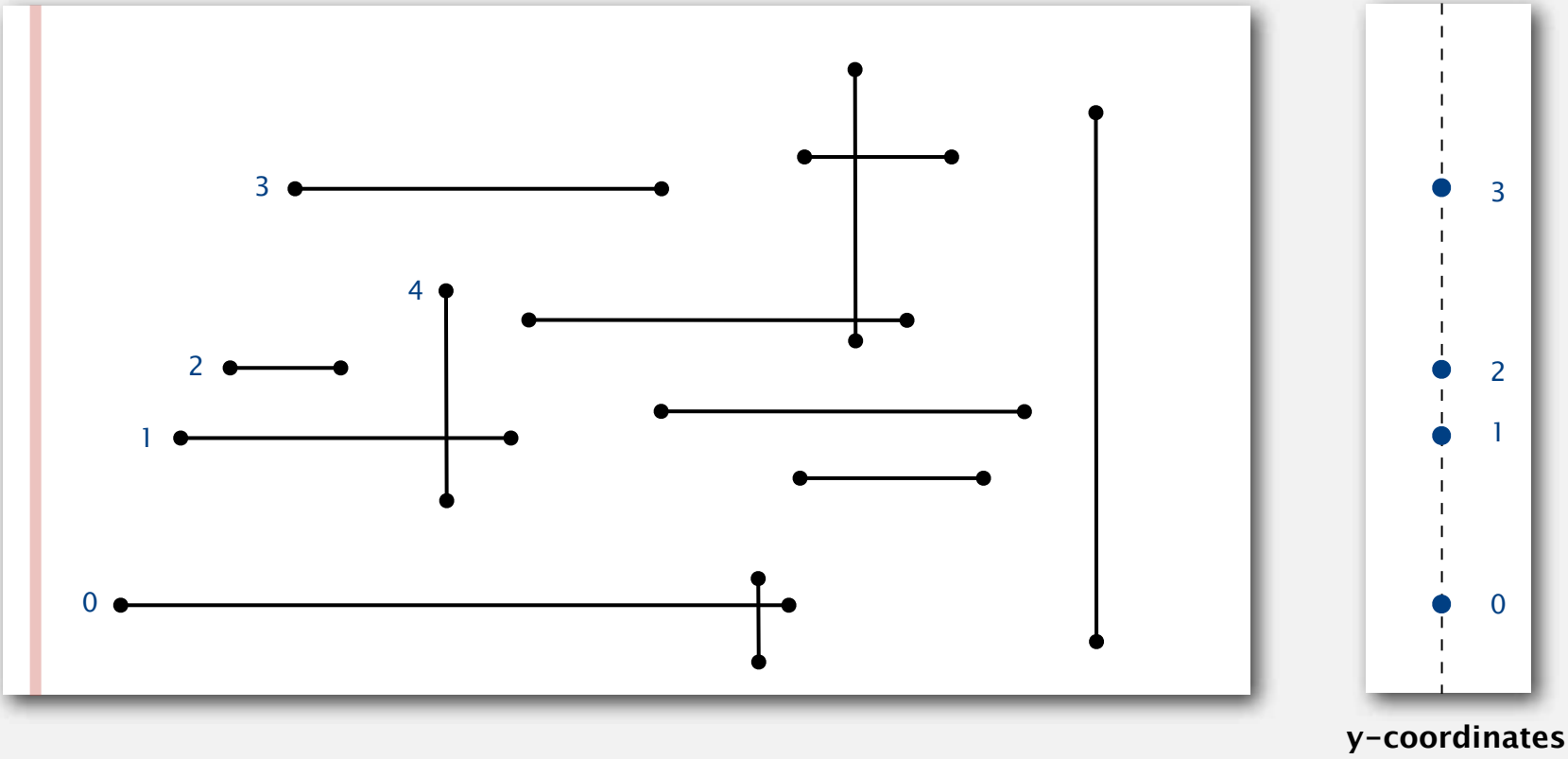
Quadratic algorithm. Check all pairs of line segments for intersection.

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

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

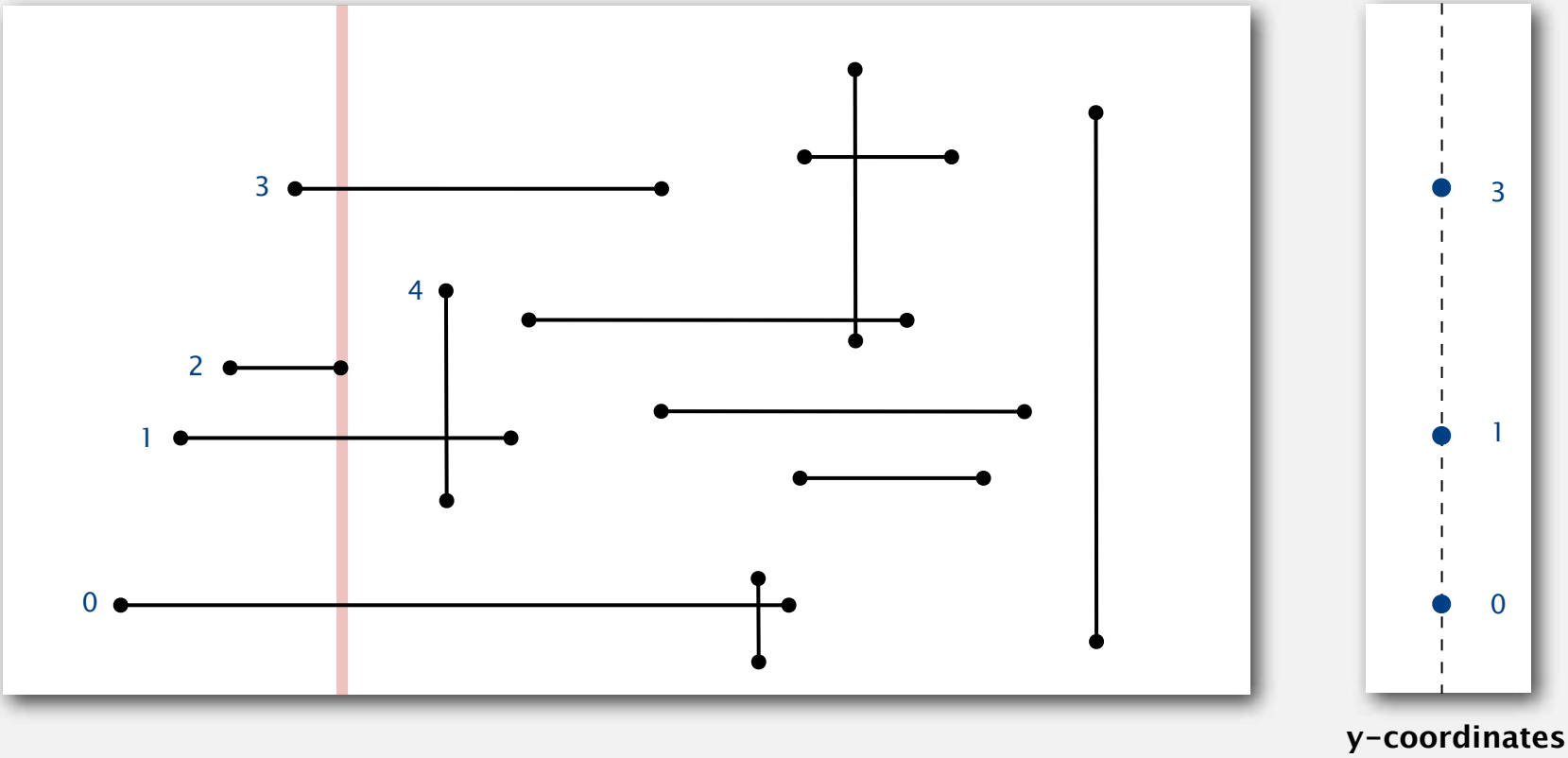
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

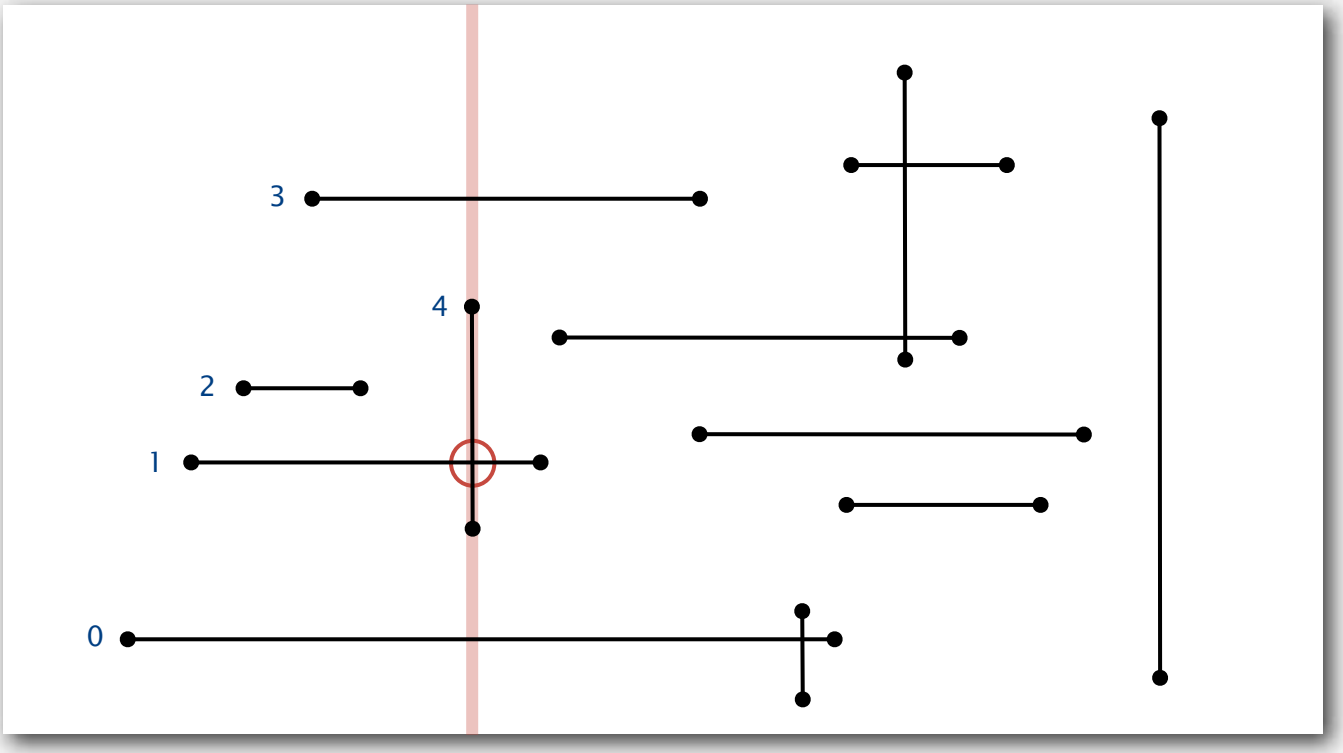
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.
- h-segment (right endpoint): remove y-coordinate from BST.

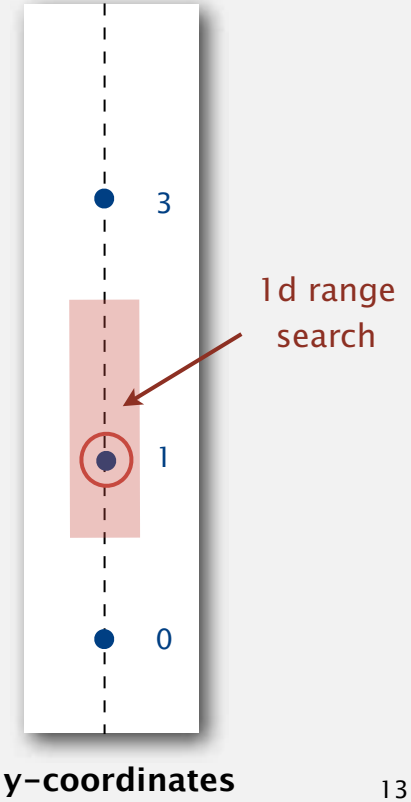


Orthogonal line segment intersection: sweep-line algorithm

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.





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.

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

2-d orthogonal range search

Extension of ordered symbol-table to 2d keys.

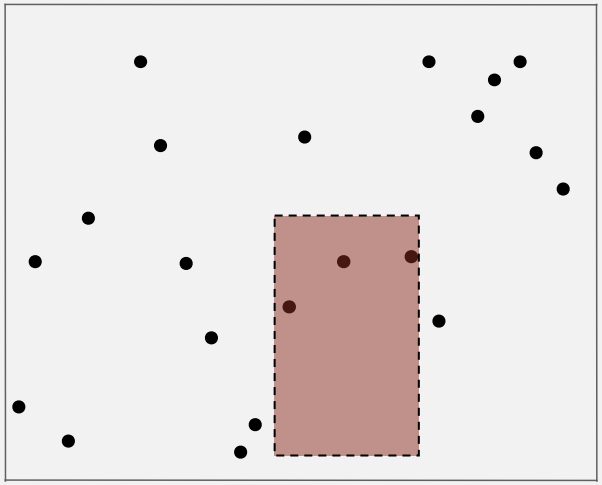
- Insert a 2d key.
- Delete a 2d key.
- Search for a 2d key.
- Range search: find all keys that lie in a 2d range.
- Range count: number of keys that lie in a 2d range.

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

Geometric interpretation.

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

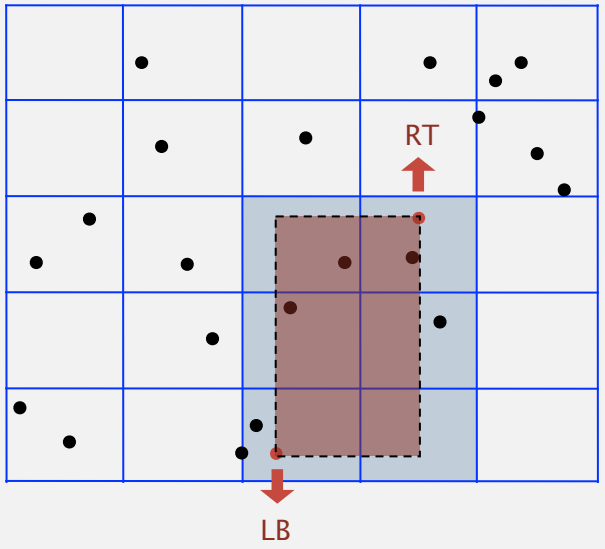
rectangle is axis-aligned



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.

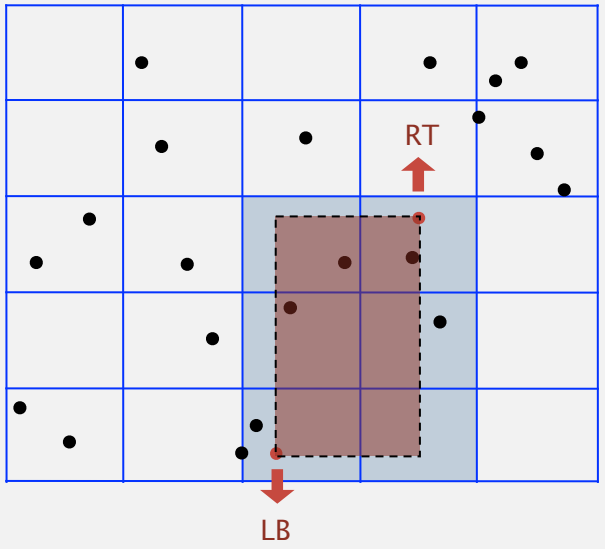
- Space: $M^2 + N$.
- Time: $1 + N / M^2$ per square examined, on average.

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.

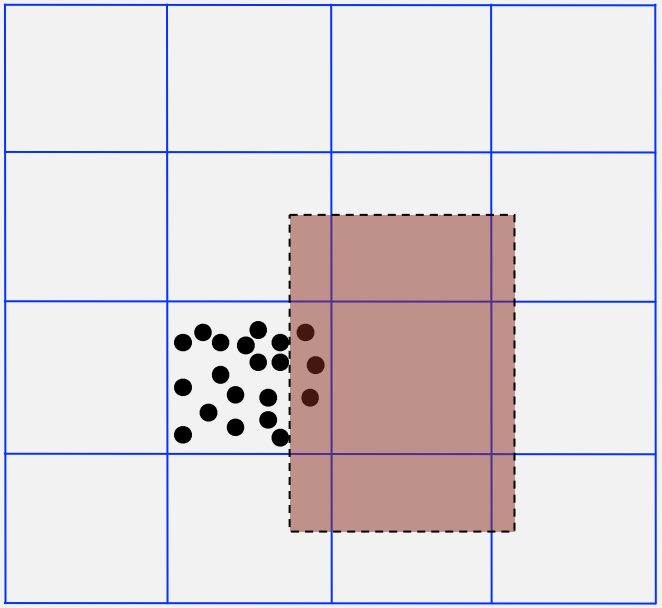


Clustering

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.

13,000 points, 1000 grid squares

half the squares are empty

half the points are
in 10% of the squares

Space-partitioning trees

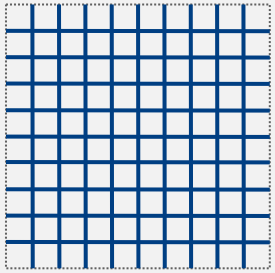
Use a tree to represent a recursive subdivision of 2d space.

Grid. Divide space uniformly into squares.

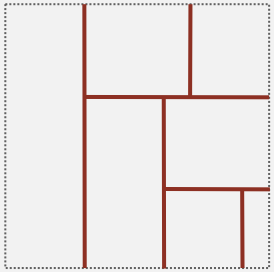
2d tree. Recursively divide space into two halfplanes.

Quadtree. Recursively divide space into four quadrants.

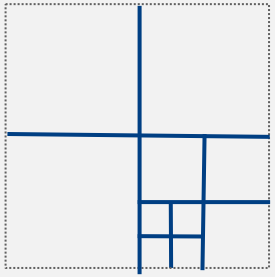
BSP tree. Recursively divide space into two regions.



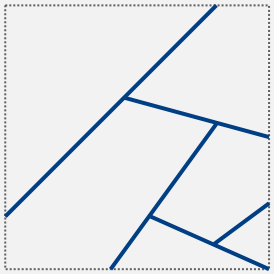
Grid



2d tree



Quadtree

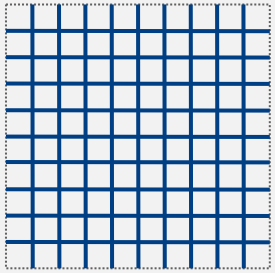


BSP tree

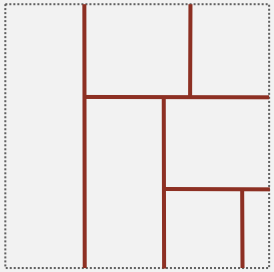
Space-partitioning trees: applications

Applications.

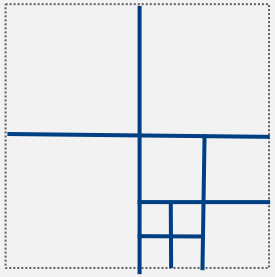
- Ray tracing.
- 2d range search.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Nearest neighbor search.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.



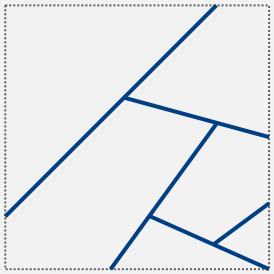
Grid



2d tree



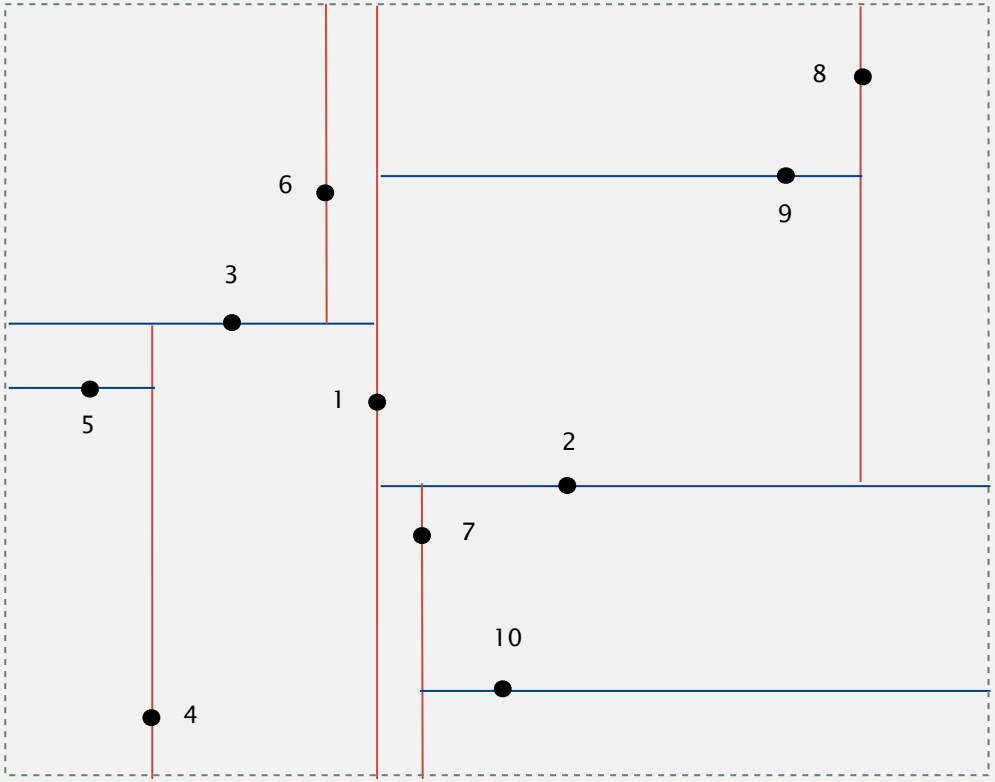
Quadtree

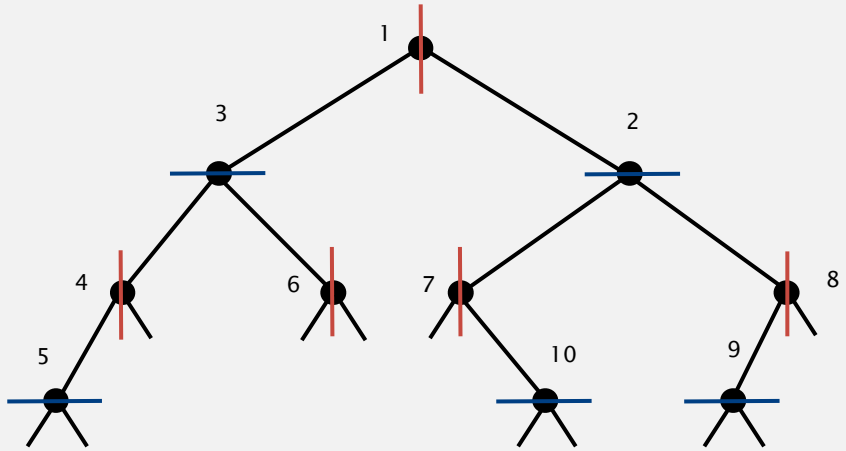


BSP tree

2d tree construction

Recursively partition plane into two halfplanes.

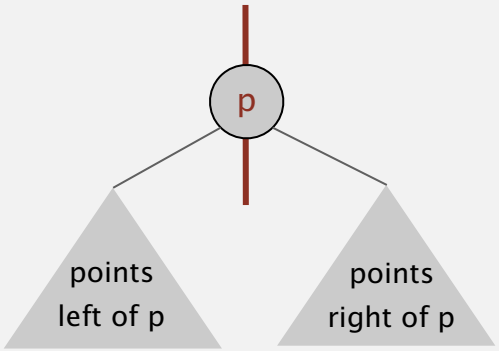


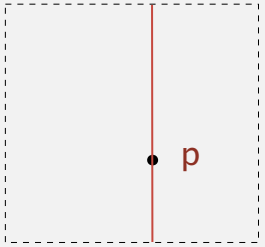


2d tree implementation

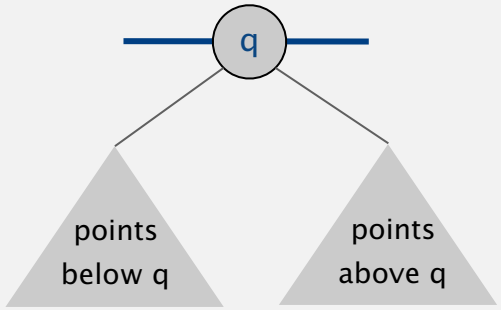
Data structure. BST, but alternate using x- and y-coordinates as key.

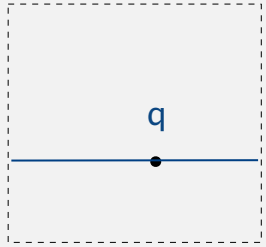
- Search gives rectangle containing point.
- Insert further subdivides the plane.



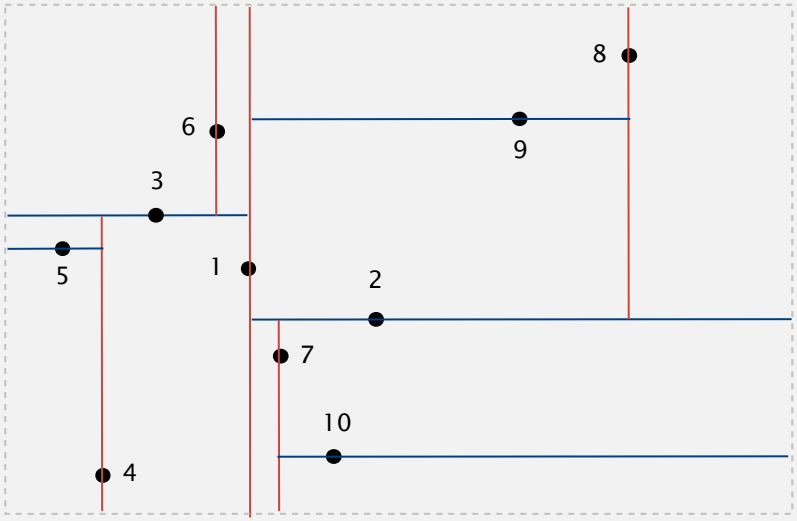


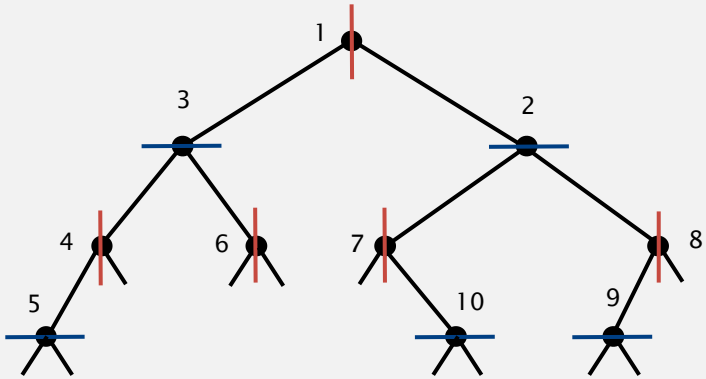
even levels





odd levels

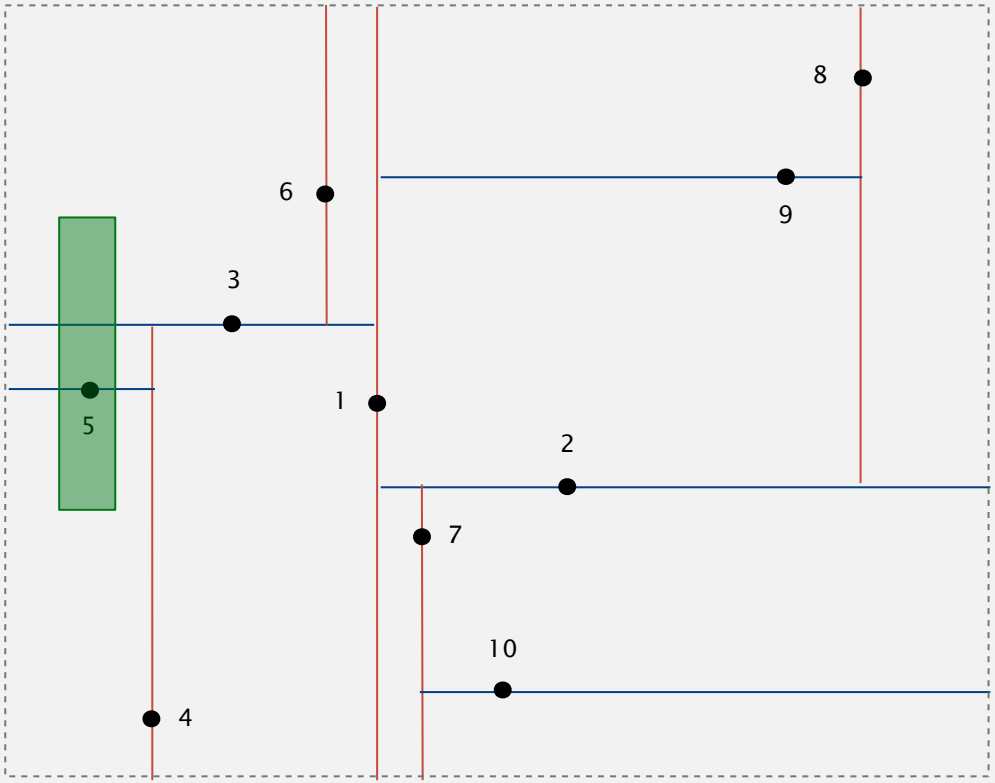


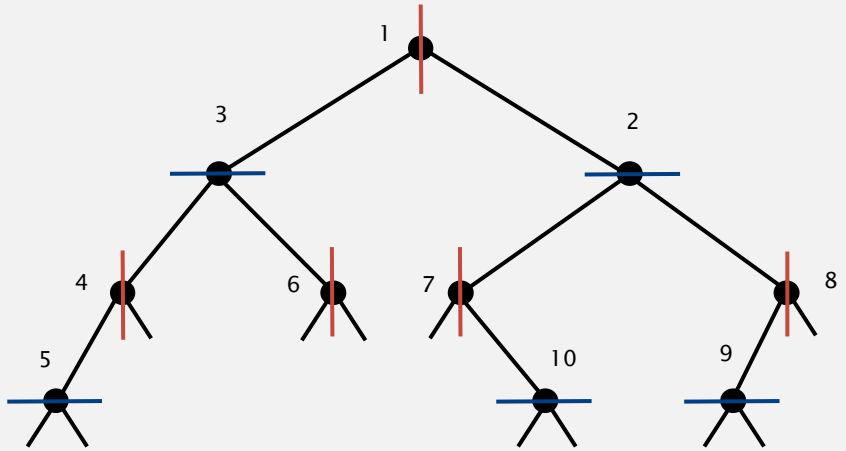


Range search in a 2d tree demo

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).

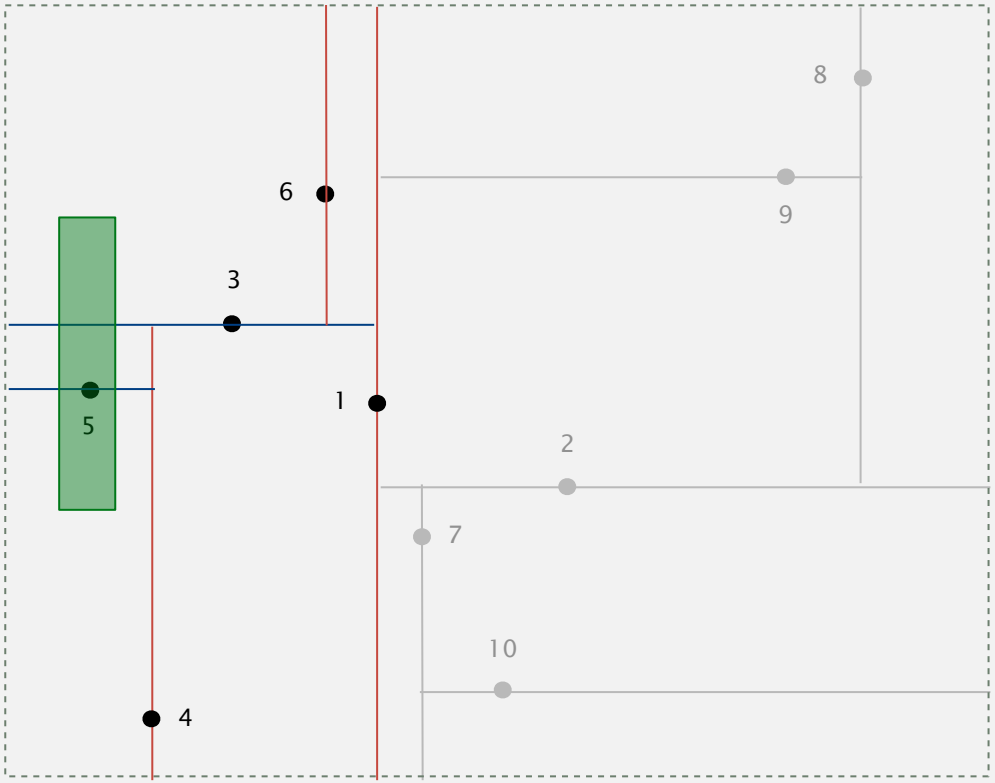


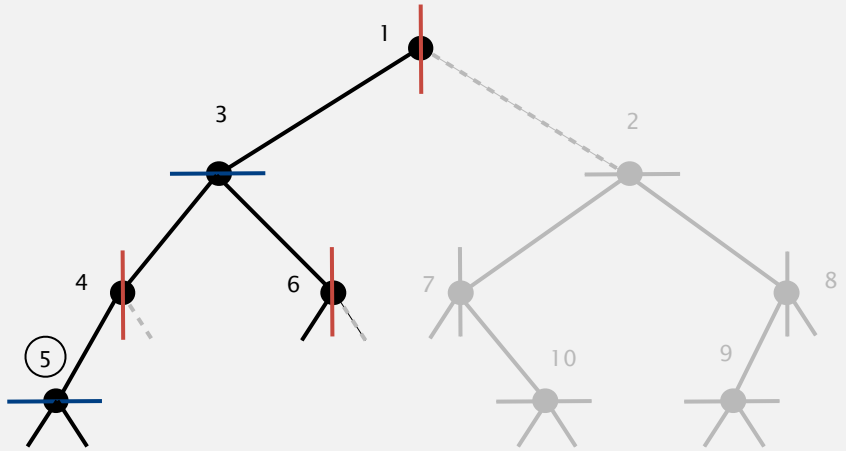


Range search in a 2d tree demo

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).



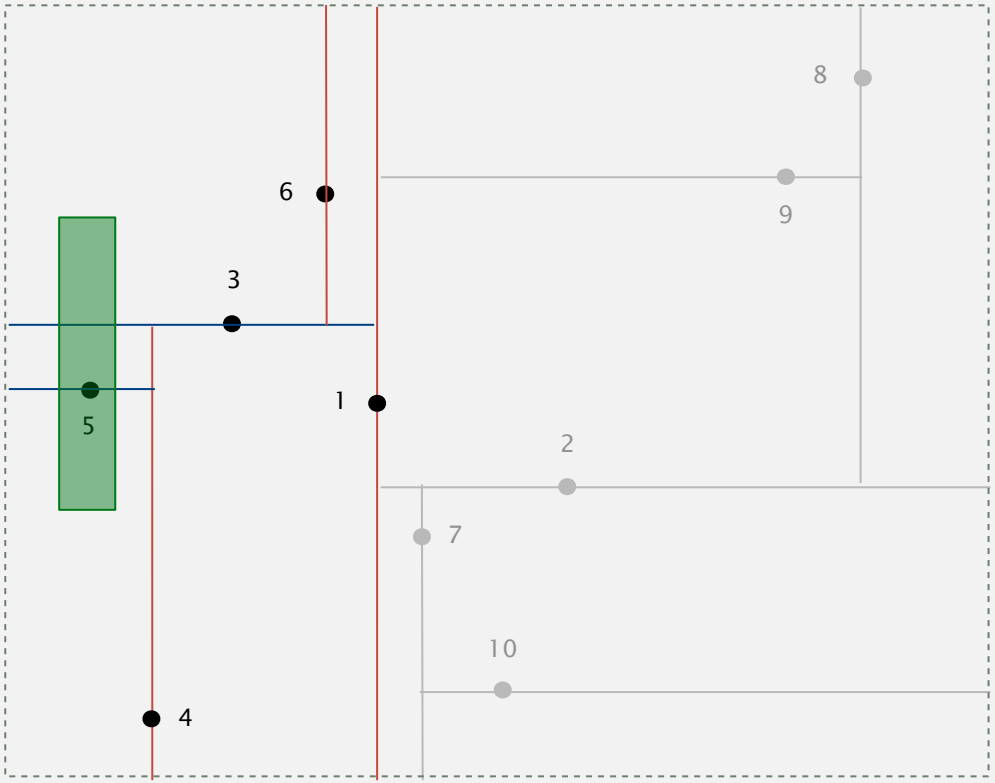


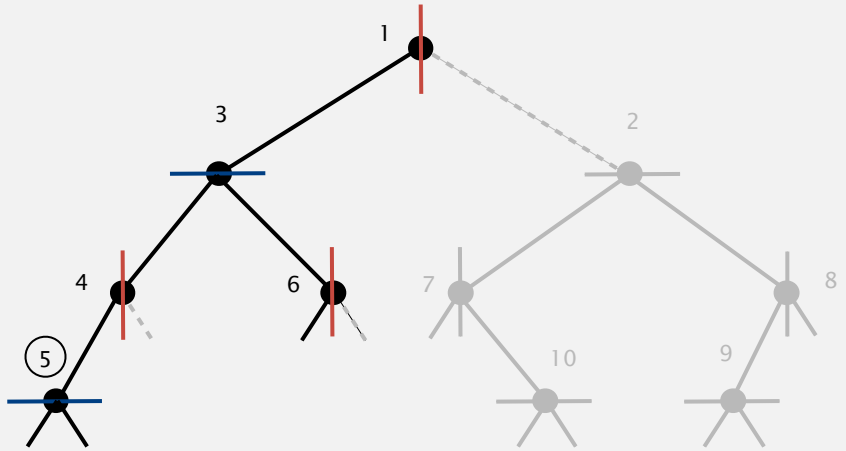
done

Range search in a 2d tree analysis

Typical case. $R + \log N$.

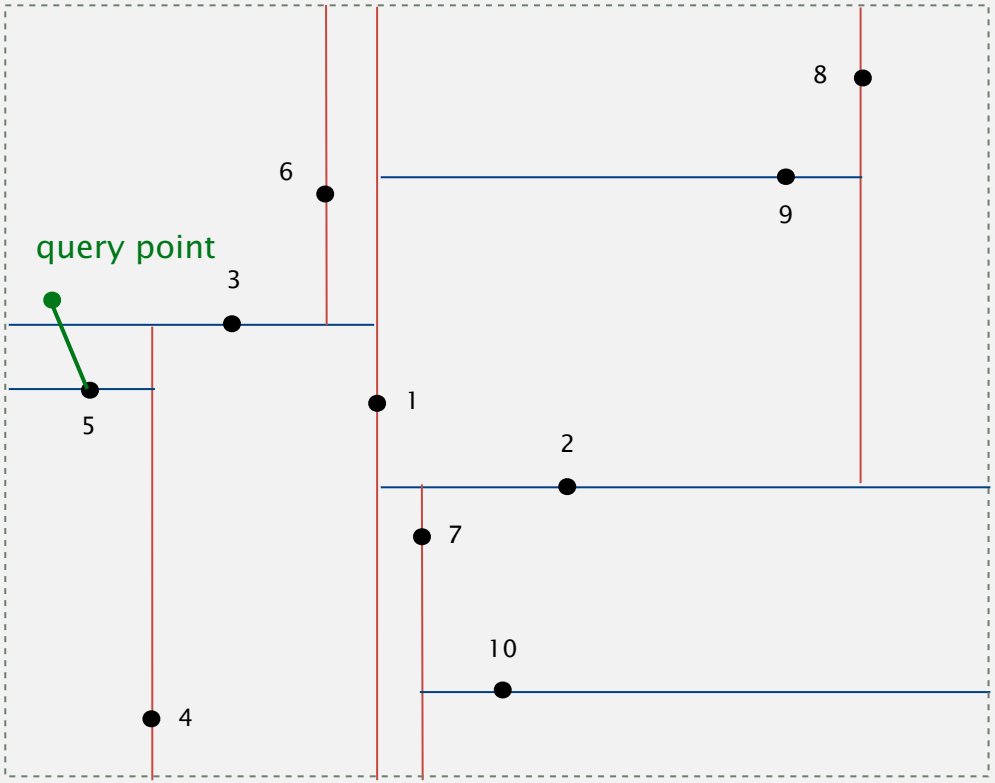
Worst case (assuming tree is balanced). $R + \sqrt{N}$.

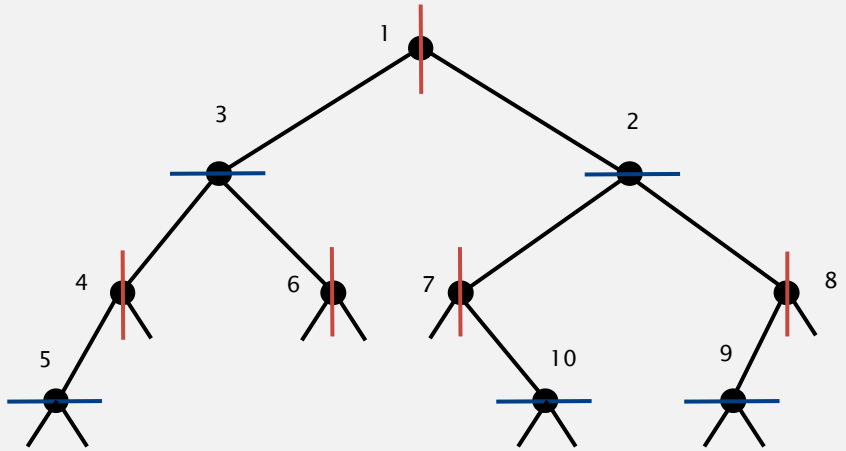




Nearest neighbor search in a 2d tree demo

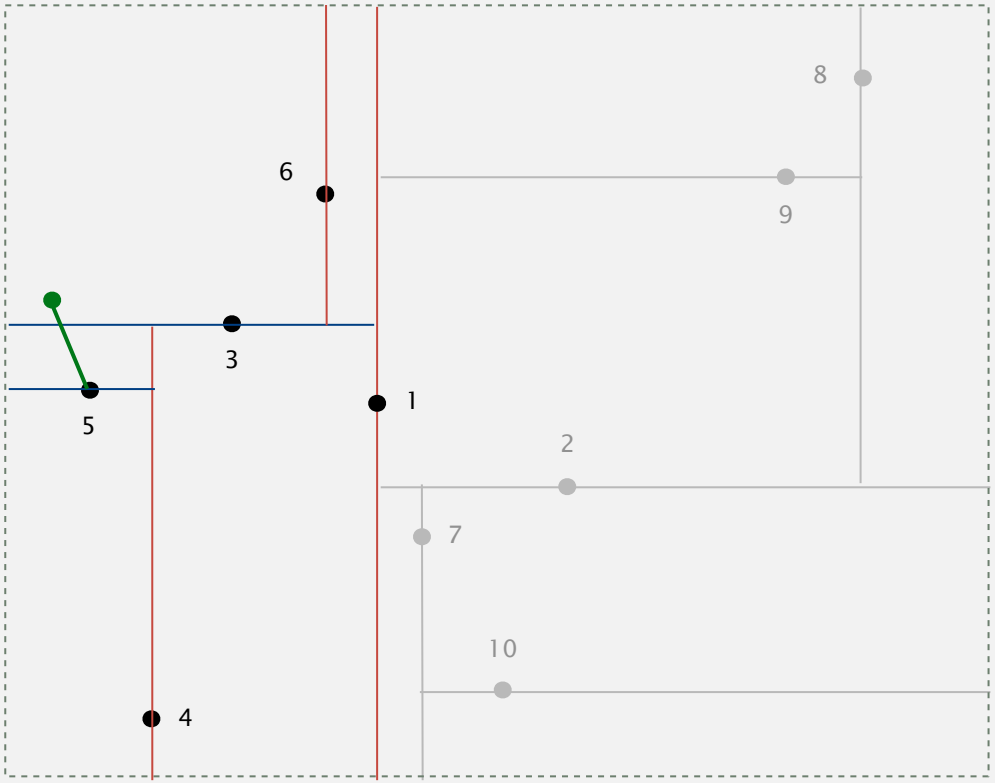
Goal. Find closest point to query point.

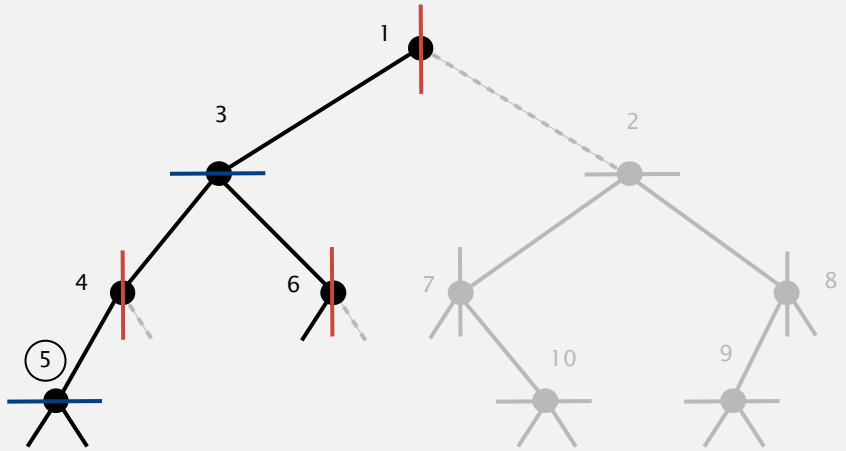




Nearest neighbor search in a 2d tree demo

- 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.



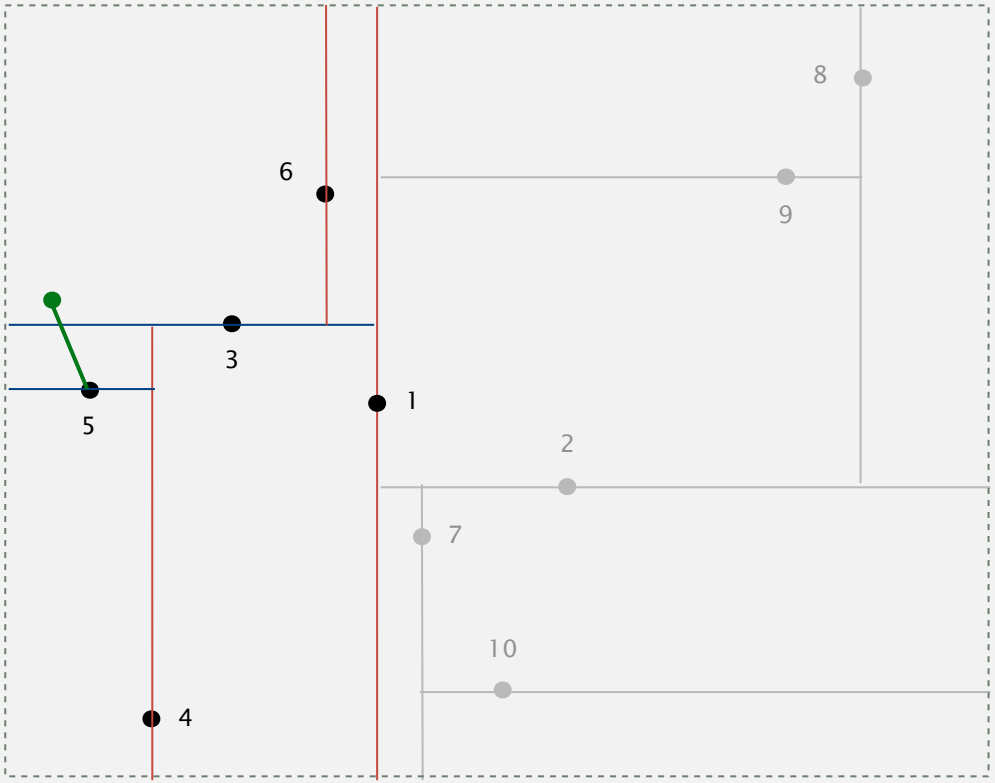


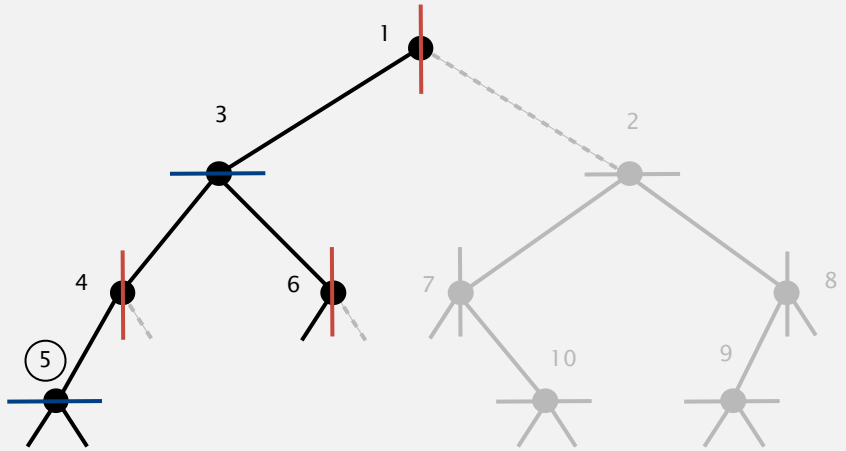
nearest neighbor = 5

Nearest neighbor search in a 2d tree analysis

Typical case. $\log N$.

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





nearest neighbor = 5

Flocking birds

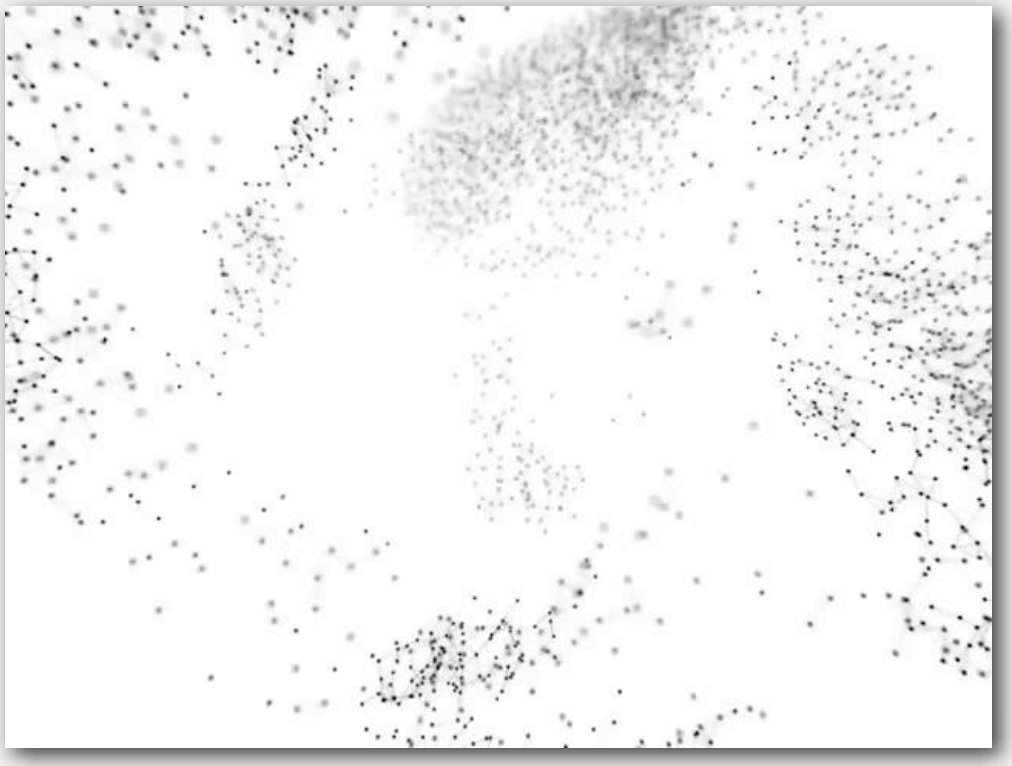
Q. What "natural algorithm" do starlings, migrating geese, starlings, cranes, bait balls of fish, and flashing fireflies use to flock?

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

Flocking boids [Craig Reynolds, 1986]

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

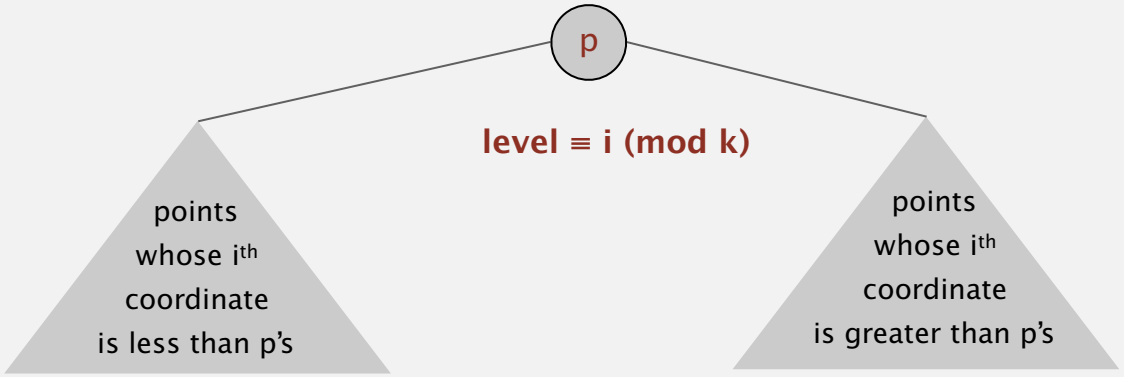
- Collision avoidance: point away from k nearest boids.
- Flock centering: point towards the center of mass of k nearest boids.
- Velocity matching: update velocity to the average of k nearest boids.

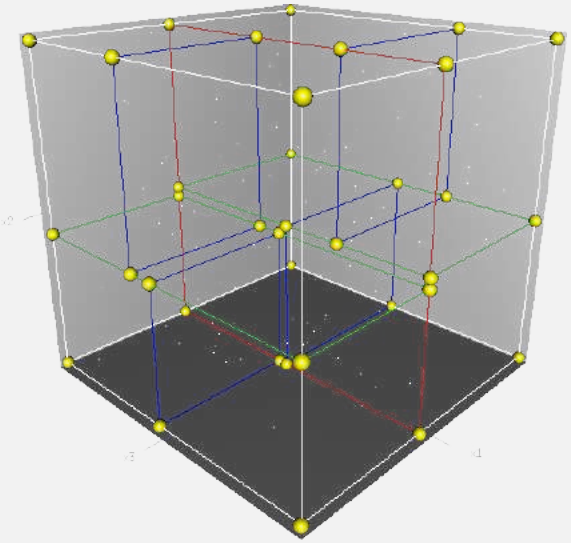


Kd tree

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

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





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!

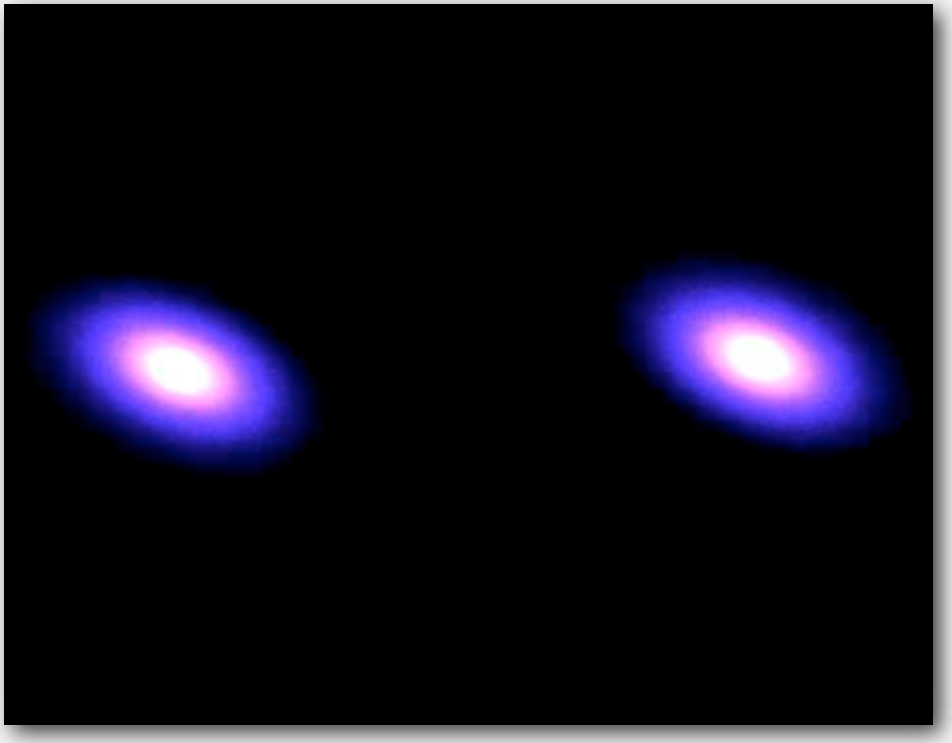
Jon Bentley

N-body simulation

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

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

Running time. Time per step is N^2.

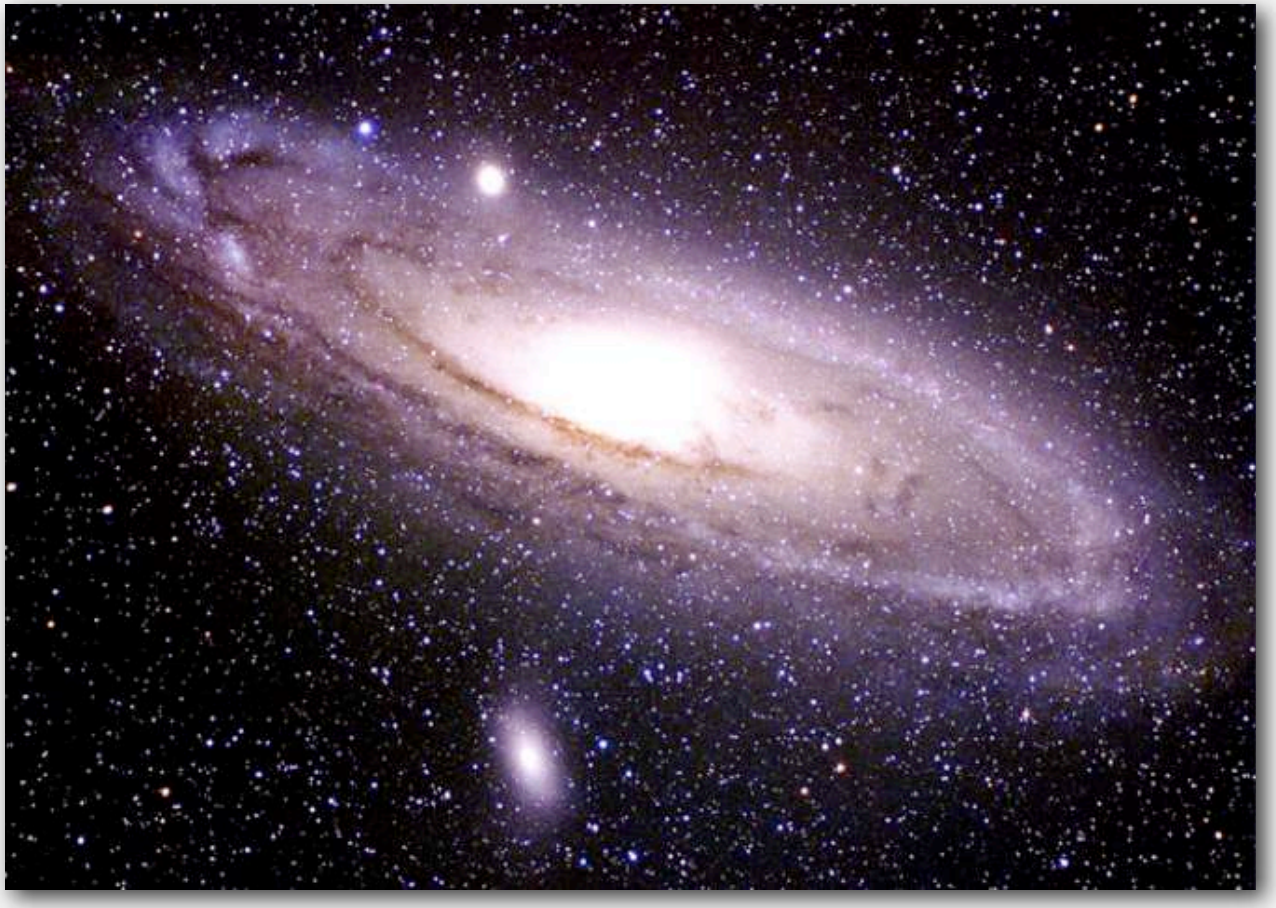


http://www.youtube.com/watch?v=ua7YlN4eL_w

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

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.

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

1d interval search

1d interval search. Data structure to hold set of (overlapping) intervals.

- Insert an interval (lo, hi).
- Search for an interval (lo, hi).
- Delete an interval (lo, hi).
- Interval intersection query: given an interval (lo, hi), find all intervals (or one interval) in data structure that intersects (lo, hi).

Q. Which intervals intersect $(9, 16)$?

A. $(7, 10)$ and $(15, 18)$.

1d interval search API

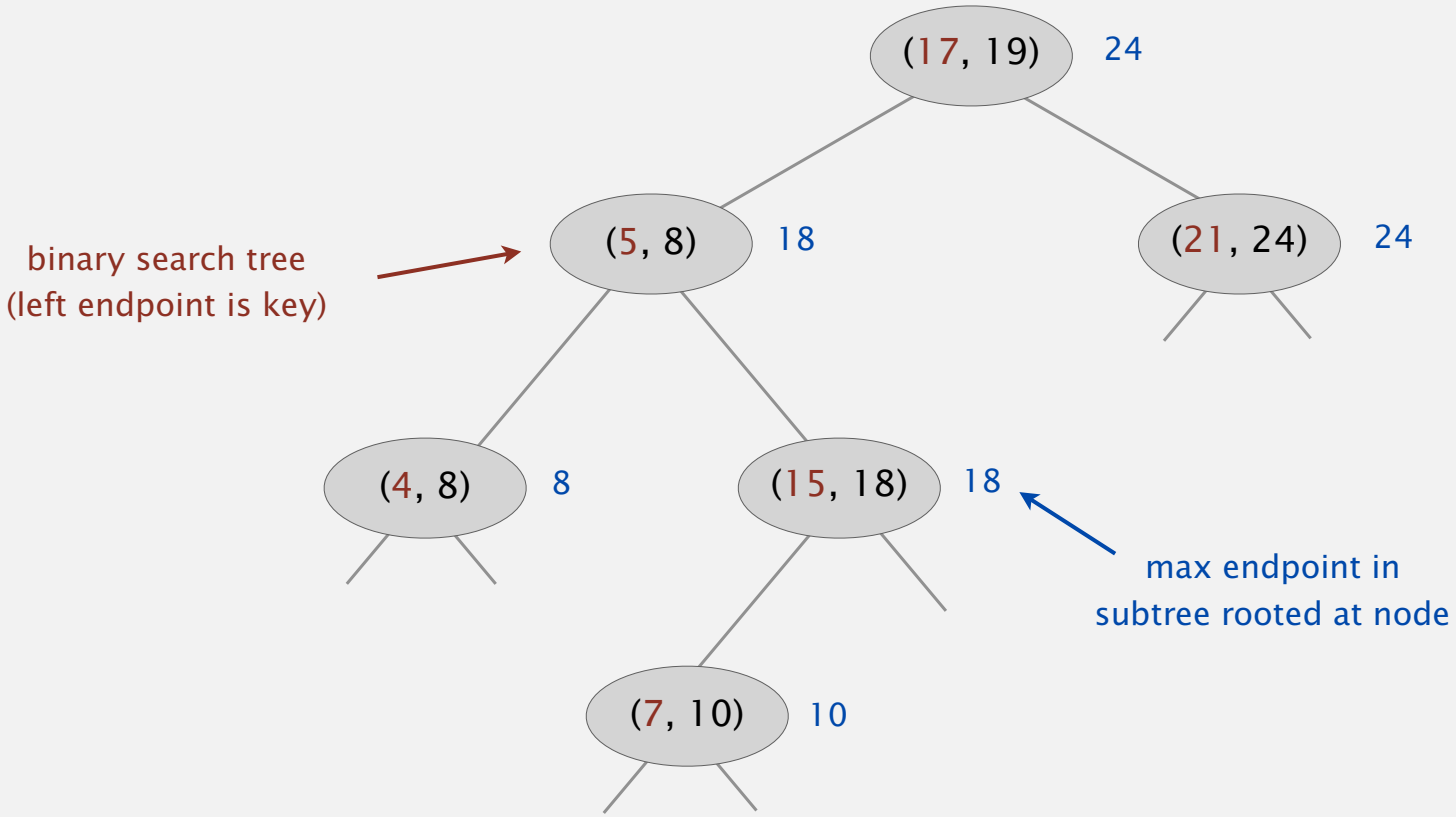
public class IntervalST<Key extends Comparable<Key>, Value>		
	IntervalST()	*create interval search tree*
void	put(Key lo, Key hi, Value val)	*put interval-value pair into ST*
Value	get(Key lo, Key hi)	*value paired with given interval*
void	delete(Key lo, Key hi)	*delete the given interval*
Iterable<Value>	intersects(Key lo, Key hi)	*all intervals that intersect the given interval*

Nondegeneracy assumption. No two intervals have the same left endpoint.

Interval search trees

Create BST, where each node stores an interval (lo, hi).

- Use left endpoint as BST key.
- Store max endpoint in subtree rooted at node.

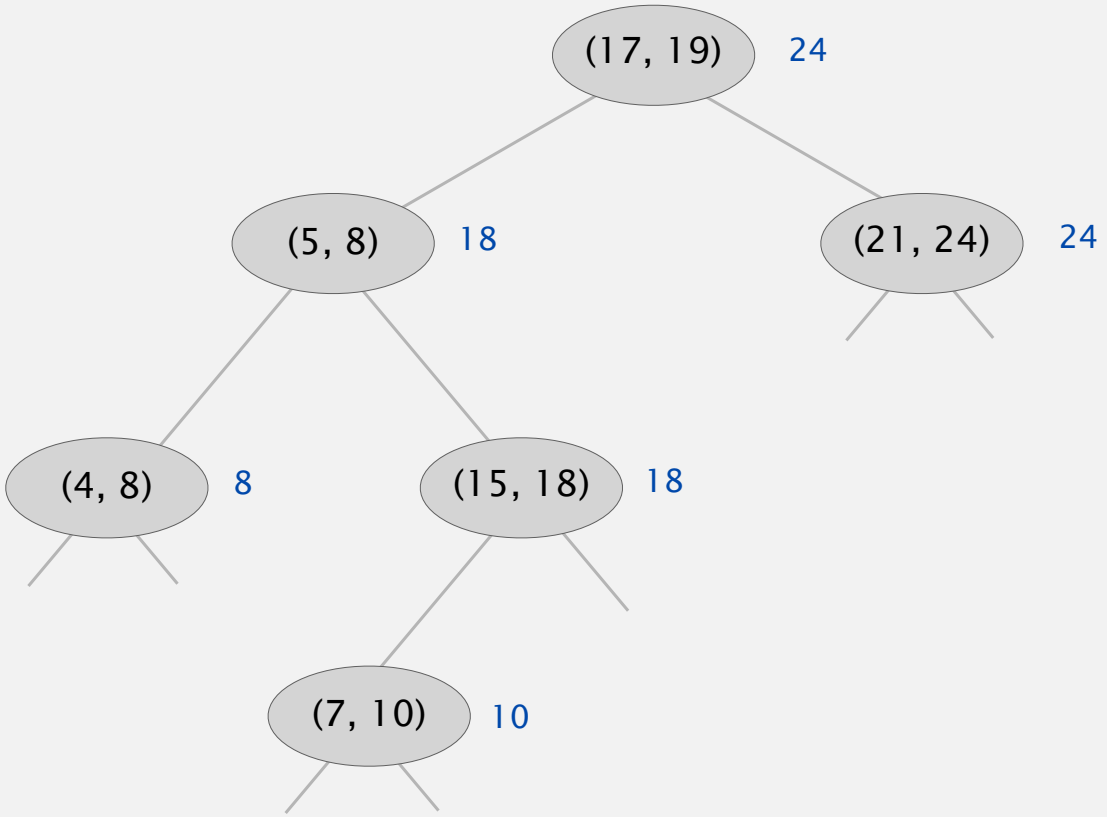


Interval search tree demo

To insert an interval (lo, hi) :

- Insert into BST, using lo as the key.
- Update max in each node on search path.

insert interval (16, 22)

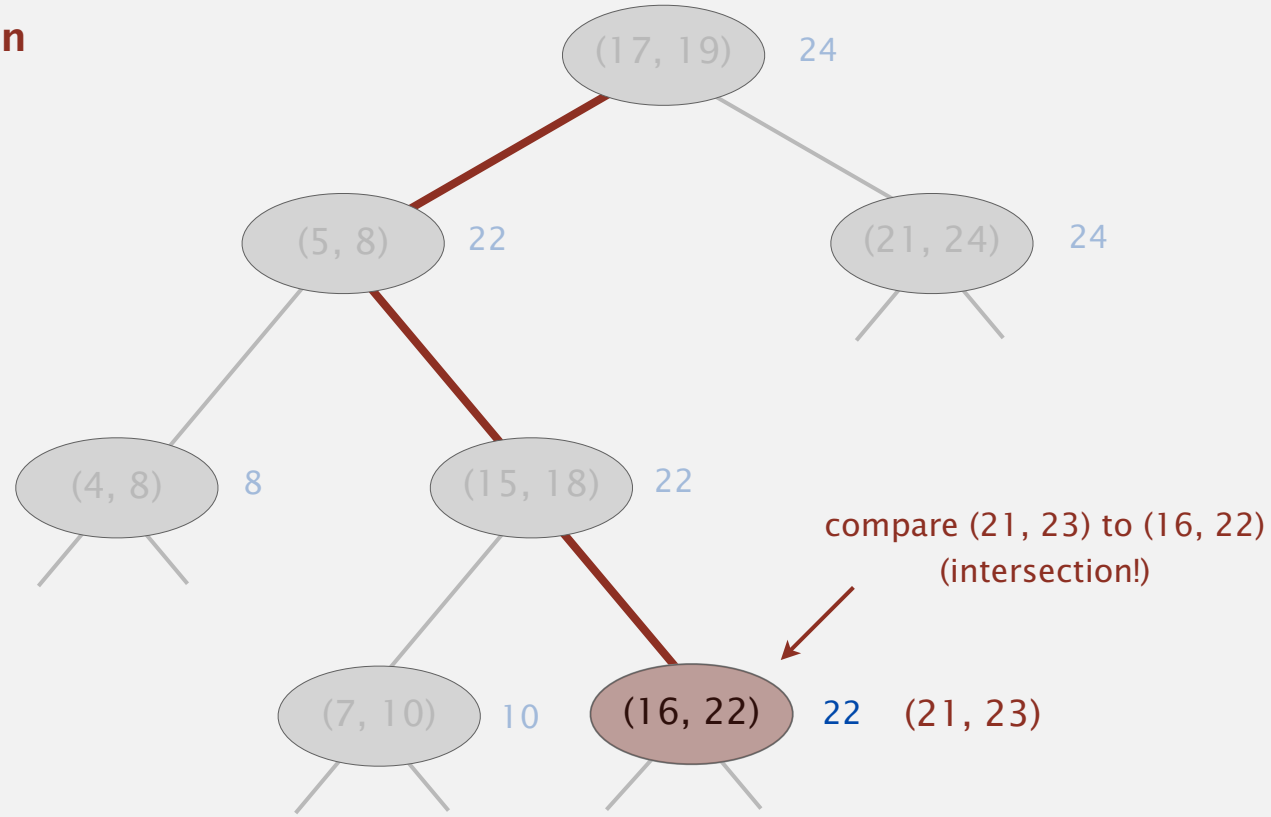


Interval search tree demo

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection search for (21, 23)



Search for an intersecting interval implementation

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

```
Node x = root;
while (x != null)
{
   if      (x.interval.intersects(lo, hi)) return x.interval;
   else if (x.left == null)                x = x.right;
   else if (x.left.max < lo)               x = x.right;
   else                                    x = x.left;
}
return null;
```

Search for an intersecting interval analysis

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

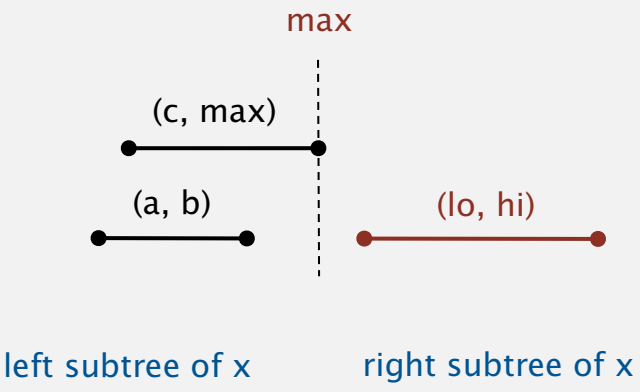
Case 1. If search goes right, then no intersection in left.

Pf. Suppose search goes right and left subtree is non empty.

- Max endpoint max in left subtree is less than lo.
- For any interval (a, b) in left subtree of x, we have $b \leq max < lo$.

definition of max

reason for going right



Search for an intersecting interval analysis

To search for any one interval that intersects query interval (lo, hi):

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

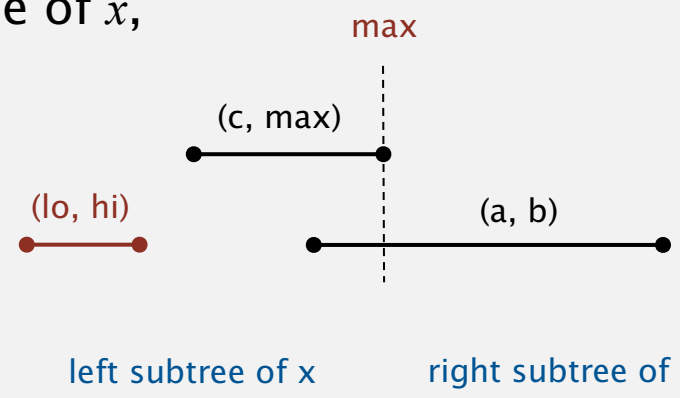
Case 2. If search goes left, then there is either an intersection in left subtree or no intersections in either.

Pf. Suppose no intersection in left.

- Since went left, we have $lo \le max$.
- Then for any interval (a, b) in right subtree of x, $hi < c \le a \Rightarrow$ no intersection in right.

no intersections in left subtree

intervals sorted by left endpoint



Interval search tree: analysis

Implementation. Use a red-black BST to guarantee performance.

easy to maintain auxiliary information
using log N extra work per op

operation	brute	interval search tree	best in theory
insert interval	1	log N	log N
find interval	N	log N	log N
delete interval	N	log N	log N
find any one interval that intersects (lo, hi)	N	log N	log N
find all intervals that intersects (lo, hi)	N	R log N	R + log N

order of growth of running time for N intervals

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

http://algs4.cs.princeton.edu

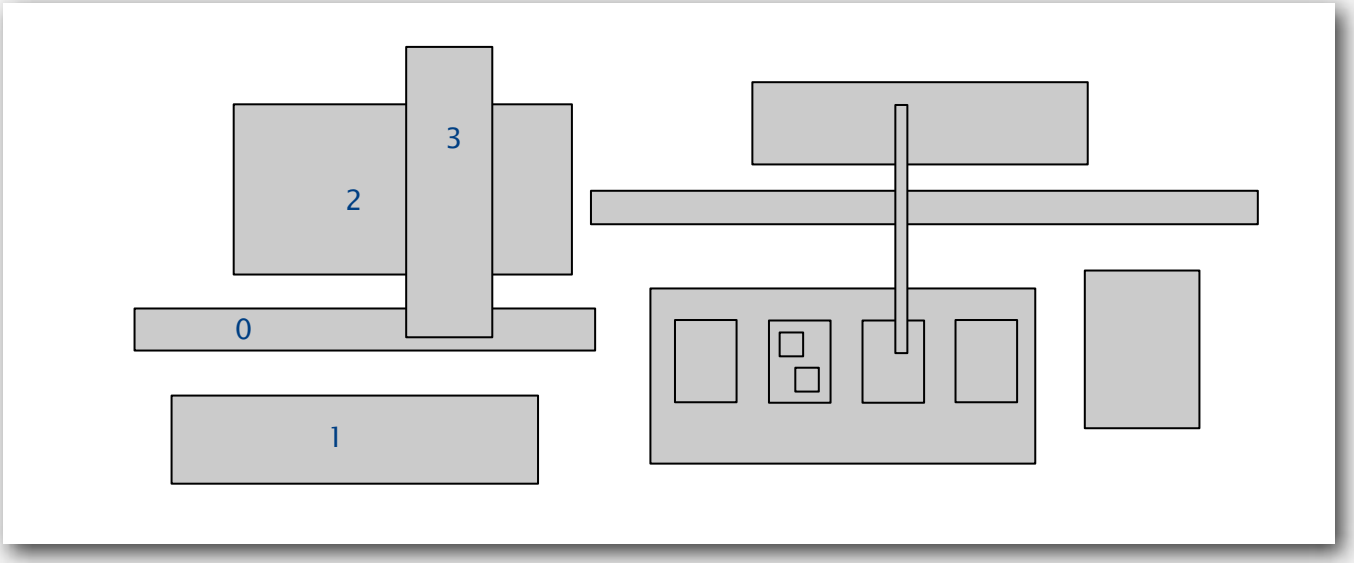
Geometric Applications of BSTs

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

Orthogonal rectangle intersection

Goal. Find all intersections among a set of N orthogonal rectangles.

Quadratic algorithm. Check all pairs of rectangles for intersection.



Non-degeneracy assumption. All x- and y-coordinates are distinct.

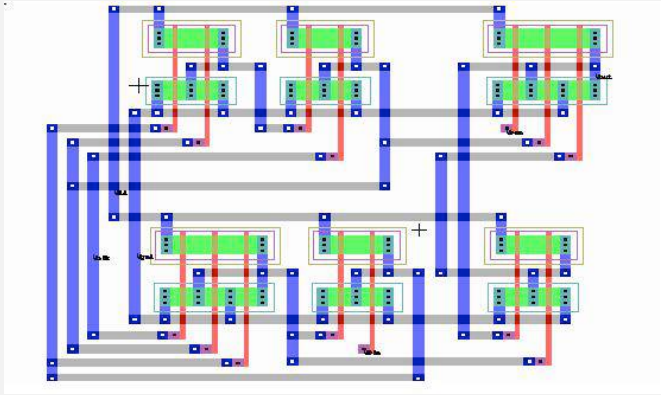
Microprocessors and geometry

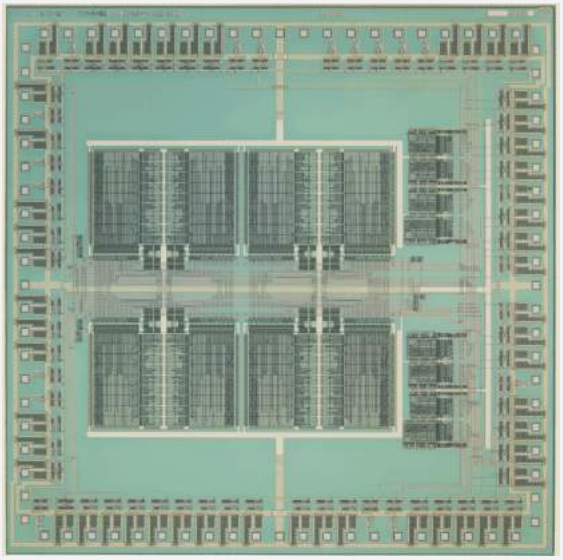
Early 1970s. microprocessor design became a geometric problem.

- 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 = orthogonal rectangle intersection search.

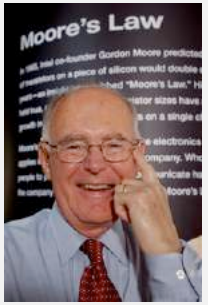




Algorithms and Moore's law

"Moore's law." Processing power doubles every 18 months.

- 197x: check N rectangles.
- 197(x+1.5): check $2N$ rectangles on a 2x-faster computer.



Gordon Moore

Bootstrapping. We get to use the faster computer for bigger circuits.

But bootstrapping is not enough if using a quadratic algorithm:

- 197x: takes M days.
- 197(x+1.5): takes $(4M) / 2 = 2M$ days. (!)

quadratic algorithm

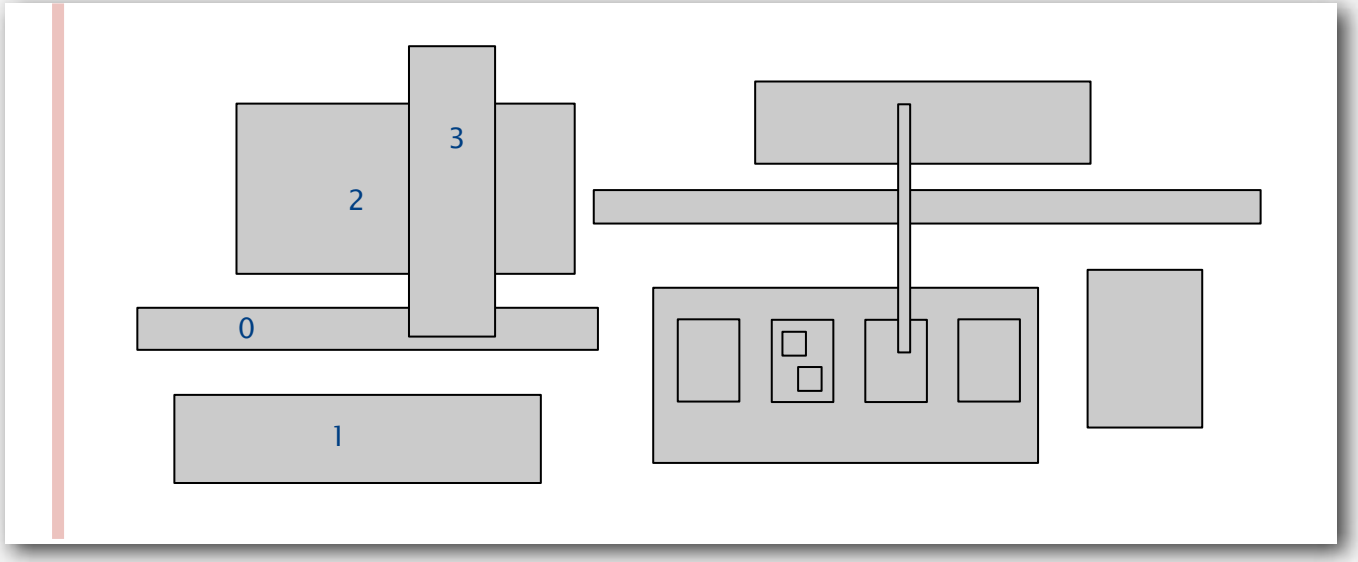
2x-faster computer

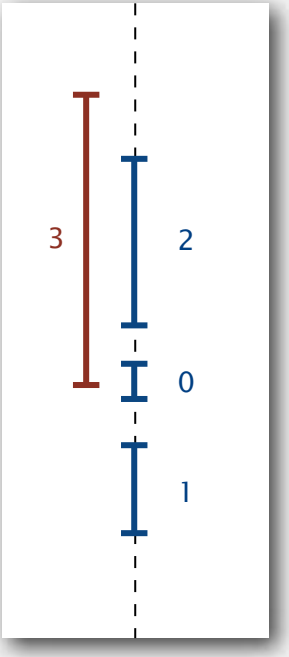
Bottom line. Linearithmic algorithm is necessary to sustain Moore's Law.

Orthogonal rectangle intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates of left and right endpoints define events.
- Maintain set of rectangles that intersect the sweep line in an interval search tree (using y-intervals of rectangle).
- Left endpoint: interval search for y-interval of rectangle; insert y-interval.
- Right endpoint: remove y-interval.





y-coordinates

Orthogonal rectangle intersection: sweep-line analysis

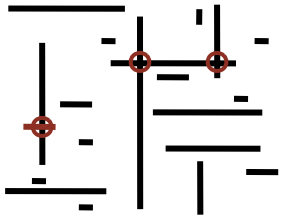
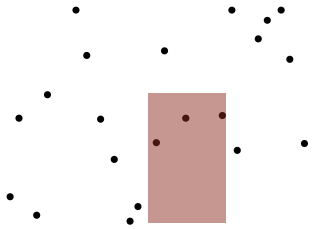
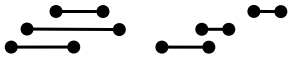
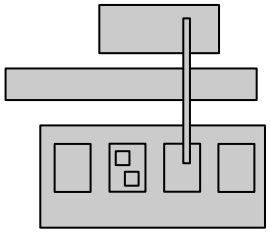
Proposition. Sweep line algorithm takes time proportional to $N \log N + R \log N$ to find R intersections among a set of N rectangles.

Pf.

- Put x-coordinates on a PQ (or sort). ← N log N
- Insert y-intervals into ST. ← N log N
- Delete y-intervals from ST. ← N log N
- Interval searches for y-intervals. ← N log N + R log N

Bottom line. Sweep line reduces 2d orthogonal rectangle intersection search to 1d interval search.

Geometric applications of BSTs

problem	example	solution
1d range search		BST
2d orthogonal line segment intersection		sweep line reduces to 1d range search
kd range search		kd tree
1d interval search		interval search tree
2d orthogonal rectangle intersection		sweep line reduces to 1d interval search

http://algs4.cs.princeton.edu

Geometric Applications of BSTs

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

http://algs4.cs.princeton.edu

GEOMETRIC APPLICATIONS OF BSTS

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