COS 226, FALL 2014

ALGORITHMS AND DATA STRUCTURES

KEVIN WAYNE

PRINCETON UNIVERSITY

http://www.princeton.edu/~cos226

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.

topic	data structures and algorithms
data types	stack, queue, bag, union-find, priority queue
sorting	quicksort, mergesort, heapsort, radix sorts
searching	BST, red-black BST, hash table
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	KMP, regular expressions, tries, data compression
advanced	B-tree, kd-tree, suffix array, maxflow

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

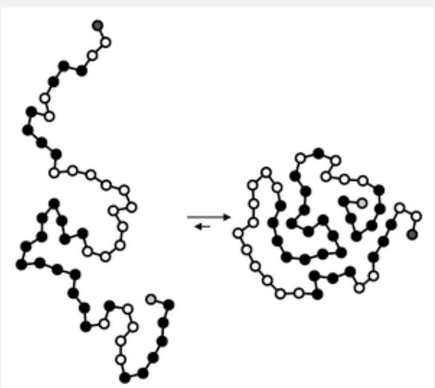
Security. Cell phones, e-commerce, voting machines, ...

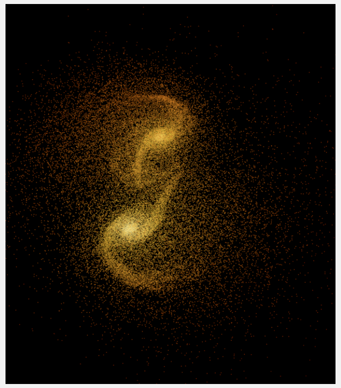
Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮

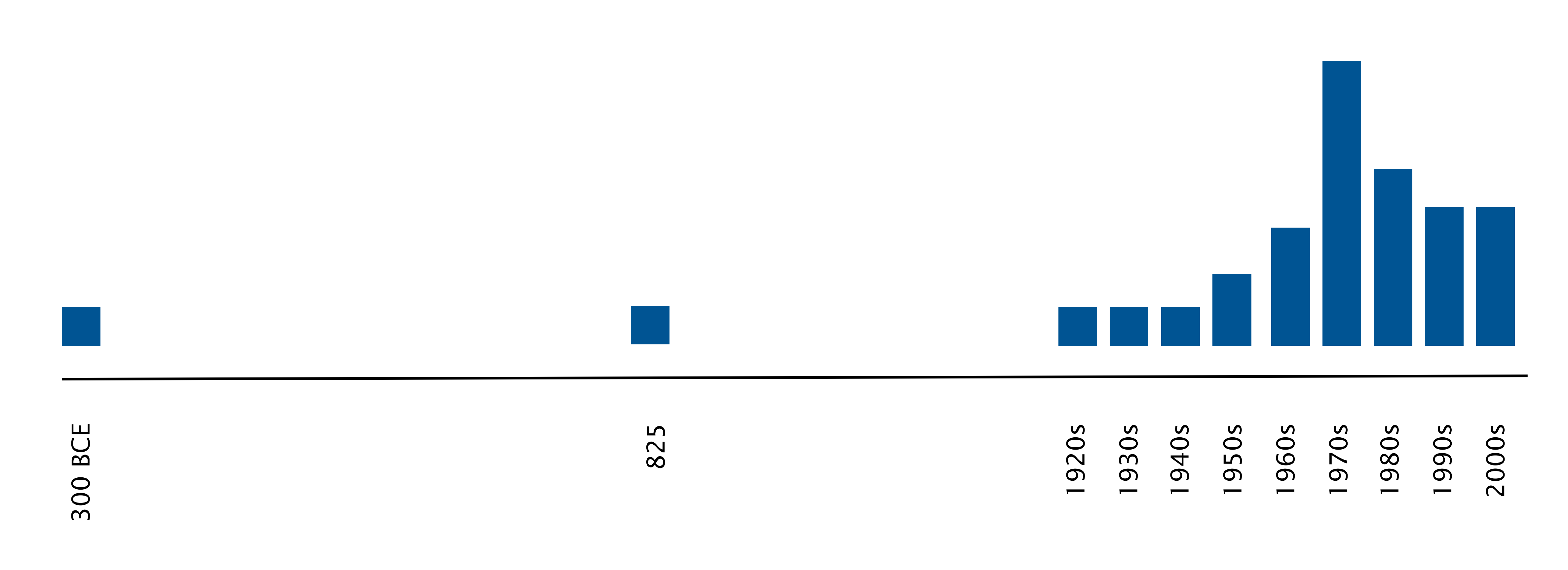




Why study algorithms?

Old roots, new opportunities.

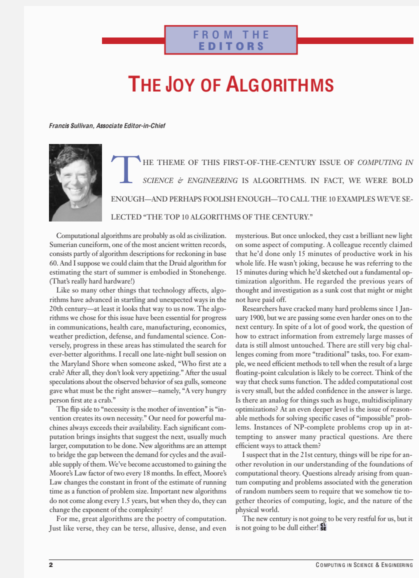
- Study of algorithms dates at least to Euclid.
- Named after Muḥammad ibn Mūsā al-Khwārizmī.
- Formalized by Church and Turing in 1930s.
- Some important algorithms were discovered by undergraduates in a course like this!



Why study algorithms?

For intellectual stimulation.

> *" For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. "* — *Francis Sullivan*


FROM THE EDITORS

THE JOY OF ALGORITHMS

Mysterious algorithm was 4% of trading activity last week

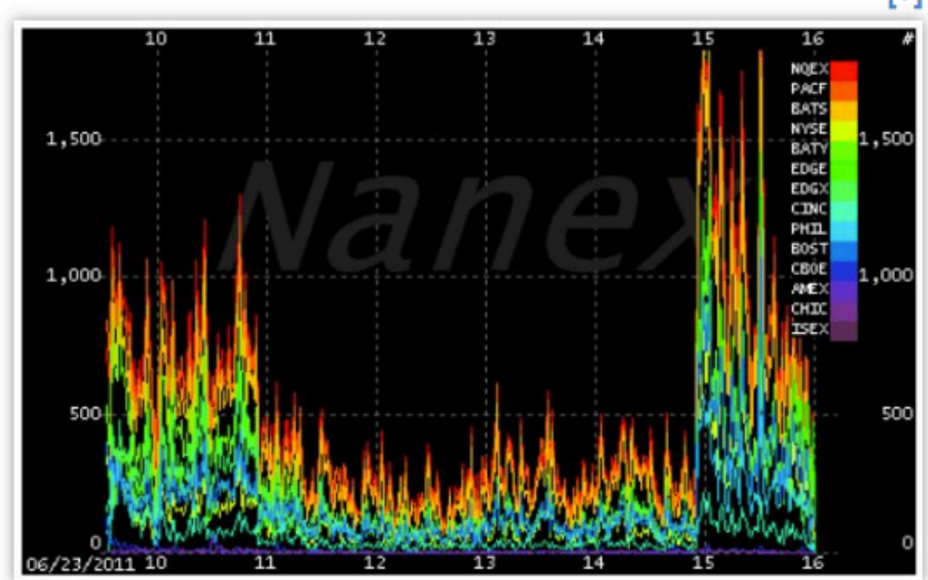
October 11, 2012

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of high-frequency trading activity.

The motive of the algorithm is still unclear, *CNBC* reports.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

[+]



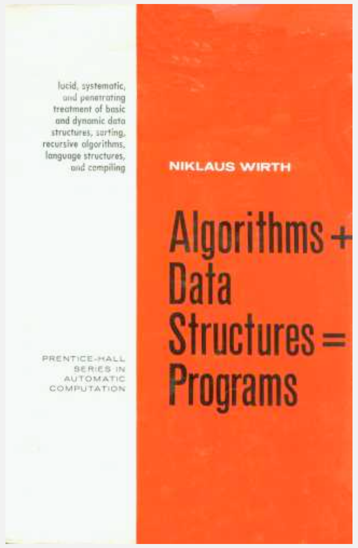
Generic high frequency rrading chart (credit: Nanex)

Why study algorithms?

To become a proficient programmer.

" *I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.* "

— Linus Torvalds (creator of Linux)



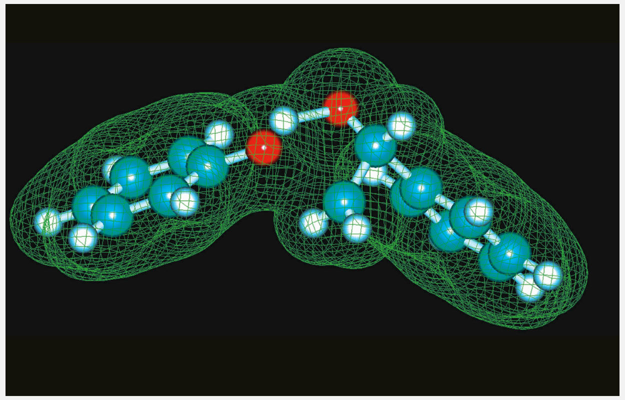
" *Algorithms + Data Structures = Programs.* " *— Niklaus Wirth*

Why study algorithms?

They may unlock the secrets of life and of the universe.

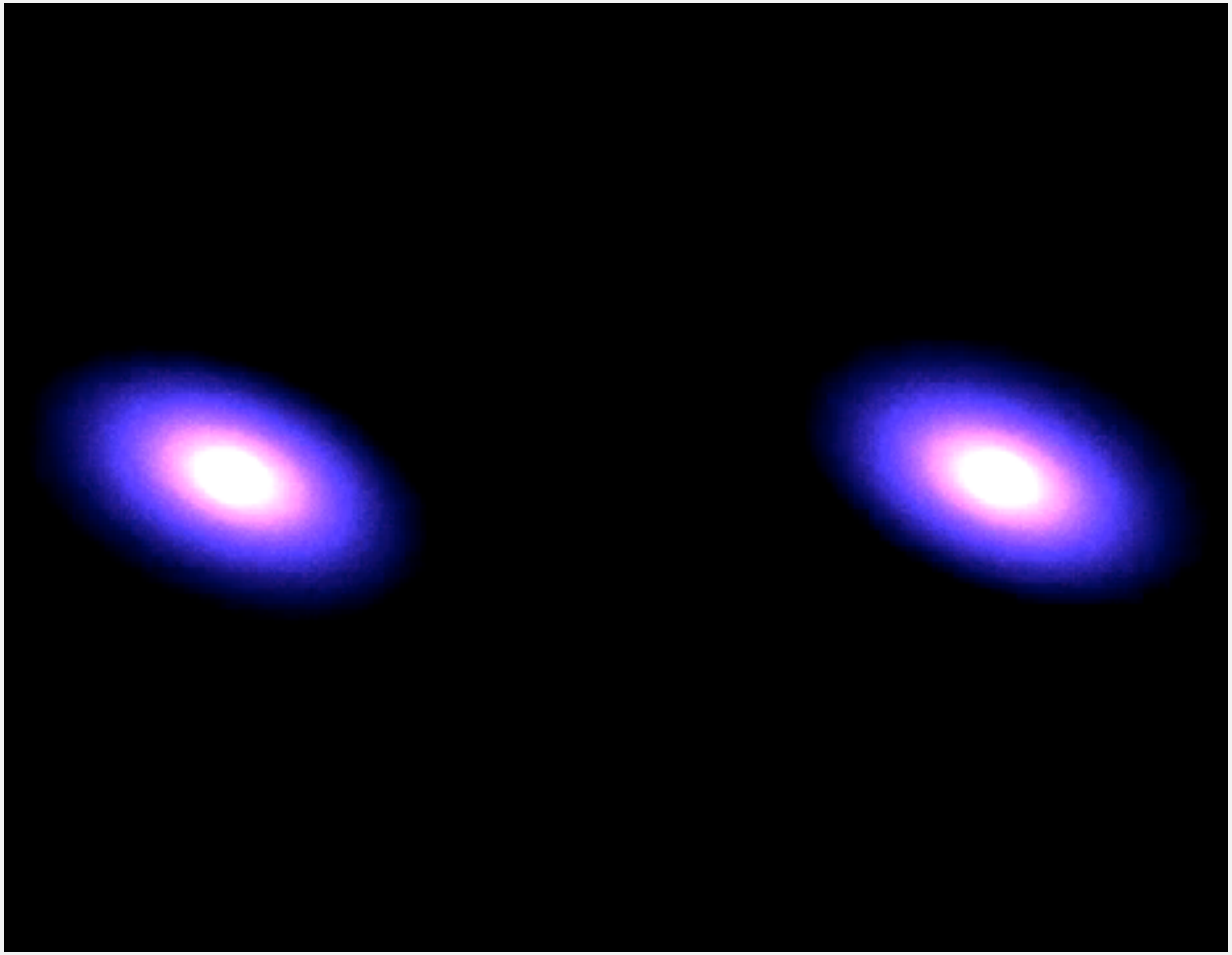
> *" Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube. "*
> — *Royal Swedish Academy of Sciences*
> *(Nobel Prize in Chemistry 2013)*

Martin Karplus, Michael Levitt, and Arieh Warshel



Why study algorithms?

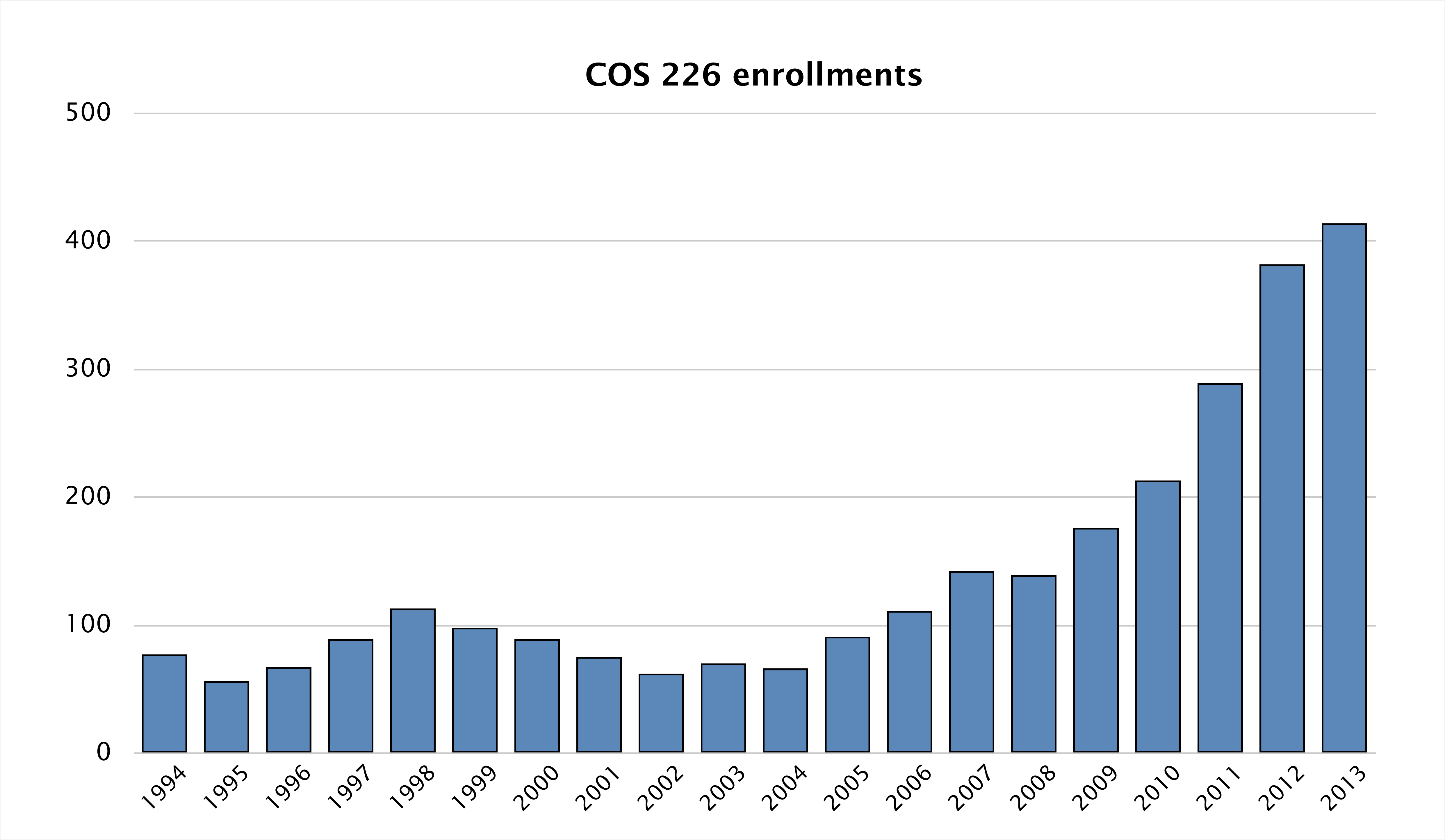
To solve problems that could not otherwise be addressed.



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

Why study algorithms?

Everybody else is doing it.



Why study algorithms?

For fun and profit.

Apple Computer
facebook
Cisco Systems
Google
Nintendo
Jane Street
IBM
Morgan Stanley
NETFLIX
Adobe
RSA Security
D E Shaw & Co
ORACLE
PANDORA
Akamai
YAHOO!
amazon.com
Microsoft
PIXAR Animation Studios

Why study algorithms?

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- To solve problems that could not otherwise be addressed.
- Everybody else is doing it.
- For fun and profit.

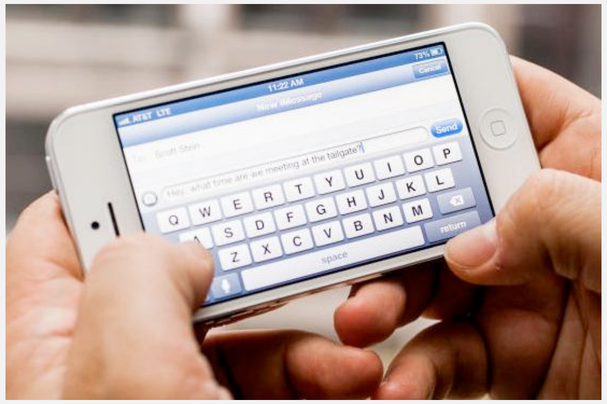
Lectures

Traditional lectures. Introduce new material.

Electronic devices. Permitted, but only to enhance lecture.

no

no



no

What	When	Where	Who	Office Hours
L01	**TTh 11–12:20**	**Friend 101**	**Kevin Wayne**	see web

Lectures

Traditional lectures. Introduce new material.

Flipped lectures.

- Watch videos online before lecture.
- Complete pre-lecture activities.
- Attend two "flipped" lecture per week (interactive, collaborative, experimental).
- Apply via web ASAP: results by 5pm today.

What	When	Where	Who	Office Hours
L01	**TTh 11-12:20**	**Friend 101**	**Kevin Wayne**	see web
L02	**TTh 11-12:20**	**Sherrerd 001**	**Andy Guna**	see web

Precepts

Discussion, problem-solving, background for assignments.

What	When	Where	Who	Office Hours
P01	**F 9–9:50am**	**Friend 108**	**Andy Guna †**	see web
P02	**F 10–10:50am**	**Friend 108**	**Jérémie Lumbroso**	see web
P03	**F 11–11:50am**	**Friend 109**	**Joshua Wetzel**	see web
P03A	**F 11–11:50am**	**Friend 108**	**Jérémie Lumbroso**	see web
P04	**F 12:30–1:20pm**	**Friend 108**	**Robert MacDavid**	see web
P04A	**F 12:30–1:20pm**	**Friend 109**	**Shivam Agarwal**	see web

† lead preceptor

Coursework and grading

Programming assignments. 45%

- Due on Wednesday at 11pm via electronic submission.
- Collaboration/lateness policies: see web.

Exercises. 10%

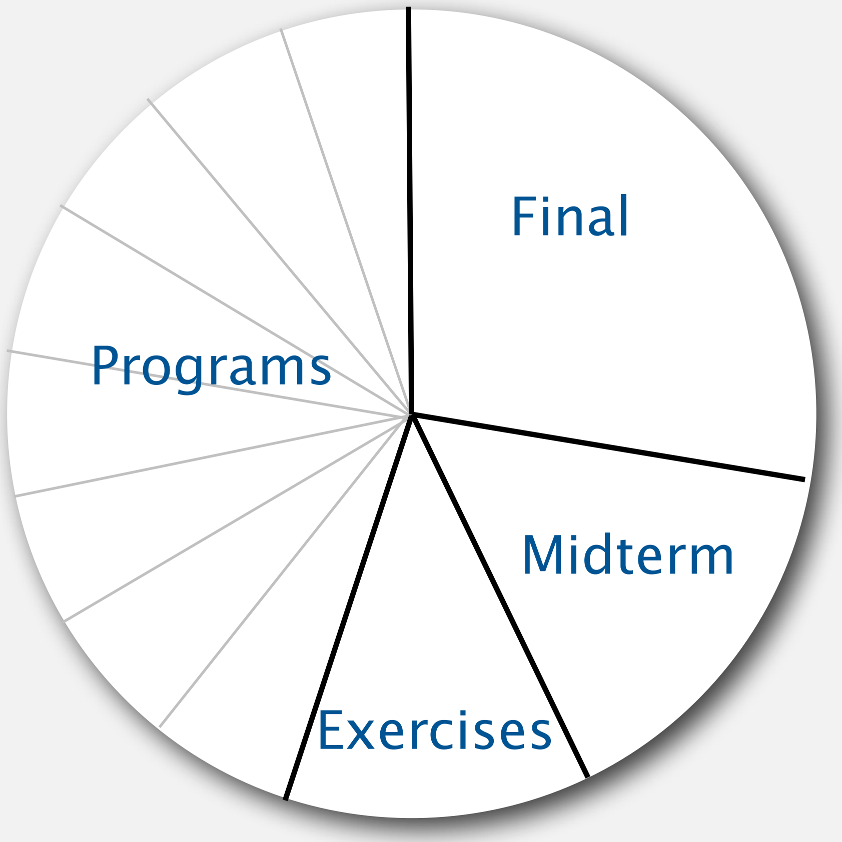
- Due on Sundays at 11pm in Blackboard.
- Collaboration/lateness policies: see web.

Exams. 15% + 30%

- Midterm (in class on Tuesday, October 21).
- Final (to be scheduled by Registrar).

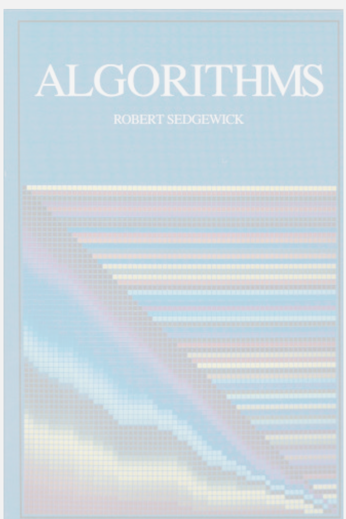
Staff discretion. [adjust borderline cases]

- Report errata.
- Contribute to Piazza discussion forum.
- Attend and participate in precept/lecture.

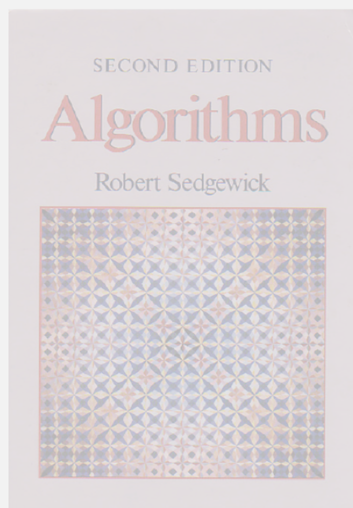


Resources (textbook)

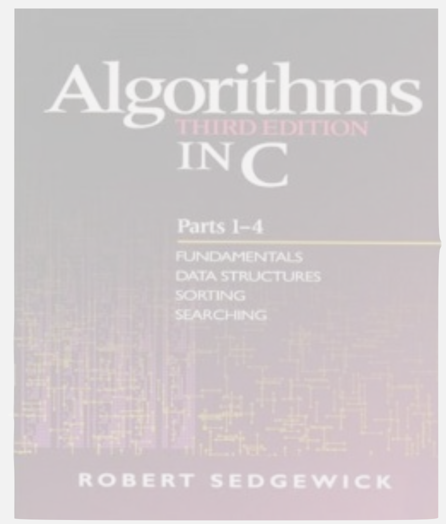
Required reading. Algorithms 4th edition by R. Sedgewick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.



1st edition (1982)



2nd edition (1988)



3rd edition (1997)

3rd book scanned by Google books

4th edition (2011)

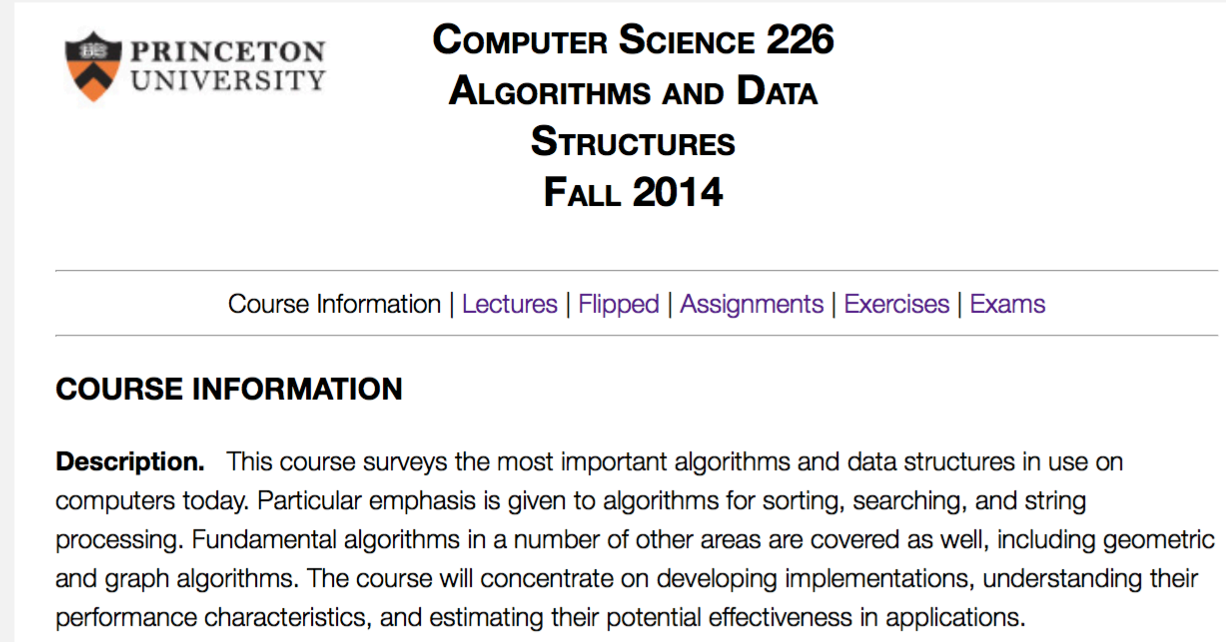
Available in hardcover and Kindle.

- Online: Amazon ($60/$35 to buy), Chegg ($25 to rent), ...
- Brick-and-mortar: Labyrinth Books (122 Nassau St.).
- On reserve: Engineering library.

Resources (web)

Course content.

- Course info.
- Lecture slides.
- Flipped lectures.
- Programming assignments.
- Exercises.
- Exam archive.


PRINCETON UNIVERSITY

COMPUTER SCIENCE 226
ALGORITHMS AND DATA STRUCTURES
FALL 2014

Course Information | Lectures | Flipped | Assignments | Exercises | Exams

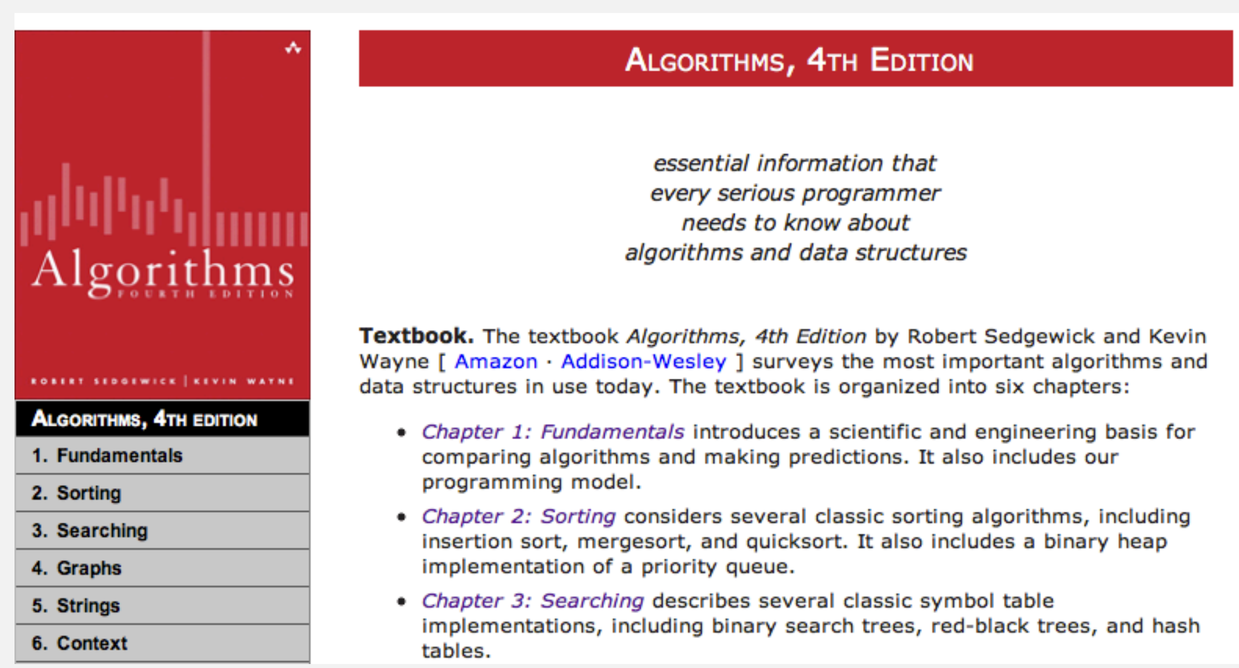
COURSE INFORMATION

Description. This course surveys the most important algorithms and data structures in use on computers today. Particular emphasis is given to algorithms for sorting, searching, and string processing. Fundamental algorithms in a number of other areas are covered as well, including geometric and graph algorithms. The course will concentrate on developing implementations, understanding their performance characteristics, and estimating their potential effectiveness in applications.

http://www.princeton.edu/~cos226

Booksite.

- Brief summary of content.
- Download code from book.
- APIs and Javadoc.


Algorithms FOURTH EDITION

ROBERT SEDGEWICK | KEVIN WAYNE

ALGORITHMS, 4TH EDITION

1. Fundamentals
2. Sorting
3. Searching
4. Graphs
5. Strings
6. Context

ALGORITHMS, 4TH EDITION

essential information that every serious programmer needs to know about algorithms and data structures

Textbook. The textbook *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne [Amazon · Addison-Wesley] surveys the most important algorithms and data structures in use today. The textbook is organized into six chapters:

- *Chapter 1: Fundamentals* introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- *Chapter 2: Sorting* considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also includes a binary heap implementation of a priority queue.
- *Chapter 3: Searching* describes several classic symbol table implementations, including binary search trees, red-black trees, and hash tables.

http://algs4.cs.princeton.edu

Resources (people)

Piazza discussion forum.

- Low latency, low bandwidth.
- Mark solution-revealing questions as private.

http://piazza.com/princeton/fall2014/cos226

Office hours.

- High bandwidth, high latency.
- See web for schedule.

http://www.princeton.edu/~cos226

Computing laboratory.

- Undergrad lab TAs.
- For help with debugging.
- See web for schedule.

http://labta.cs.princeton.edu

What's ahead?

Today. Attend traditional lecture (everyone).

Tomorrow. Attend precept (everyone).

FOR i = 1 to N

- Sunday: exercises due (via Bb submission).
- Tuesday: traditional/flipped lecture.
- Wednesday: programming assignment due.
- Thursday: traditional/flipped lecture.
- Friday: precept.

protip: start early

Q+A

Not registered? Go to any precept tomorrow.

Change precept? Use SCORE.

All possible precepts closed? See Colleen Kenny-McGinley in CS 210.

Haven't taken COS 126? See COS placement officer.

Placed out of COS 126? Review Sections 1.1–1.2 of Algorithms 4/e.

http://algs4.cs.princeton.edu

1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why not.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.

http://algs4.cs.princeton.edu

1.5 UNION-FIND

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Dynamic connectivity problem

Given a set of N elements, support two operation:

- Connect two elements.
- Is there a path connecting the two elements?

connect 4 and 3

connect 3 and 8

connect 6 and 5

connect 9 and 4

connect 2 and 1

are 0 and 7 connected? ✗

are 8 and 9 connected? ✔

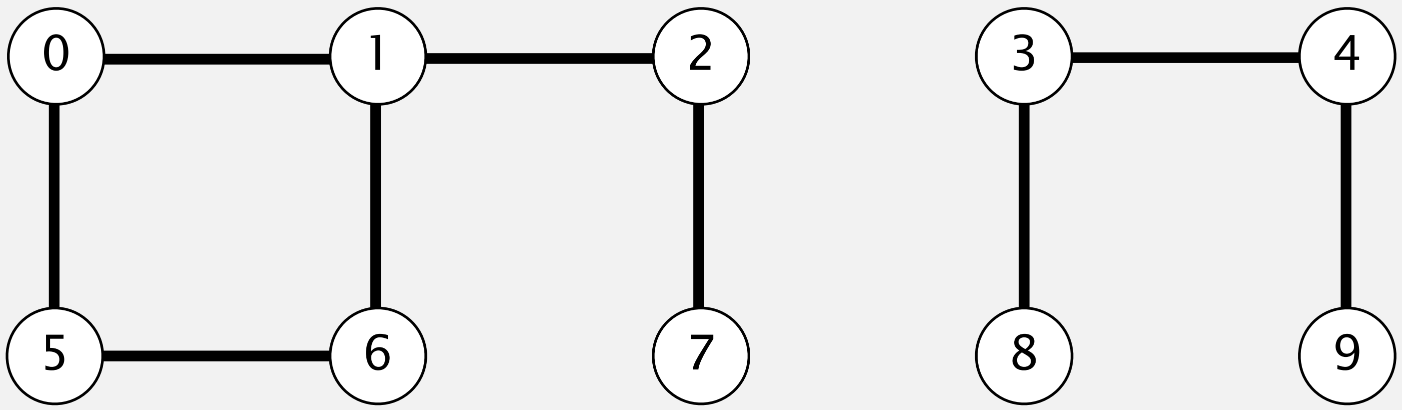
connect 5 and 0

connect 7 and 2

connect 6 and 1

connect 1 and 0

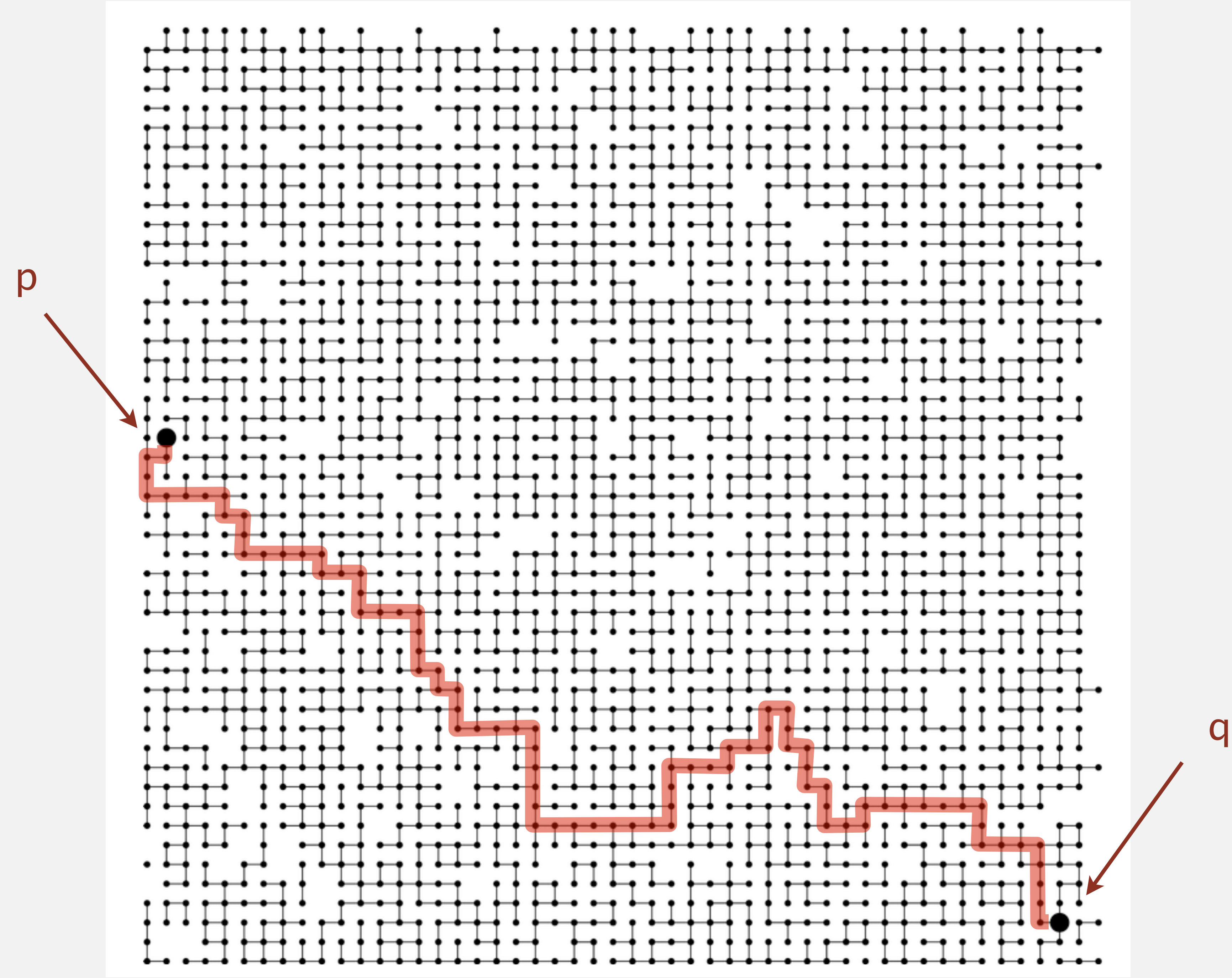
are 0 and 7 connected? ✔



A larger connectivity example

Q. Is there a path connecting elements p and q ?

finding the path explicitly is a harder problem
(stay tuned for graph algorithms)



A. Yes.

Modeling the elements

Applications involve manipulating elements of all types.

- Pixels in a digital photo.
- Computers in a network.
- Friends in a social network.
- Transistors in a computer chip.
- Elements in a mathematical set.
- Variable names in a Fortran program.
- Metallic sites in a composite system.

When programming, convenient to name elements 0 to $N - 1$.

- Use integers as array index.
- Suppress details not relevant to union-find.

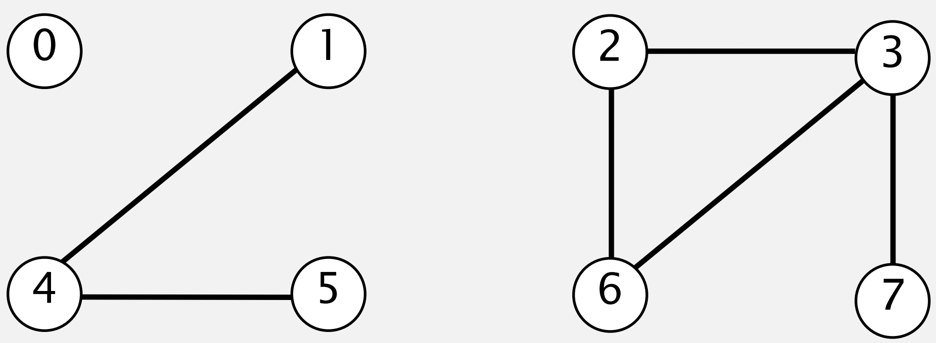
can use symbol table to translate from site names to integers: stay tuned (Chapter 3)

Modeling the connections

We assume "is connected to" is an equivalence relation:

- Reflexive: p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.

Connected component. Maximal set of elements that are mutually connected.



{ 0 } { 1 4 5 } { 2 3 6 7 }

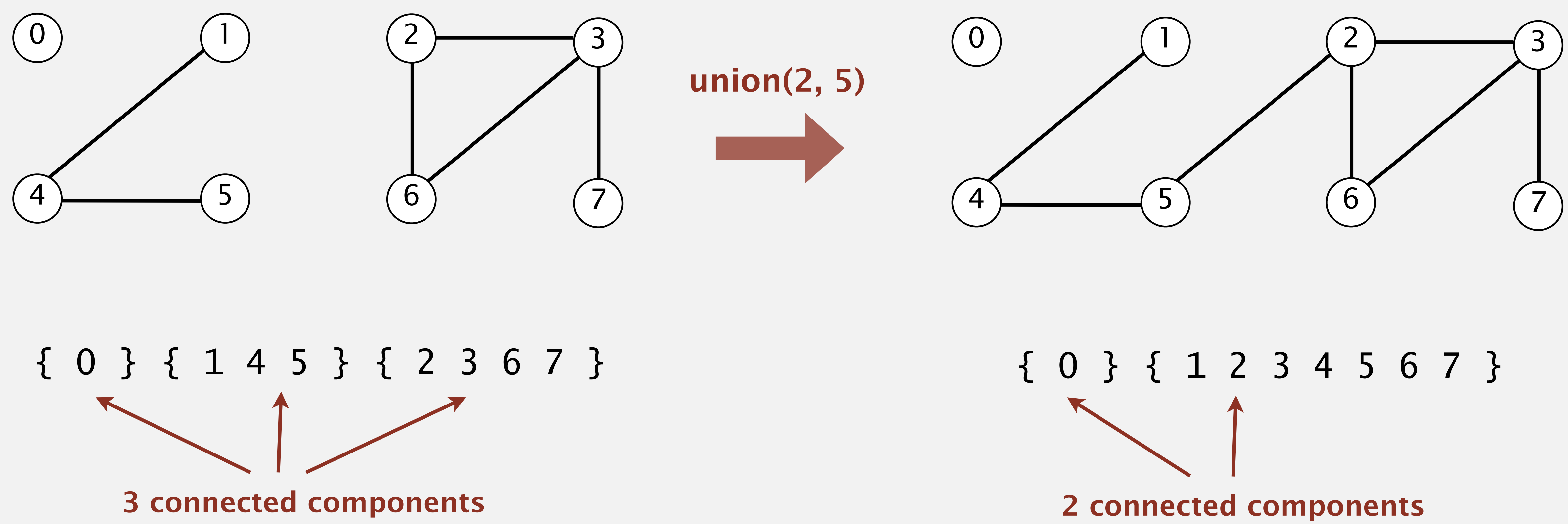
3 connected components

Implementing the operations

Find. In which component is element p ?

Connected. Are elements p and q in the same component?

Union. Replace components containing p and q with their union.



Union-find data type (API)

Goal. Design efficient data structure for union-find.

- Number of elements N can be huge.
- Number of operations M can be huge.
- Union and find operations can be intermixed.

public class UF		
	`UF(int N)`	*initialize union-find data structure with N singleton elements* $(0$ *to* $N-1)$
`void`	`union(int p, int q)`	*add connection between p and q*
`int`	`find(int p)`	*component identifier for p* $(0$ *to* $N-1)$
`boolean`	`connected(int p, int q)`	*are p and q in the same component?*

```
public boolean connected(int p, int q)
{   return find(p) == find(q);   }
```

1-line implementation of connected()

Dynamic-connectivity client

- Read in number of elements N from standard input.
- Repeat:
  - read in pair of integers from standard input
  - if they are not yet connected, connect them and print out pair

```
public static void main(String[] args)
{
   int N = StdIn.readInt();
   UF uf = new UF(N);
   while (!StdIn.isEmpty())
   {
      int p = StdIn.readInt();
      int q = StdIn.readInt();
      if (!uf.connected(p, q))
      {
         uf.union(p, q);
         StdOut.println(p + " " + q);
      }
   }
}
```

```
% more tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
```

already connected

http://algs4.cs.princeton.edu

1.5 Union-Find

- dynamic connectivity
- *quick find*
- quick union
- improvements
- applications

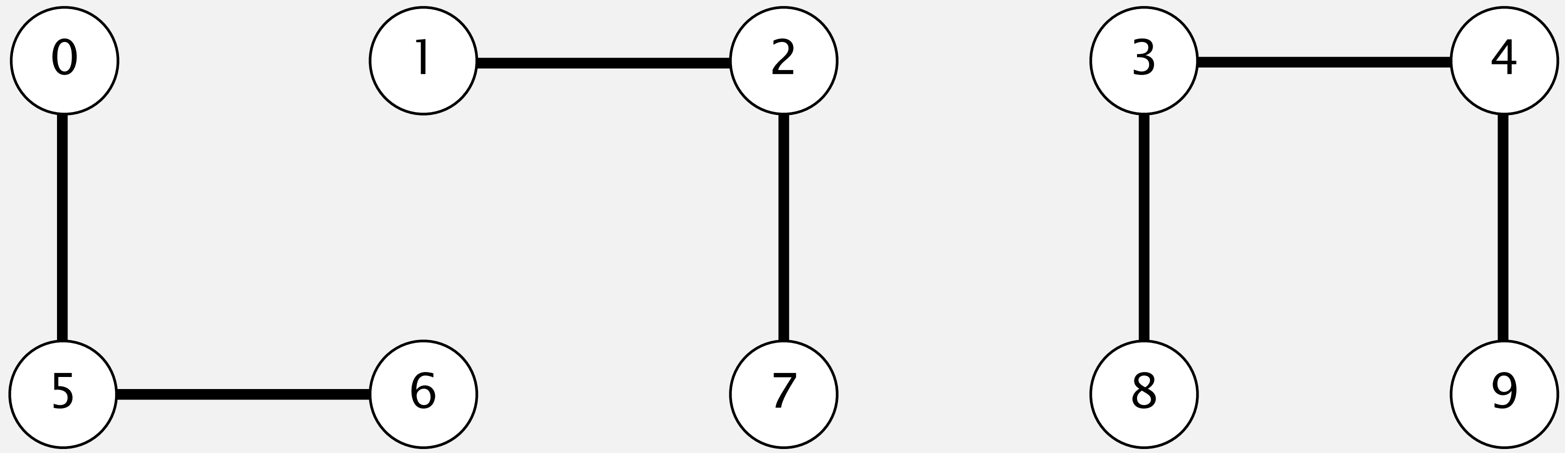
Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` is the id of the component containing `p`.

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected



Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` is the id of the component containing `p`.

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

Find. What is the id of `p`?

Connected. Do `p` and `q` have the same id?

`id[6] = 0; id[1] = 1`
6 and 1 are not connected

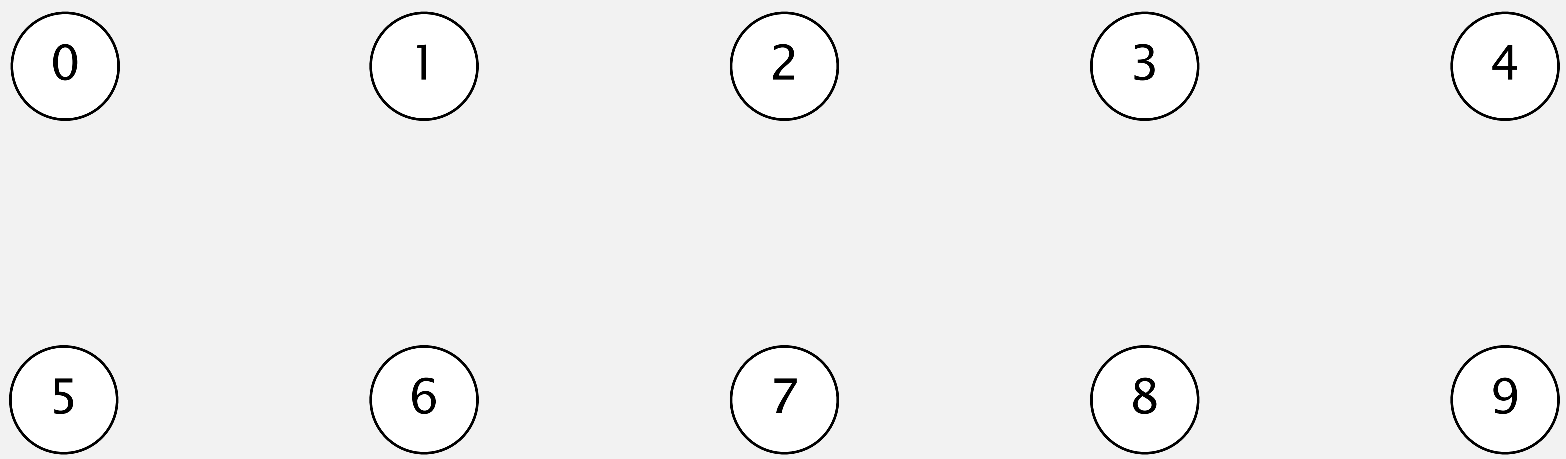
Union. To merge components containing `p` and `q`, change all entries whose id equals `id[p]` to `id[q]`.

	0	1	2	3	4	5	6	7	8	9
`id[]`	1	1	1	8	8	1	1	1	8	8

after union of 6 and 1

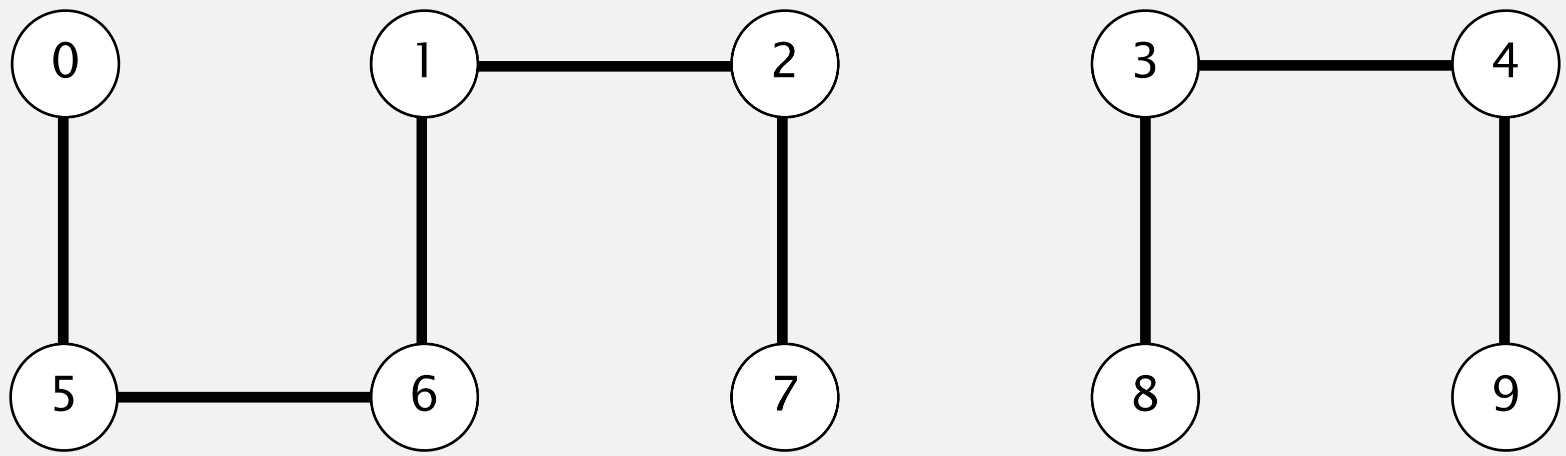
problem: many values can change

Quick-find demo



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Quick-find demo



	0	1	2	3	4	5	6	7	8	9
id[]	1	1	1	8	8	1	1	1	8	8

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
   }

   public int find(int p)
   {  return id[p];  }

   public void union(int p, int q)
   {
      int pid = id[p];
      int qid = id[q];
      for (int i = 0; i < id.length; i++)
         if (id[i] == pid) id[i] = qid;
   }
}
```

set id of each element to itself
(N array accesses)

return the id of p
(1 array access)

change all entries with id[p] to id[q]
(at most 2N + 2 array accesses)

Quick-find is too slow

Cost model. Number of array accesses (for read or write).

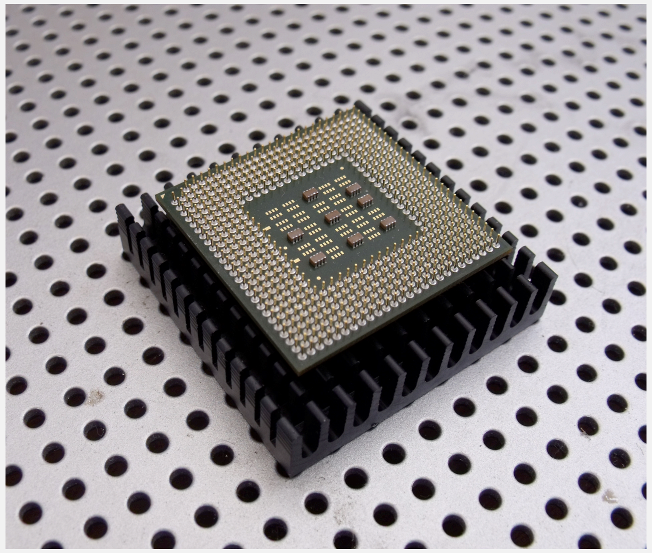
algorithm	initialize	union	find	connected
quick-find	N	N	1	1

order of growth of number of array accesses

quadratic

Union is too expensive. It takes N^2 array accesses to process a sequence of N union operations on N elements.

Quadratic algorithms do not scale



Rough standard (for now).

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

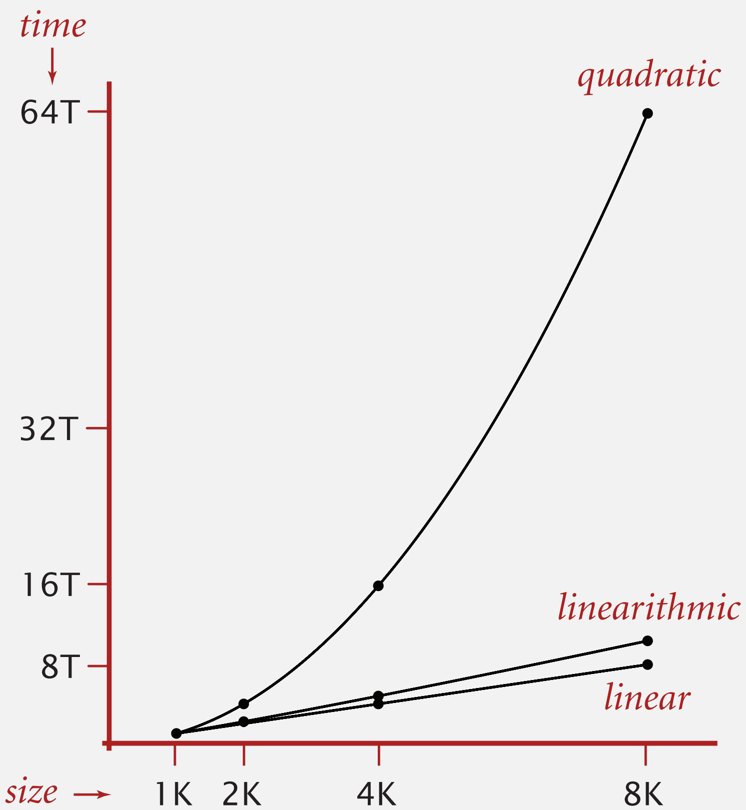
a truism (roughly) since 1950!

Ex. Huge problem for quick-find.

- 10^9 union commands on 10^9 elements.
- Quick-find takes more than 10^{18} operations.
- 30+ years of computer time!

Quadratic algorithms don't scale with technology.

- New computer may be 10x as fast.
- But, has 10x as much memory $\Rightarrow$ want to solve a problem that is 10x as big.
- With quadratic algorithm, takes 10x as long!



http://algs4.cs.princeton.edu

1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

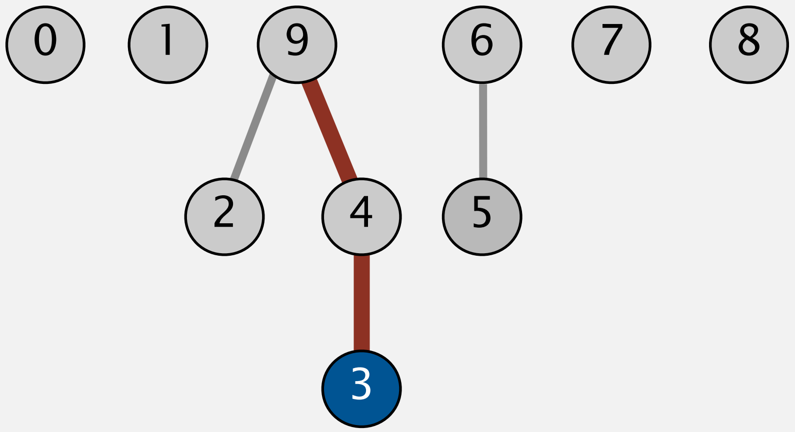
Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change
(algorithm ensures no cycles)

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



parent of 3 is 4

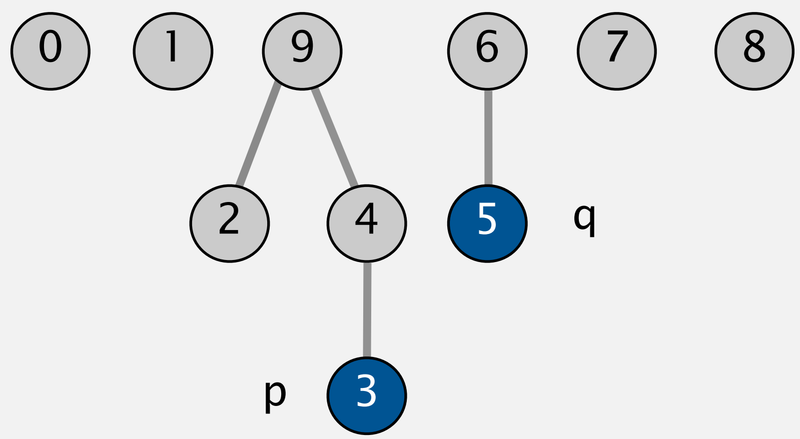
root of 3 is 9

Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



root of 3 is 9
root of 5 is 6
3 and 5 are not connected

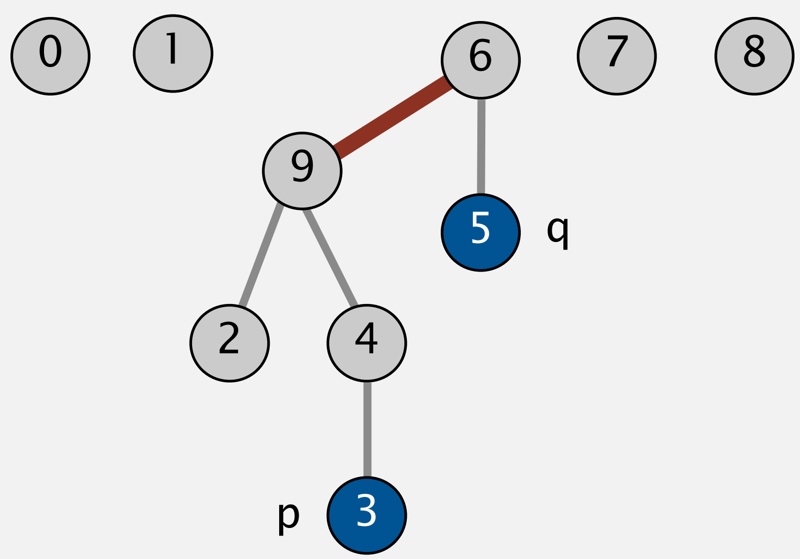
Find. What is the root of `p`?

Connected. Do `p` and `q` have the same root?

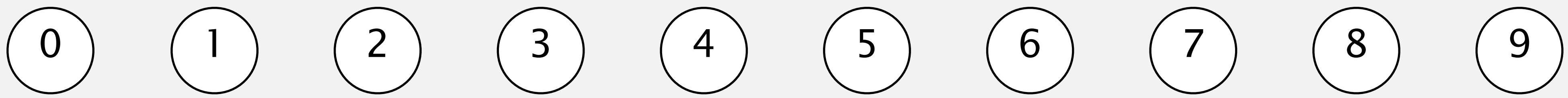
Union. To merge components containing `p` and `q`, set the id of `p`'s root to the id of `q`'s root.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	6

only one value changes

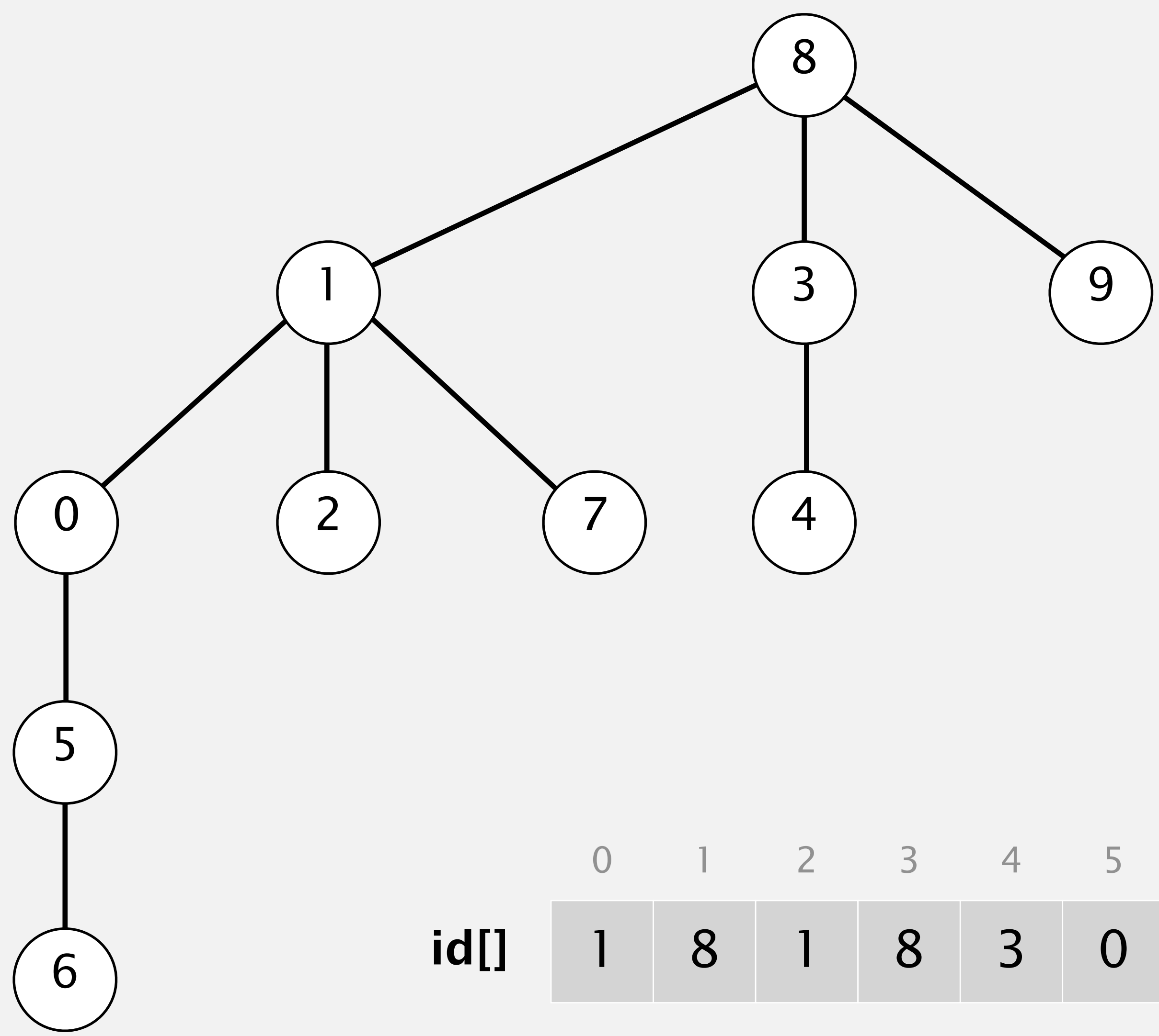


Quick-union demo



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Quick-union demo



	0	1	2	3	4	5	6	7	8	9
id[]	1	8	1	8	3	0	5	1	8	8

Quick-union: Java implementation

```
public class QuickUnionUF
{
   private int[] id;

   public QuickUnionUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++) id[i] = i;
   }

   public int find(int p)
   {
      while (p != id[p]) p = id[p];
      return p;
   }


   public void union(int p, int q)
   {
      int i = find(p);
      int j = find(q);
      id[i] = j;
   }
}
```

set id of each element to itself (N array accesses)

chase parent pointers until reach root (depth of p array accesses)

change root of p to point to root of q (depth of p and q array accesses)

Quick-union is also too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N

← worst case

† includes cost of finding two roots

Quick-find defect.

- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

- Trees can get tall.
- Find/connected too expensive (could be N array accesses).

http://algs4.cs.princeton.edu

1.5 UNION-FIND

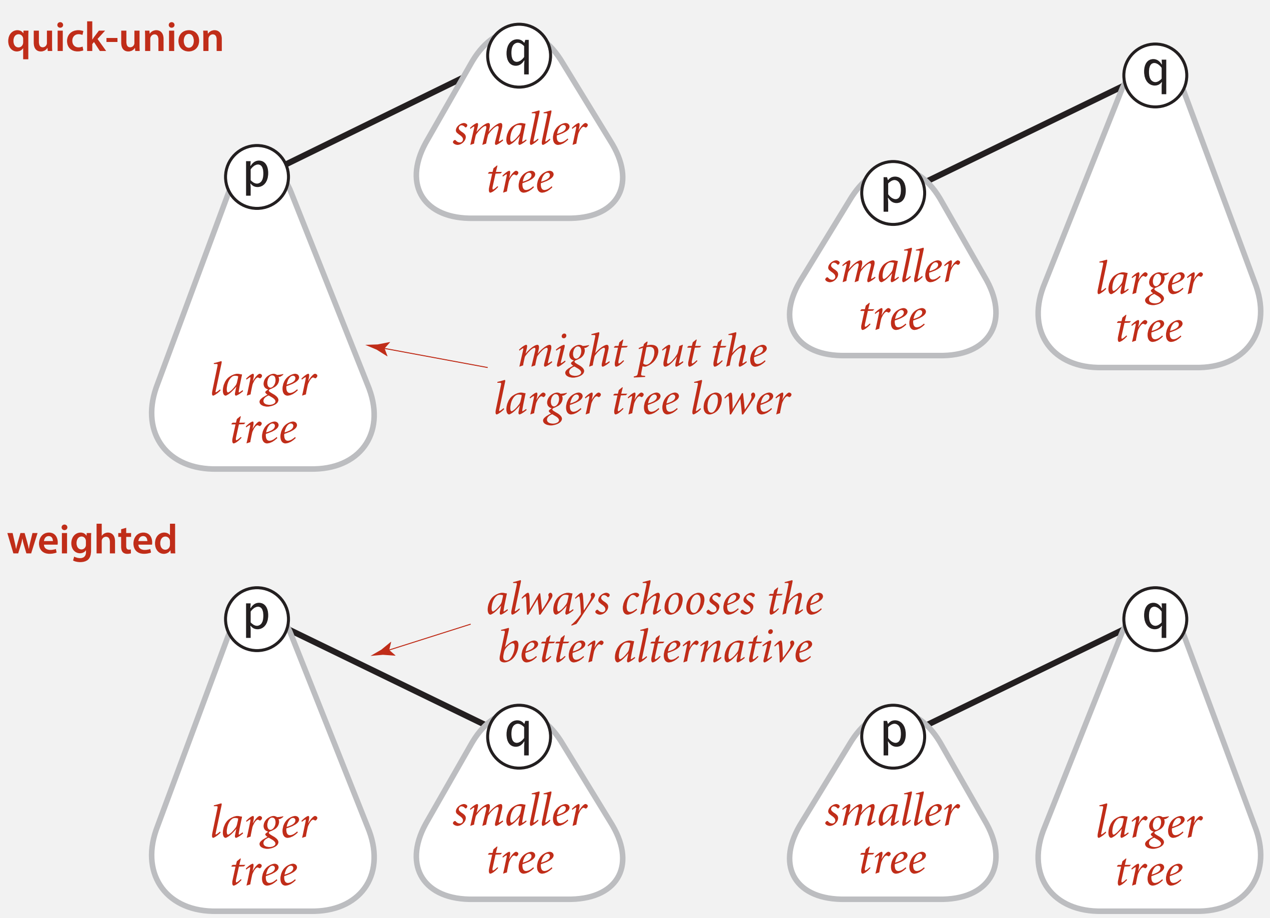
- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Improvement 1: weighting

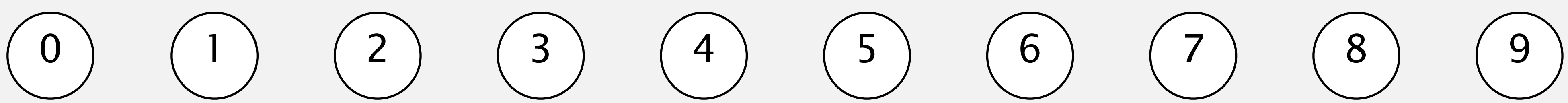
Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of elements).
- Balance by linking root of smaller tree to root of larger tree.

reasonable alternative:
union by height/rank

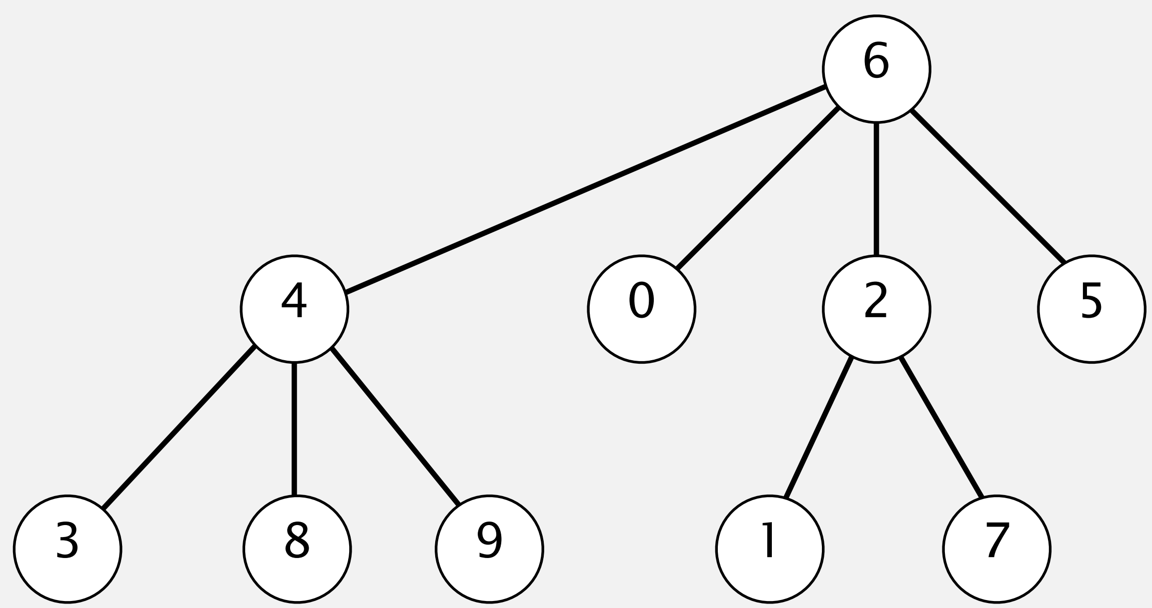


Weighted quick-union demo



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Weighted quick-union demo

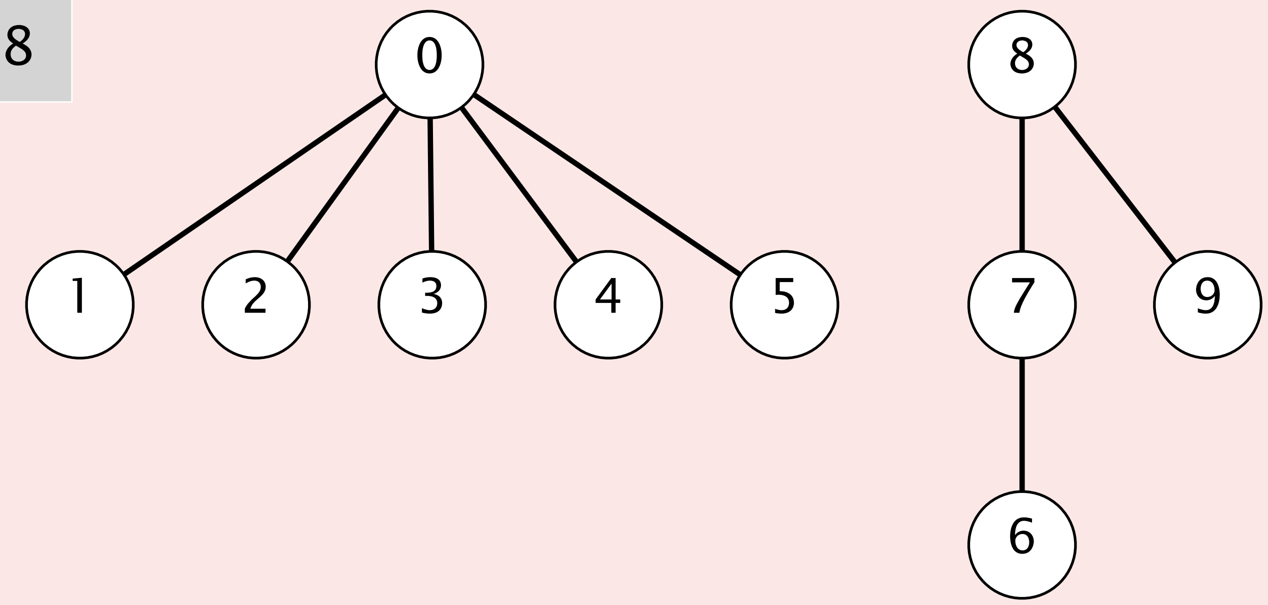


	0	1	2	3	4	5	6	7	8	9
id[]	6	2	6	4	6	6	6	2	4	4

Weighted quick-union quiz

Suppose that the `id[]` array during weighted quick union is:

	0	1	2	3	4	5	6	7	8	9
id[]	0	0	0	0	0	0	7	8	8	8



Which `id[]` entry changes when we apply the union operation to 2 and 6?

A. `id[0]`

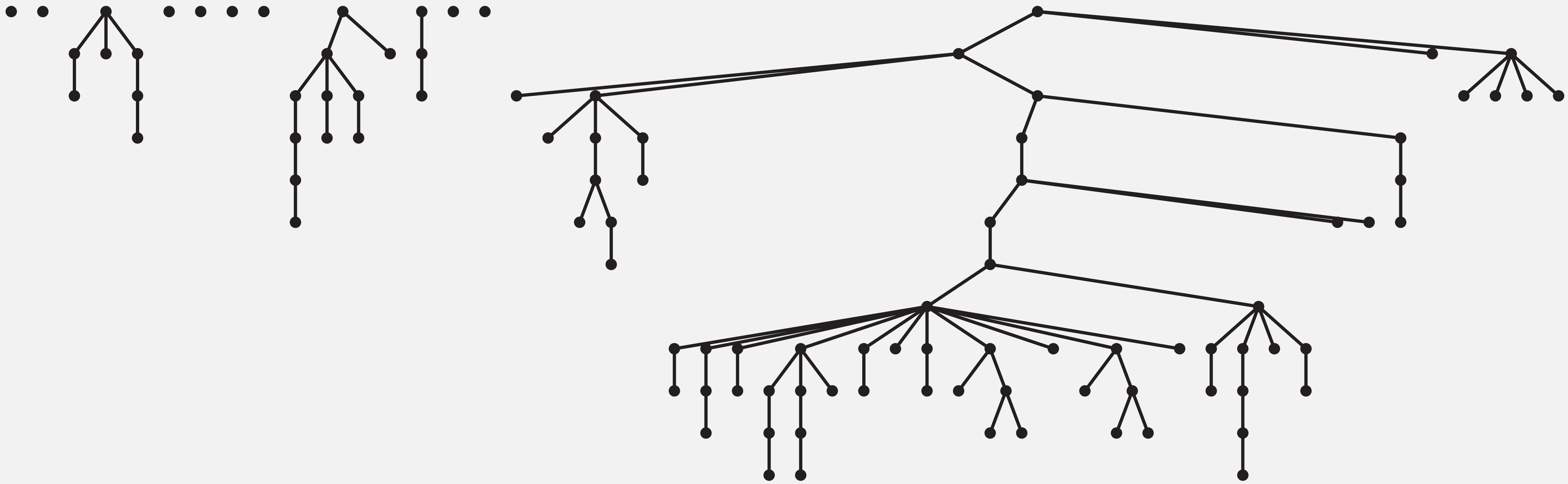
B. `id[2]`

C. `id[6]`

D. `id[8]`

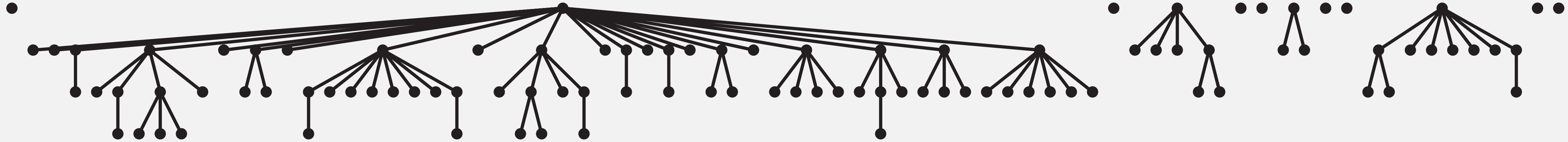
Quick-union and weighted quick-union example

quick-union



average distance to root: 5.11

weighted



average distance to root: 1.52

Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `sz[i]` to count number of elements in the tree rooted at `i`, initially `1`.

Find/connected. Identical to quick-union.

Union. Modify quick-union to:

- Link root of smaller tree to root of larger tree.
- Update the `sz[]` array.

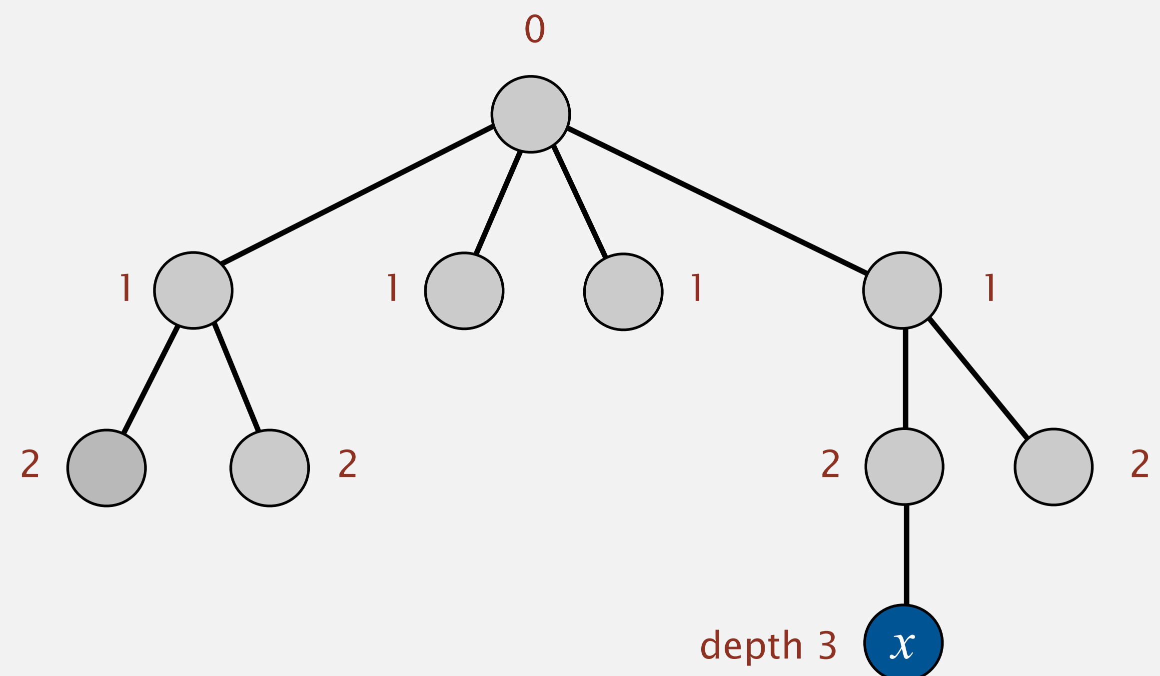
```
int i = find(p);
int j = find(q);
if (i == j) return;
if  (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else                { id[j] = i; sz[i] += sz[j]; }
```

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg N$. ← in computer science, lg means base-2 logarithm



N = 10
depth(x) = 3 ≤ lg N

Weighted quick-union analysis

Running time.

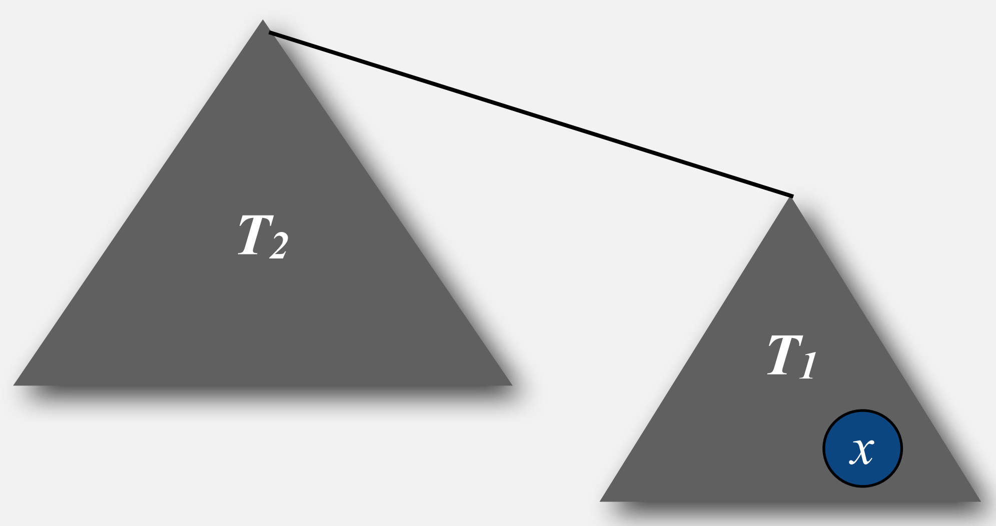
- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg N$. ← in computer science, lg means base-2 logarithm

Pf. What causes the depth of element x to increase?

Increases by 1 when tree T_1 containing x is merged into another tree T_2.

- The size of the tree containing x at least doubles since $|T_2| \geq |T_1|$.
- Size of tree containing x can double at most $\lg N$ times. Why?



$$\left.\begin{array}{c} 1 \\ 2 \\ 4 \\ 8 \\ 16 \\ \vdots \\ N \end{array}\right\} \lg N$$

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg N$.

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N
weighted QU	N	lg N †	lg N	lg N

† includes cost of finding two roots

Summary

Key point. Weighted quick union (and/or path compression) makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	N + M lg* N

order of growth for M union-find operations on a set of N elements

Ex. [10^9 unions and finds with 10^9 elements]

- WQUPC reduces time from 30 years to 6 seconds.
- Supercomputer won't help much; good algorithm enables solution.

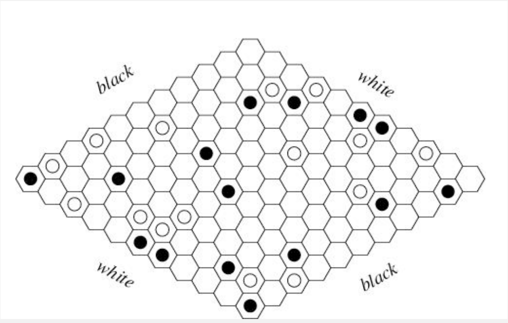
http://algs4.cs.princeton.edu

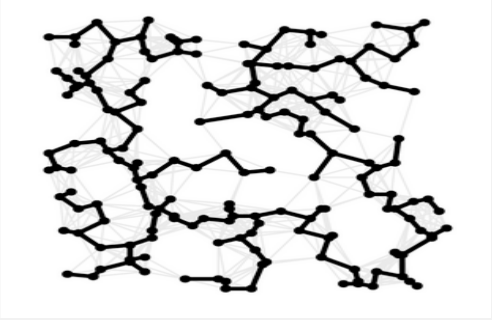
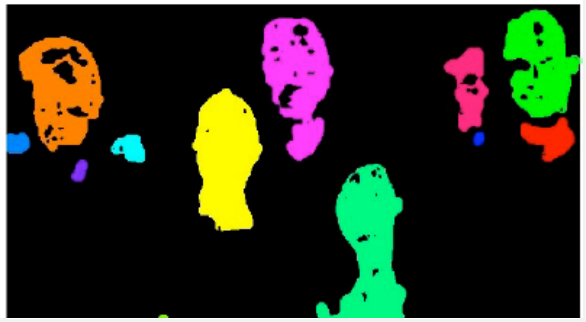
1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- ***applications***

Union-find applications

- Percolation.
- Games (Go, Hex).
- ✓ Dynamic connectivity.
- Least common ancestor.
- Equivalence of finite state automata.
- Hoshen-Kopelman algorithm in physics.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.
- Matlab's `bwlabel()` function in image processing.



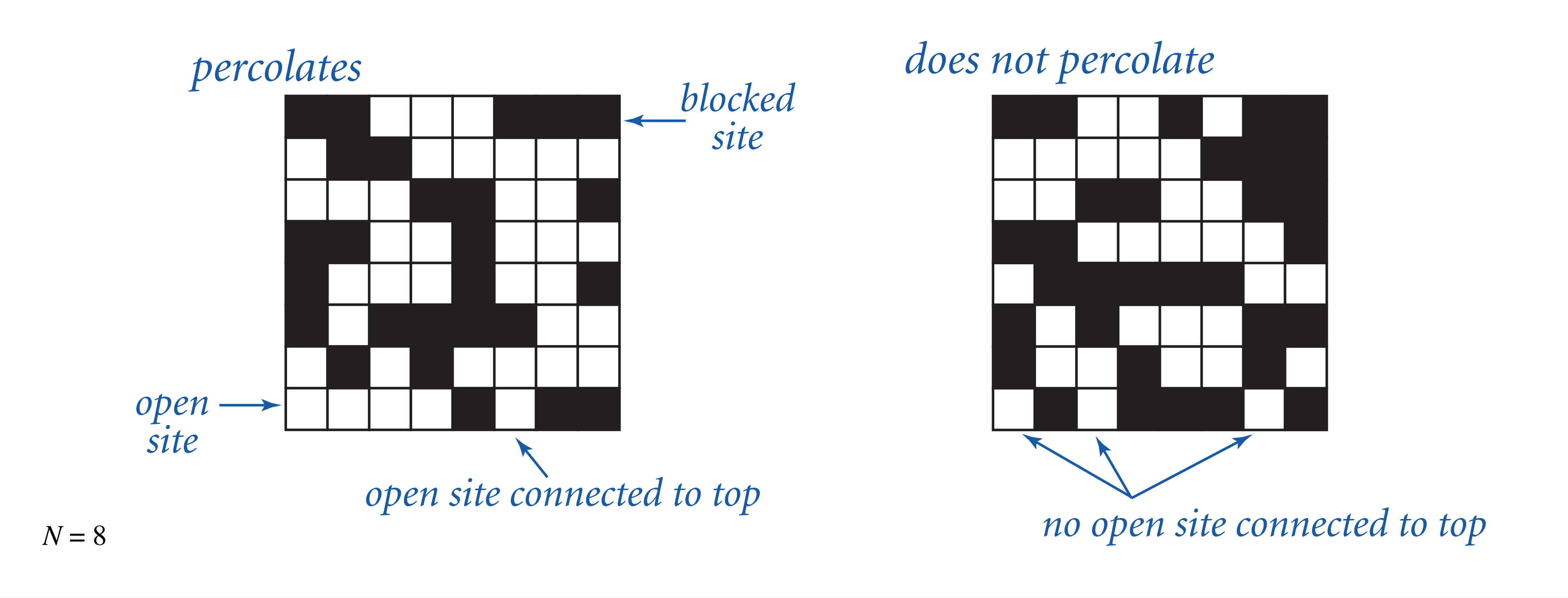



Percolation

An abstract model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (and blocked with probability $1 - p$).
- System percolates iff top and bottom are connected by open sites.

if and only if



Percolation

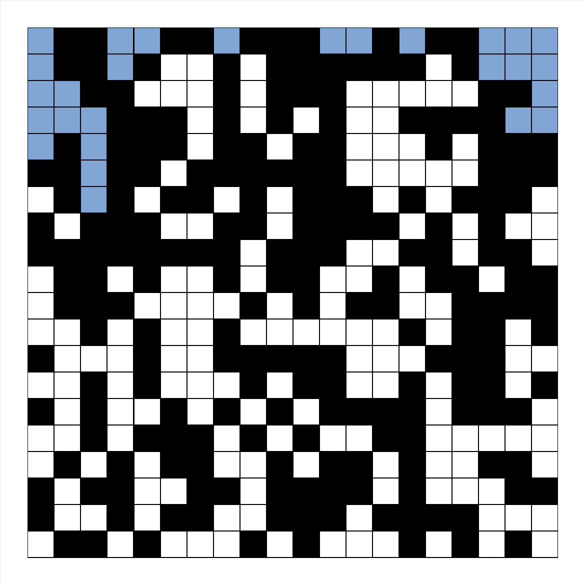
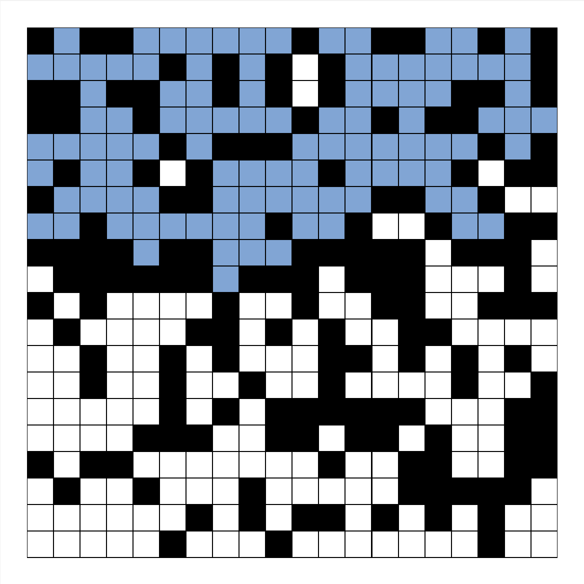
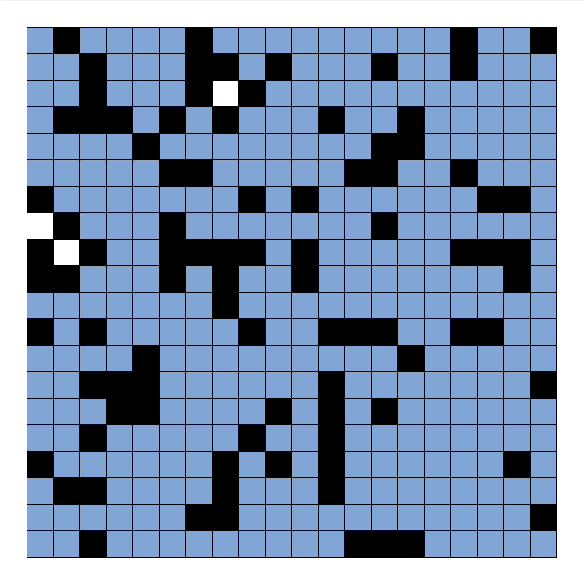
An abstract model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (and blocked with probability $1 - p$).
- System percolates iff top and bottom are connected by open sites.

model	system	vacant site	occupied site	percolates
electricity	material	conductor	insulated	conducts
fluid flow	material	empty	blocked	porous
social interaction	population	person	empty	communicates

Likelihood of percolation

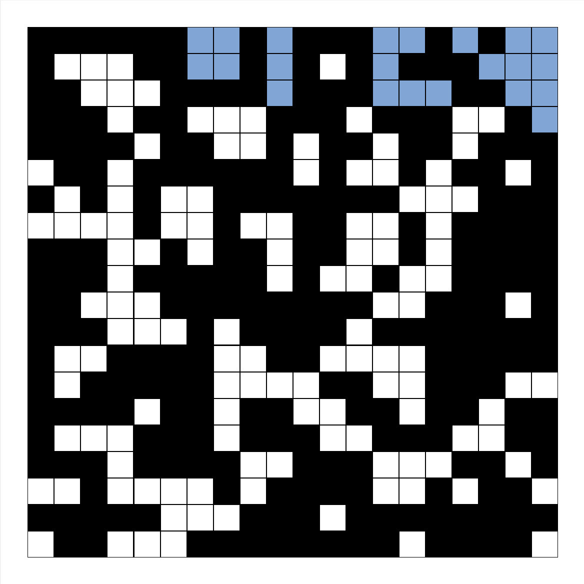
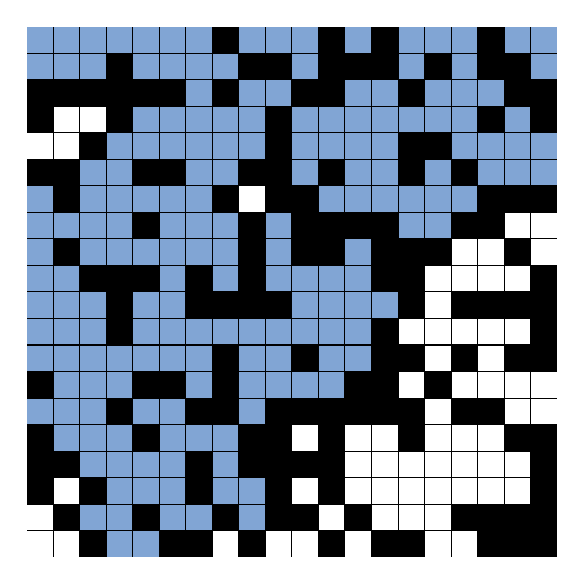
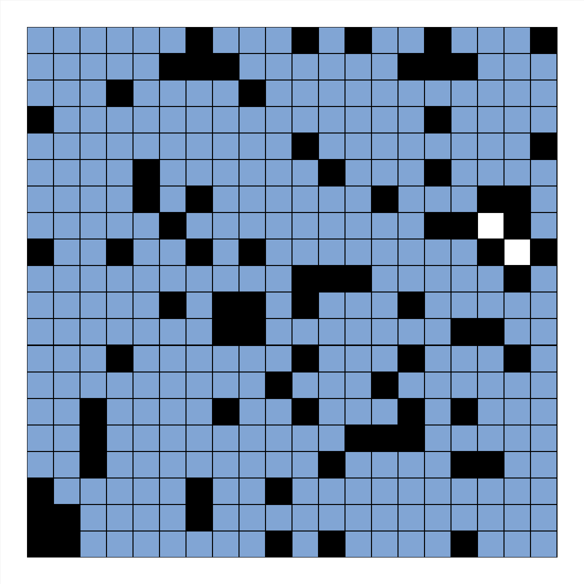
Depends on grid size N and site vacancy probability p.

p low (0.4)
does not percolate

p medium (0.6)
percolates?

p high (0.8)
percolates

empty open site
(not connected to top)

full open site
(connected to top)

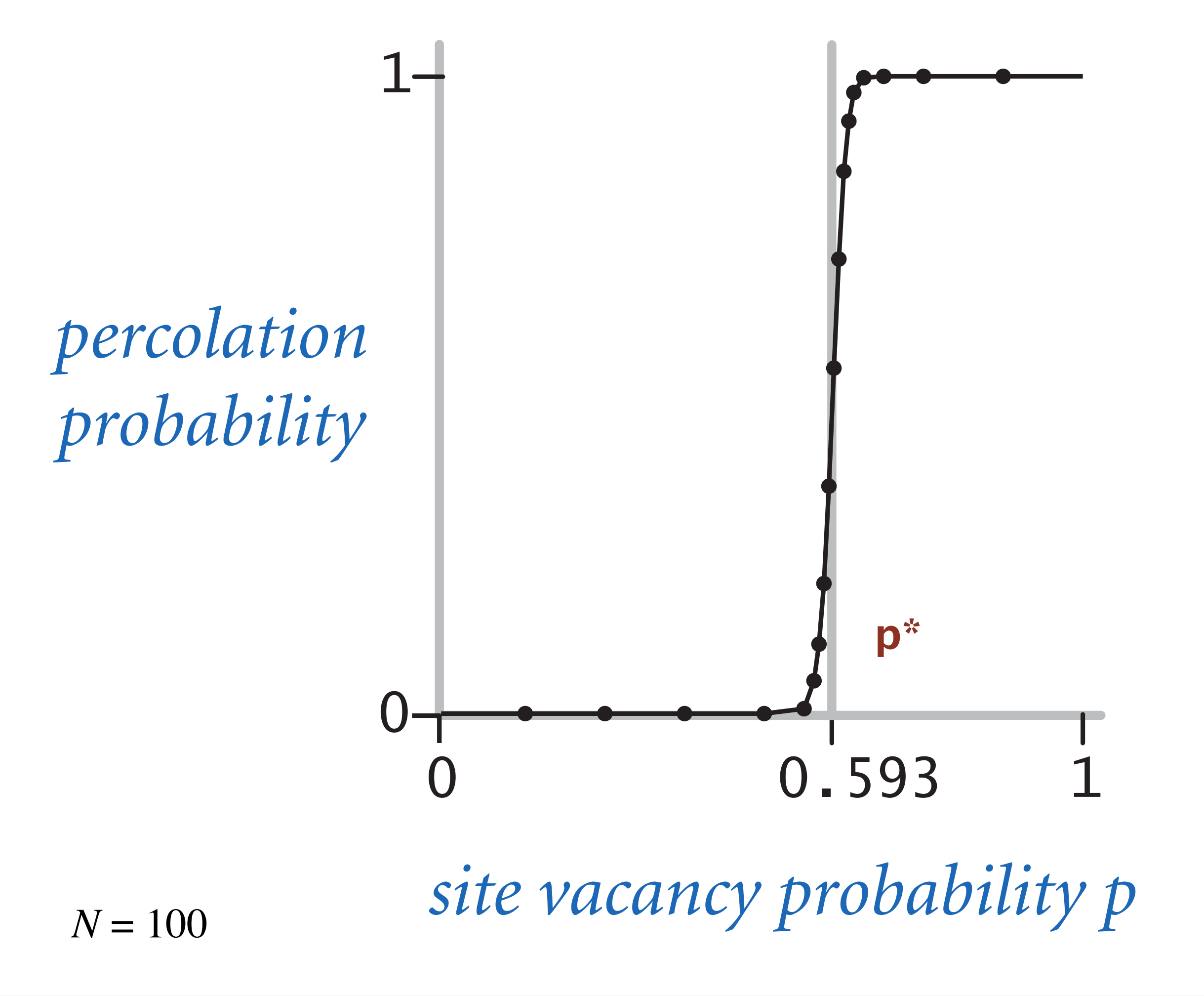
blocked site

Percolation phase transition

When N is large, theory guarantees a sharp threshold p^*.

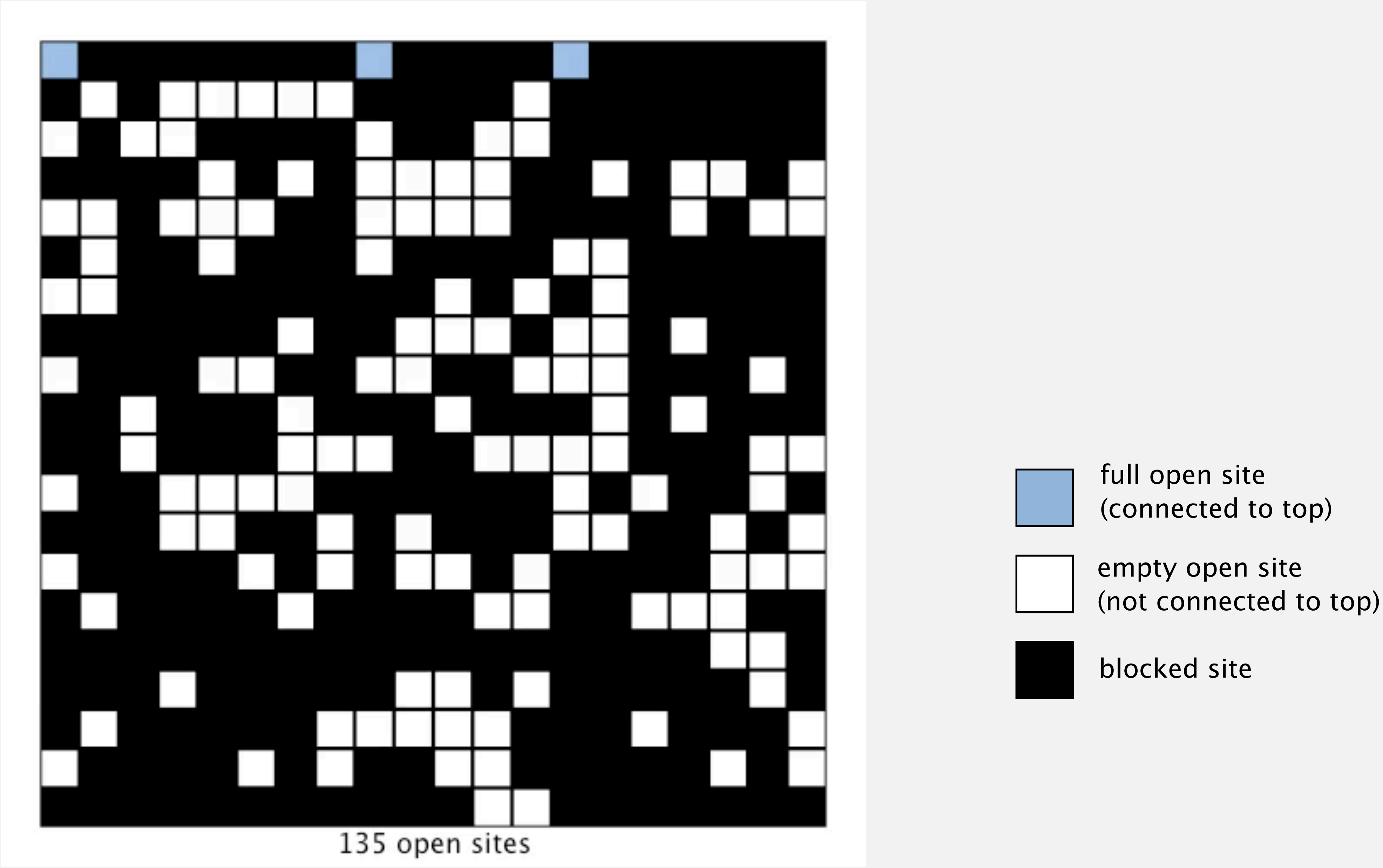
- $p > p^*$: almost certainly percolates.
- $p < p^*$: almost certainly does not percolate.

Q. What is the value of p^* ?



Monte Carlo simulation

- Initialize all sites in an N-by-N grid to be blocked.
- Declare random sites open until top connected to bottom.
- Vacancy percentage estimates p^*.



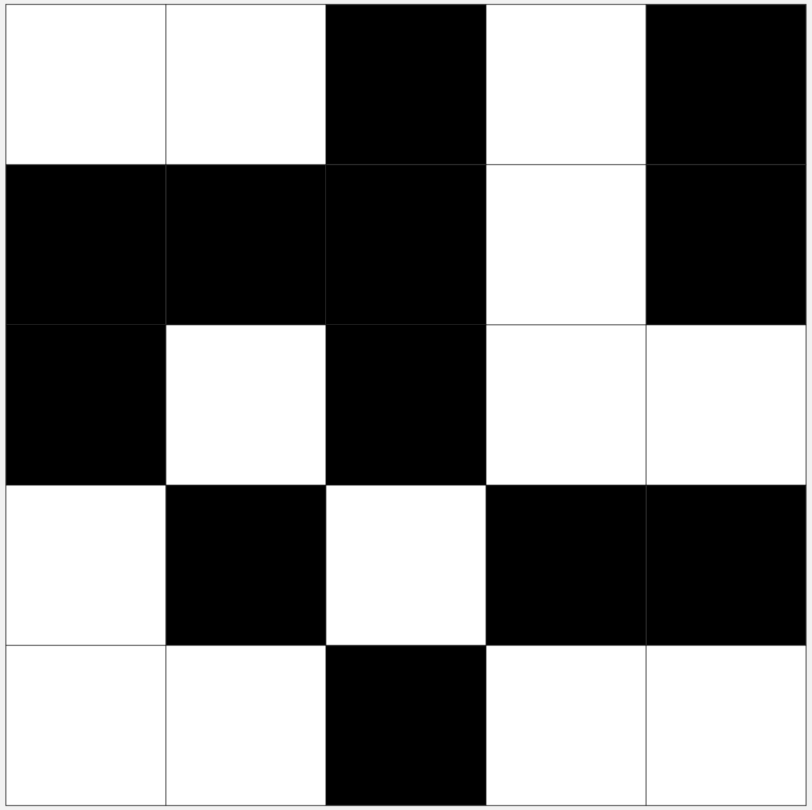
$N = 20$

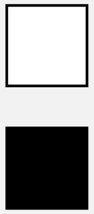
Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N-by-N system percolates?

A. Model as a dynamic connectivity problem and use union-find.

$N = 5$





open site

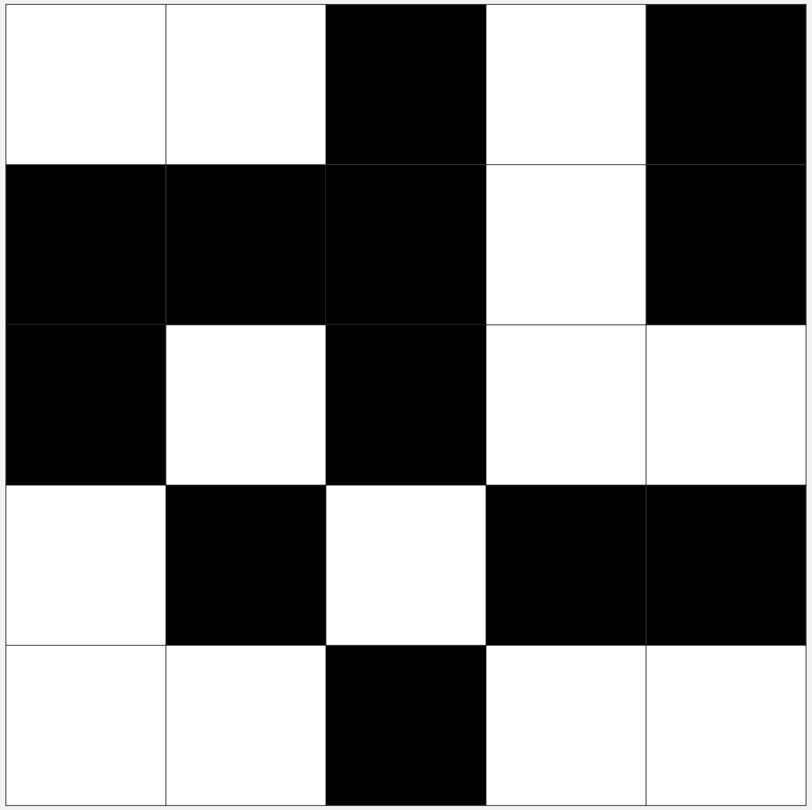
blocked site

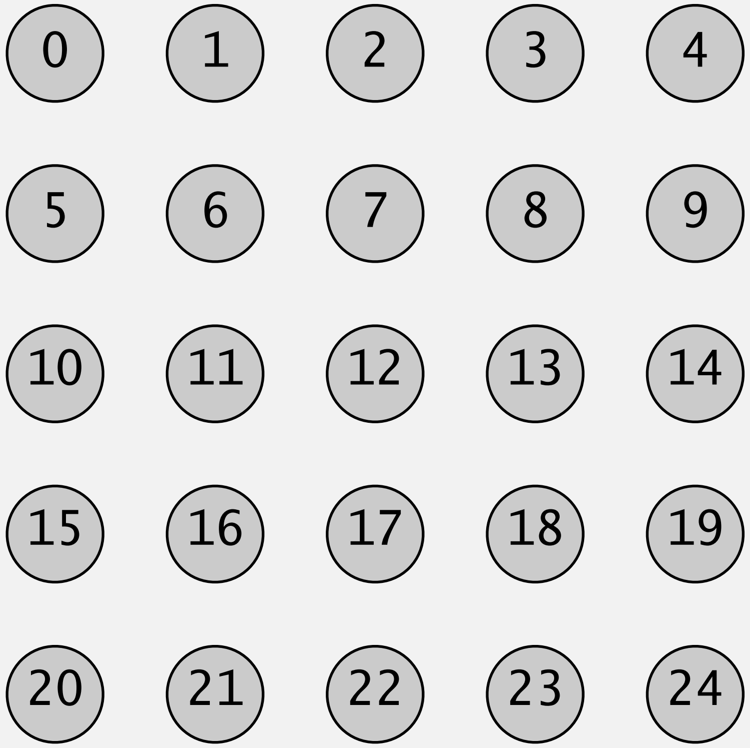
Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N-by-N system percolates?

- Create an element for each site and name them 0 to $N^2 - 1$.

$N = 5$





open site

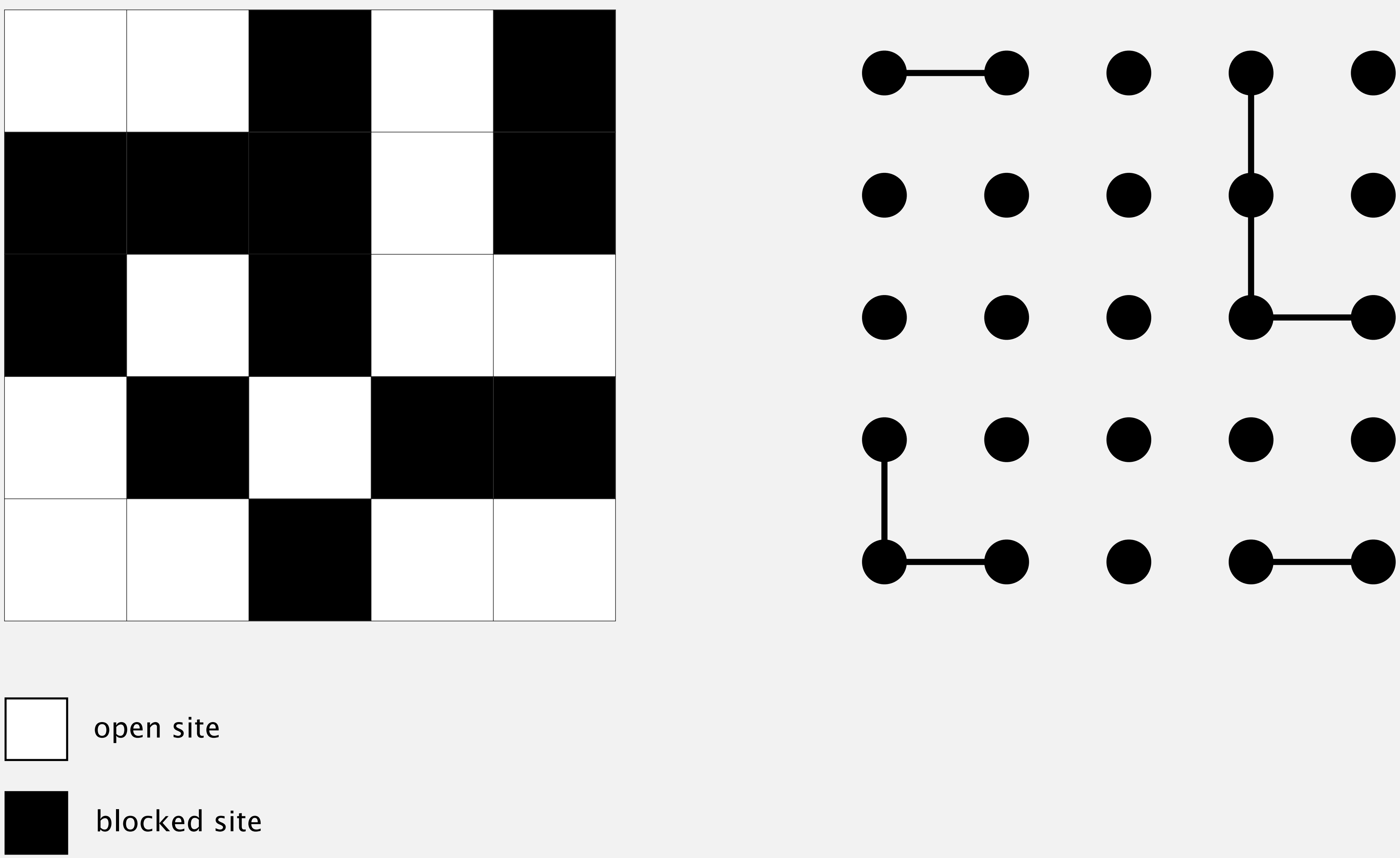
blocked site

Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N-by-N system percolates?

- Create an element for each site and name them 0 to $N^2 - 1$.
- Sites are in same component iff connected by open sites.

$N = 5$



open site

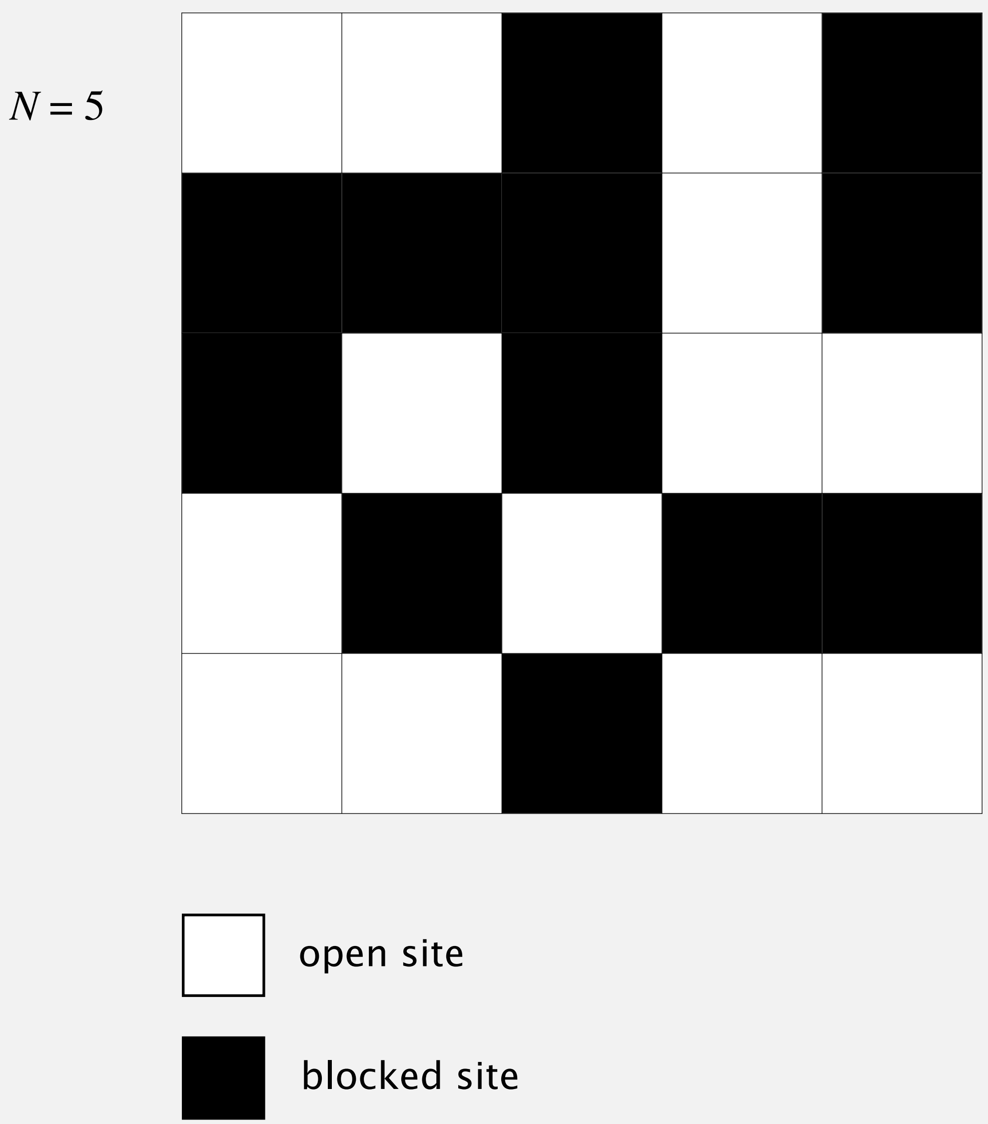
blocked site

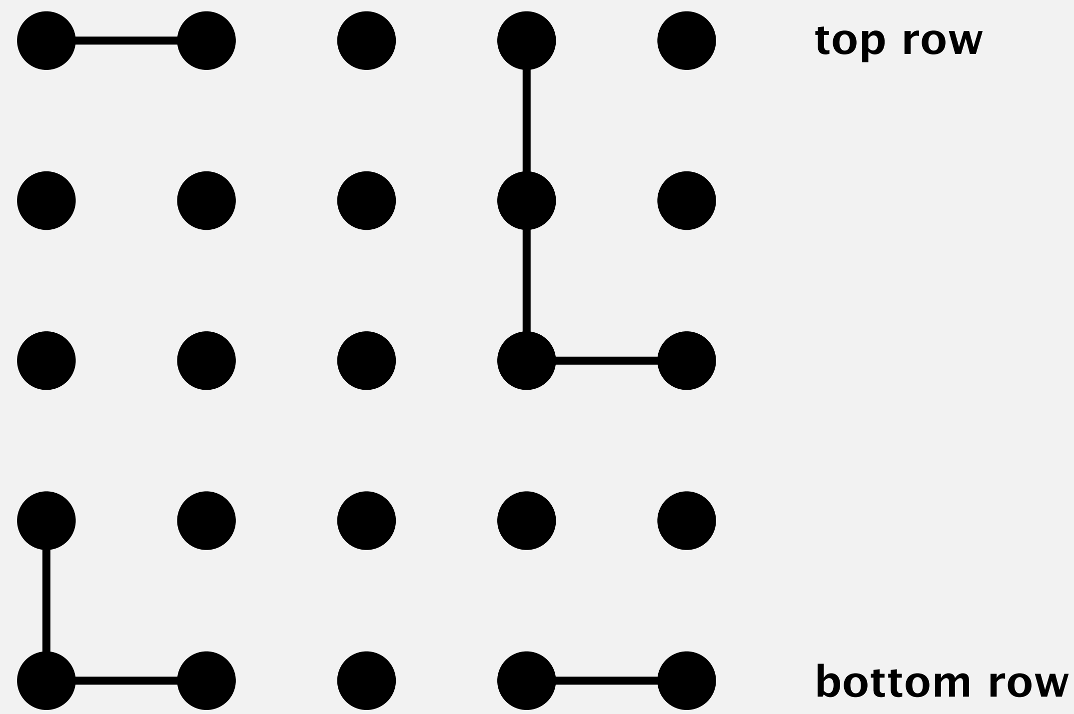
Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N-by-N system percolates?

- Create an element for each site and name them 0 to $N^2 - 1$.
- Sites are in same component iff connected by open sites.
- Percolates iff any site on bottom row is connected to any site on top row.

brute-force algorithm: N^2 calls to connected()



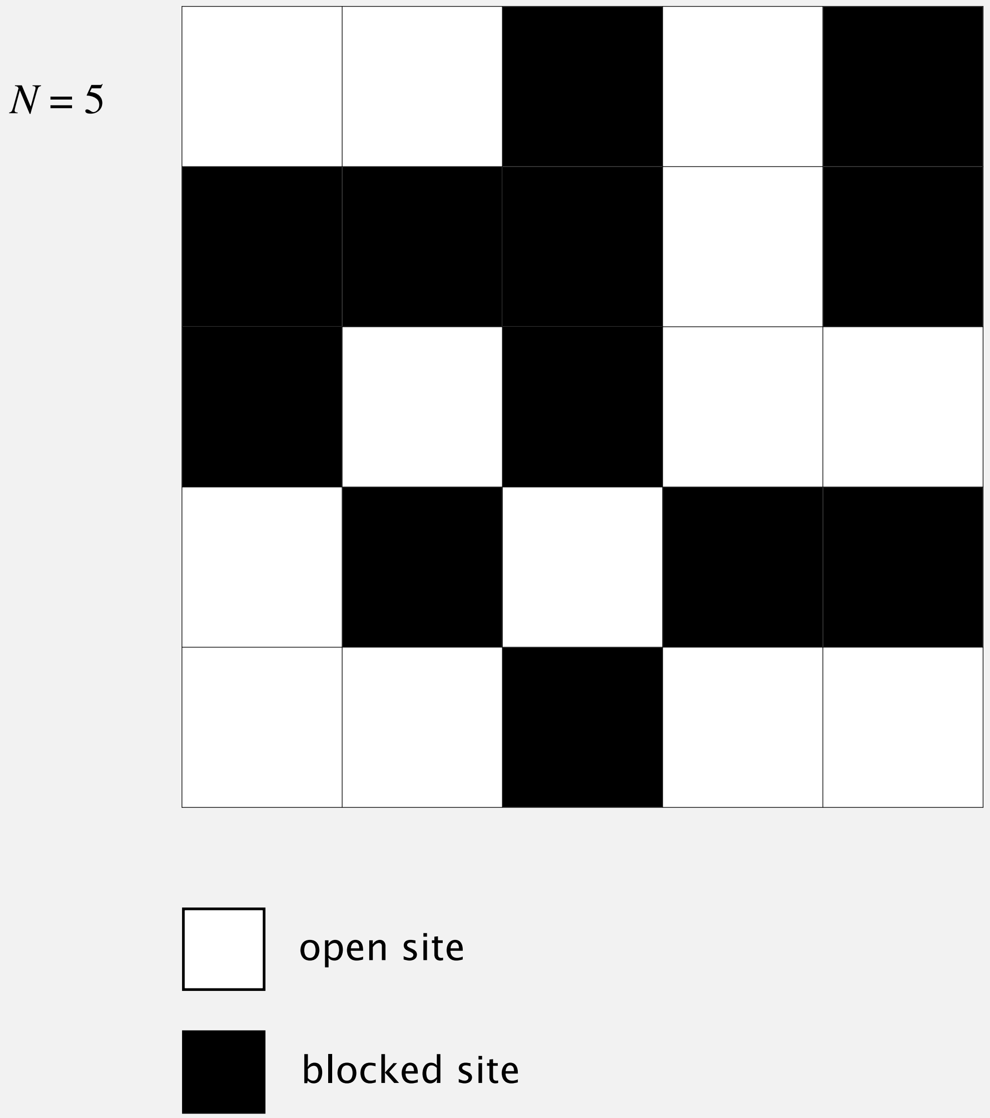


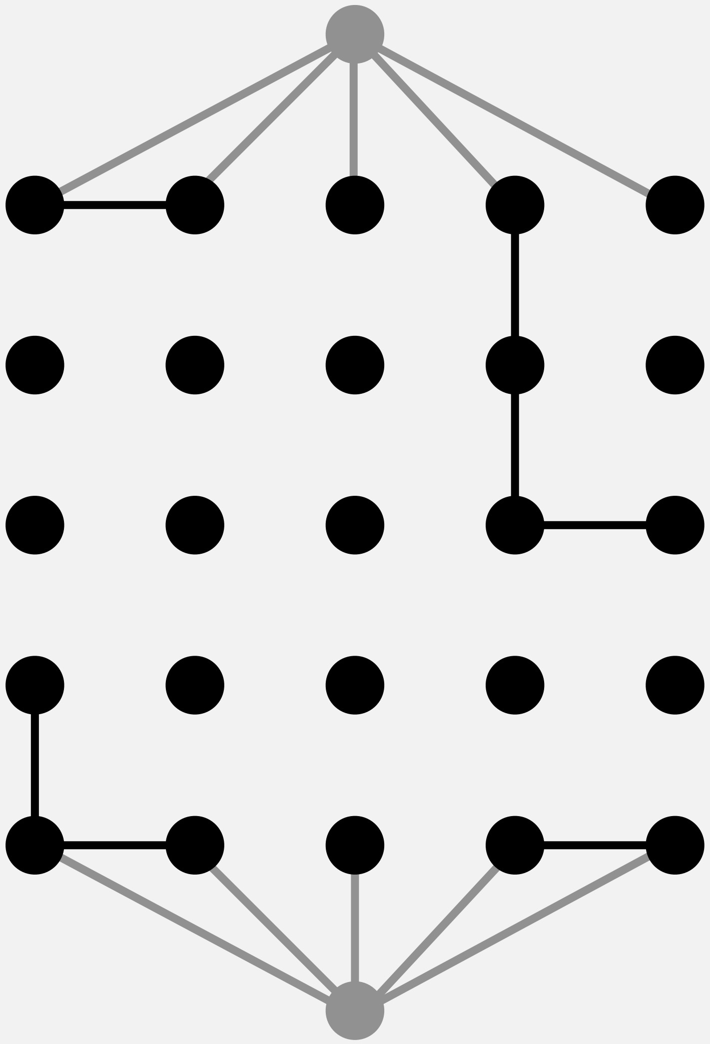
Dynamic connectivity solution to estimate percolation threshold

Clever trick. Introduce 2 virtual sites (and connections to top and bottom).

- Percolates iff virtual top site is connected to virtual bottom site.

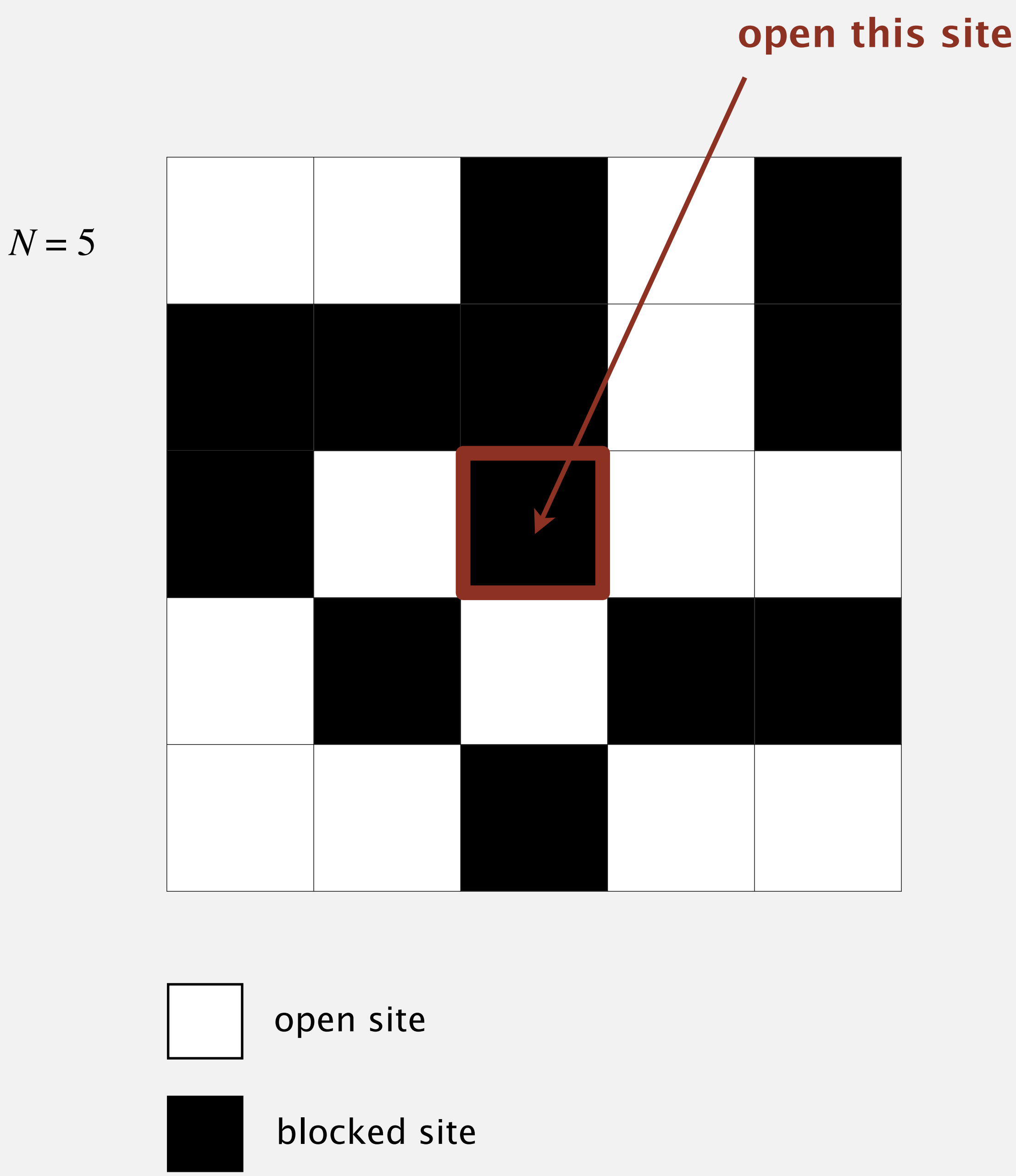
more efficient algorithm: only 1 call to connected()

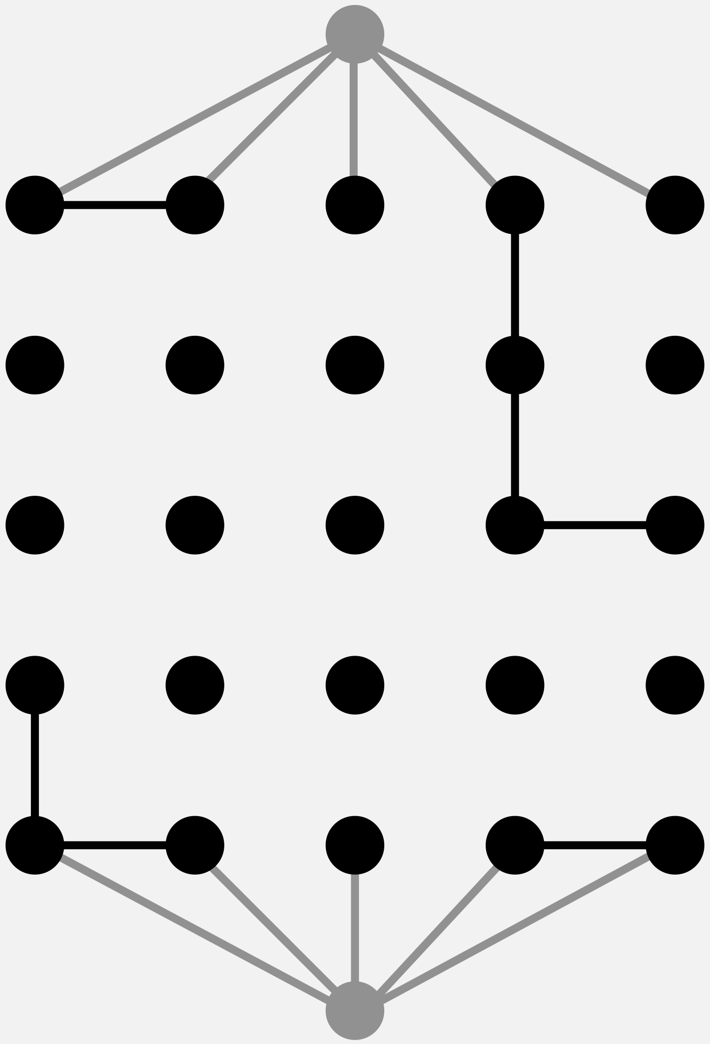




Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?



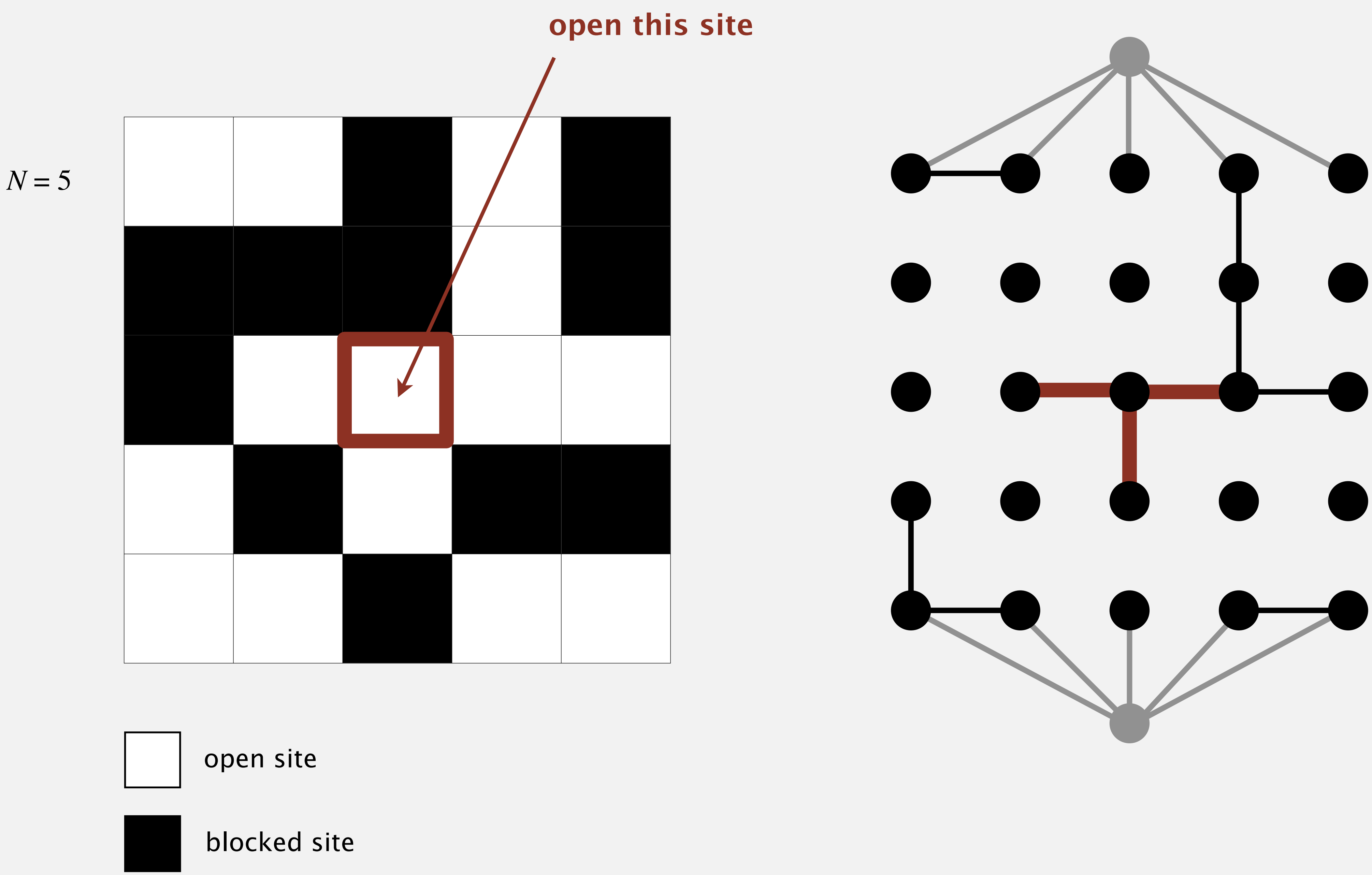


Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

A. Mark new site as open; connect it to all of its adjacent open sites.

up to 4 calls to `union()`

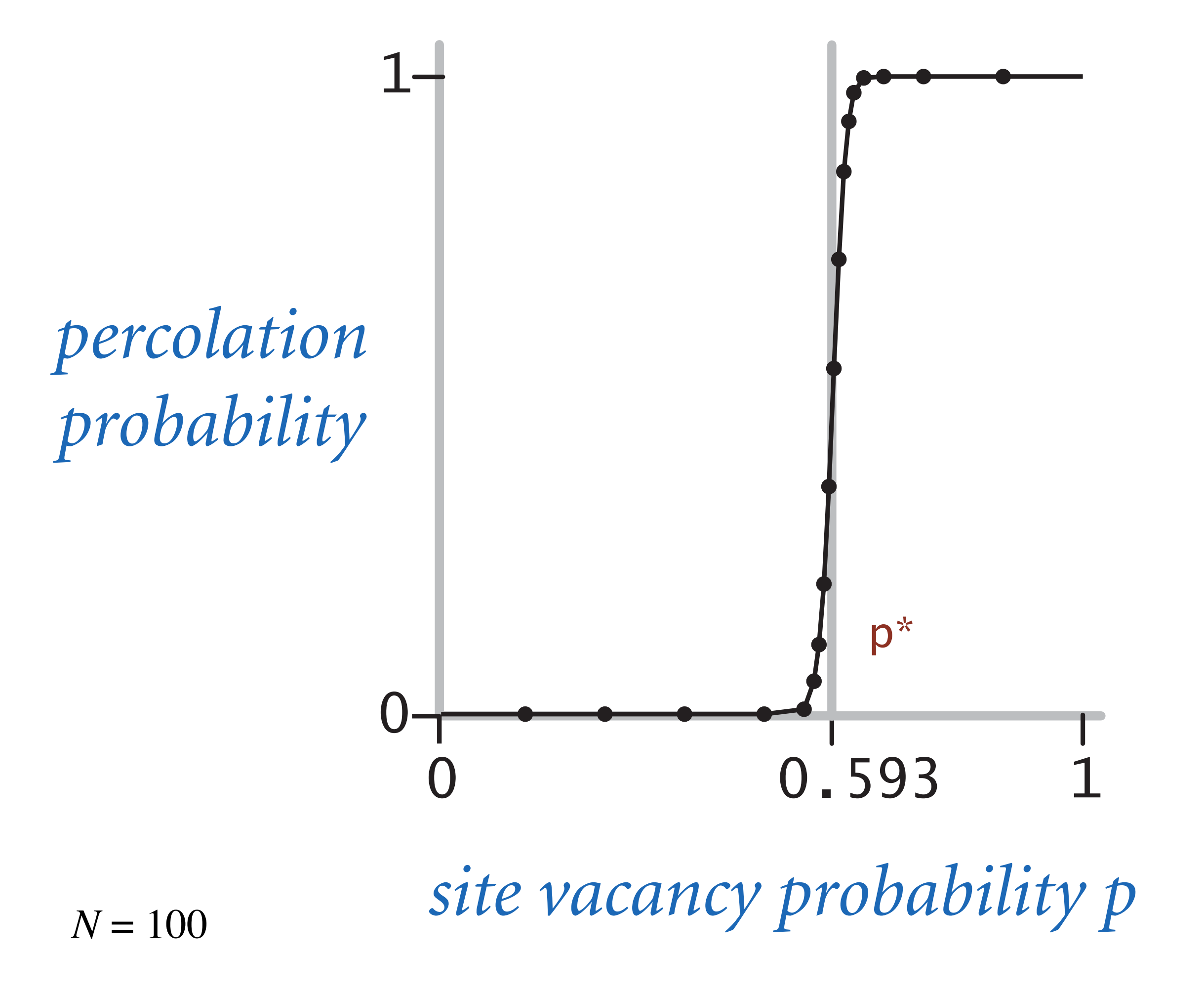


Percolation threshold

Q. What is percolation threshold p^* ?

A. About 0.592746 for large square lattices.

constant known only via simulation



Fast algorithm enables accurate answer to scientific question.

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.