

## ALGORITHMS AND DATA STRUCTURES

KEVIN WAYNE



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

#### 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
<b>data types</b>	stack, queue, bag, union-find, priority queue
<b>sorting</b>	quicksort, mergesort, heapsort, radix sorts
<b>searching</b>	BST, red-black BST, hash table
<b>graphs</b>	BFS, DFS, Prim, Kruskal, Dijkstra
<b>strings</b>	KMP, regular expressions, tries, data compression
<b>advanced</b>	B-tree, k-d 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, ...

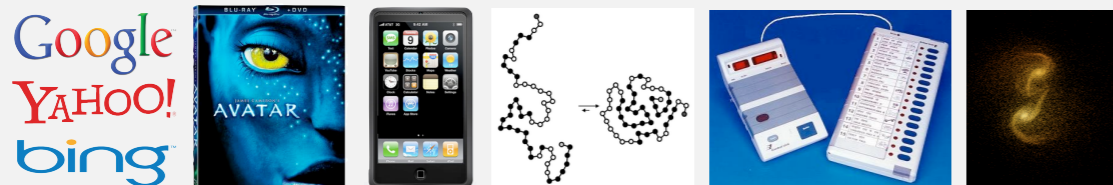
**Security.** Cell phones, e-commerce, voting machines, ...

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

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

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

⋮



#### Why study algorithms?

Their impact is broad and far-reaching.

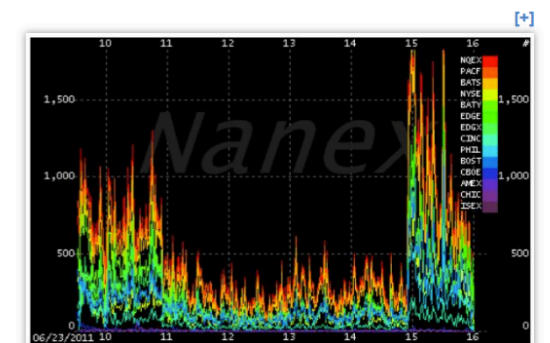
#### Mysterious algorithm was 4% of trading activity last week

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.



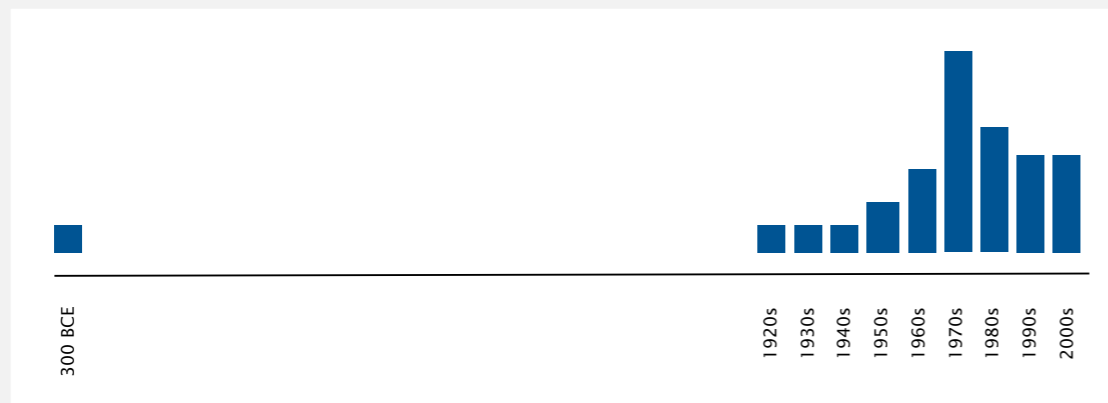
Generic high frequency trading chart (credit: Nanex)

"My guess is that the algo was testing the market, as high-frequency frequently does," says Jon Najarian, co-founder of TradeMonster.com. "As soon as they add bandwidth, the HFT crowd sees how quickly they can top out to create latency." ([Read More: Unclear What Caused Kraft Spike: Nanex Founder.](#))

## Why study algorithms?

### Old roots, new opportunities.

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



5

## 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



“ An algorithm must be seen to be believed. ” — Donald Knuth



6

## 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



7

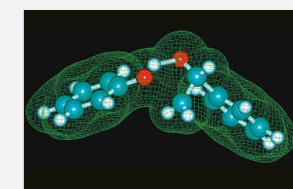
## Why study algorithms?

### 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)



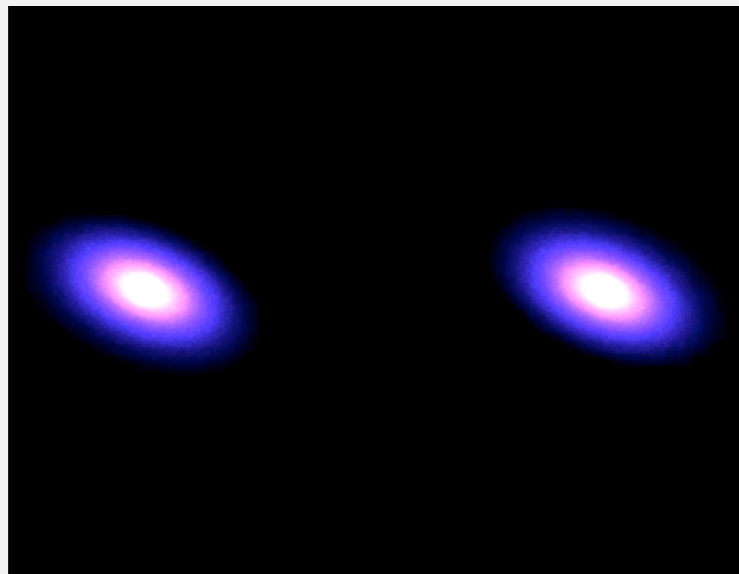
Martin Karplus, Michael Levitt, and Arieh Warshel



8

## Why study algorithms?

To solve problems that could not otherwise be addressed.



[http://www.youtube.com/watch?v=ua7YIN4eL\\_w](http://www.youtube.com/watch?v=ua7YIN4eL_w)

9

## Why study algorithms?

Everybody else is doing it.

```
% sort -rn PU2013-14.txt
 774 COS 126 General Computer Science
 615 ECO 100 Introduction to Microeconomics
 471 ECO 101 Introduction to Macroeconomics
 444 ENG 385 Children's Literature
 440 MAT 202 Linear Algebra with Applications
 414 COS 226 Algorithms and Data Structures
 405 MAT 201 Multivariable Calculus
 384 CHV 310 Practical Ethics
 344 REL 261 Christian Ethics and Modern Society
 320 PSY 101 Introduction to Psychology
 300 COS 217 Introduction to Programming Systems
 ...
```

10

## Why study algorithms?

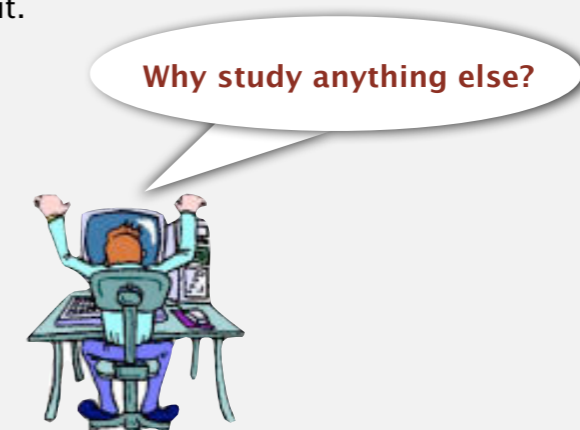
For fun and profit.



11

## 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.



12

## Lectures

**Traditional lectures.** Introduce new material.

**Electronic devices.** Permitted, but only to enhance lecture.



no



no



no

What	When	Where	Who	Office Hours
L01	MW 11-12:20	McCosh 10	Kevin Wayne	see web

13

## Lectures

**Traditional lectures.** Introduce new material.

**Flipped lectures.**

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



What	When	Where	Who	Office Hours
L01	MW 11-12:20	McCosh 10	Kevin Wayne	see web
L02	W 11-12:20	First 307	Josh Hug Andy Guna	see web

14

## Precepts

Discussion, problem-solving, background for assignments.

What	When	Where	Who	Office Hours
P01	Th 11-11:50	CS 102	Andy Guna †	see web
P02	Th 12:30-1:20	Bobst 105	Andy Guna †	see web
P03	Th 1:30-2:20	Bobst 105	Nevin Li	see web
P04	F 10-10:50	Bobst 105	Jennifer Guo	see web
P05	F 11-11:50	Bobst 105	Madhu Jayakumar	see web
P05A	F 11-11:50	Sherrerd 001	Ruth Dannenfelser	see web
P06	F 2:30-3:20	Friend 108	Chris Eubank	see web
P06A	F 2:30-3:20	Friend 111	TBA	see web
P06B	F 2:30-3:20	Friend 109	Josh Hug †	see web
P07	F 3:30-4:20	Friend 108	Josh Hug †	see web

↑  
likely to change

† lead preceptor

15

## Coursework and grading

**Programming assignments.** 45%

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

**Exercises.** 10%

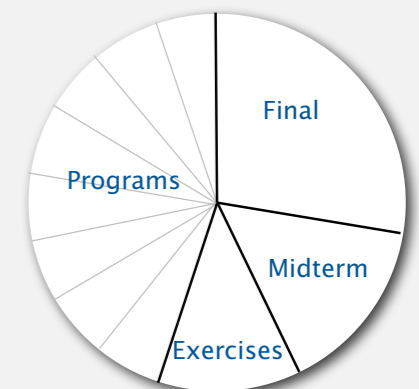
- Due on Sundays at 11pm in Blackboard.
- Collaboration/lateness policies: see web.

**Exams.** 15% + 30%

- Midterm (in class on Wednesday, March 12).
- Final (to be scheduled by Registrar).

**Staff discretion.** [adjust borderline cases]

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



16

## Resources (textbook)

**Required reading.** Algorithms 4<sup>th</sup> edition by R. Sedgwick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.



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.

17

## Resources (web)

**Course content.**

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

PRINCETON UNIVERSITY  
COMPUTER SCIENCE 226  
ALGORITHMS AND DATA STRUCTURES  
SPRING 2014

Course Information | Lectures | Flipped | Precepts | 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, 4TH EDITION

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

**Textbook.** The textbook *Algorithms, 4th Edition* by Robert Sedgwick 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>

18

## Resources (web)

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

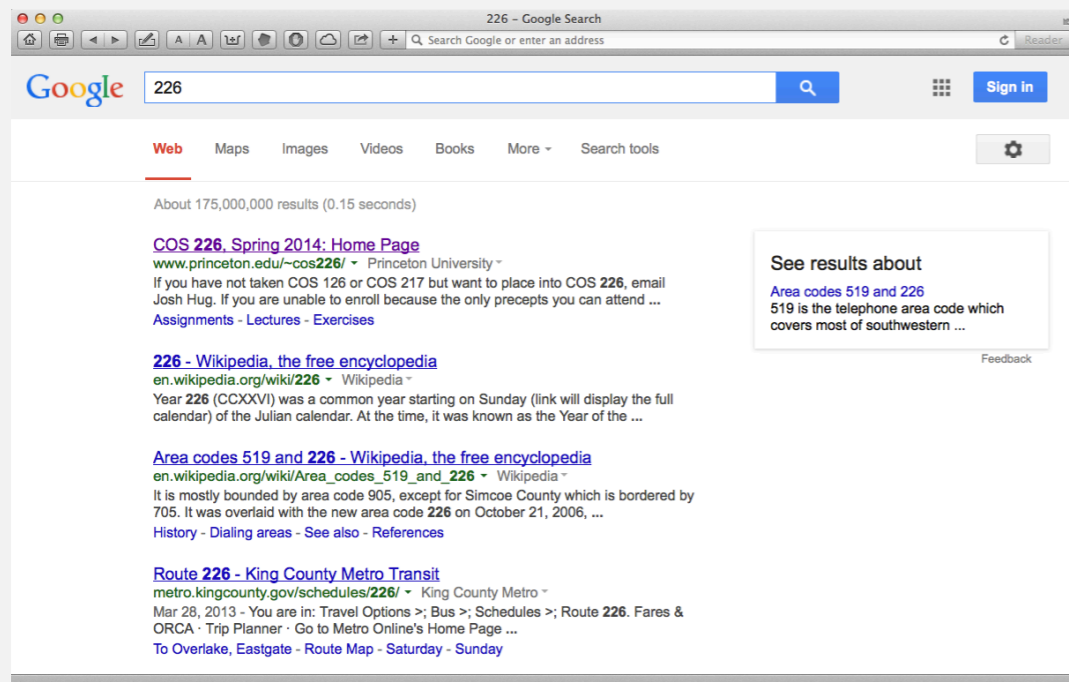
19

## Resources (web)

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

20

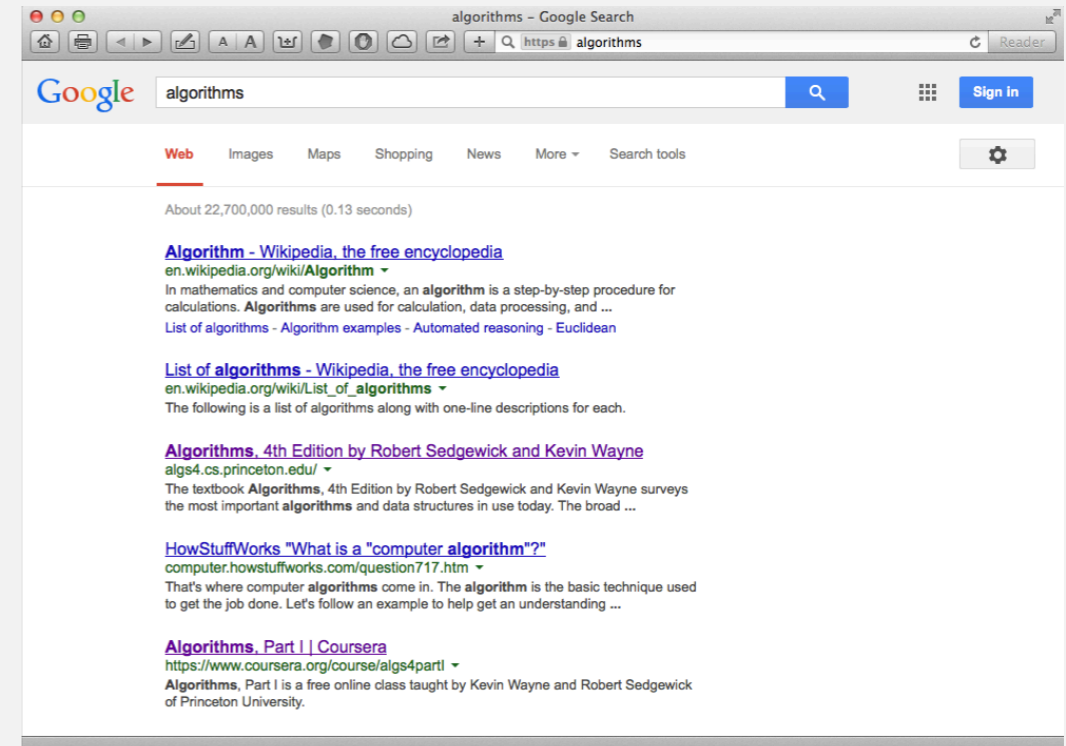
## Resources (web)



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

21

## Resources (web)



22

## Where to get help?

### Piazza discussion forum.

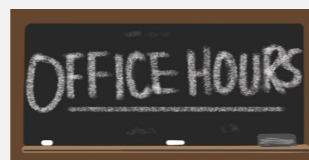
- Low latency, low bandwidth.
- Mark solution-revealing questions as private.



<http://piazza.com/princeton/spring2014/cos226>

### Office hours.

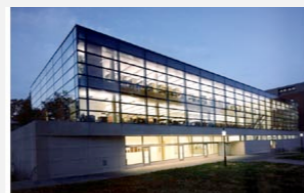
- High bandwidth, high latency.
- See web for schedule.



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

### Computing laboratory.

- Undergrad lab TAs in Friend 017.
- For help with debugging.
- See web for schedule.



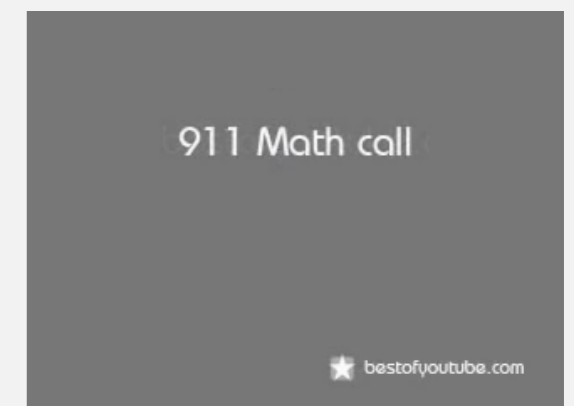
<http://www.princeton.edu/~cos226>

23

## Where not to get help?



<http://world.edu/academic-plagiarism>



<http://www.youtube.com/watch?v=FT4NOe4vt0M>

24

## What's ahead?

---

Lecture 1. [today] Union find.

Lecture 2. [Wednesday] Analysis of algorithms.

Flipped lecture 1. [Wednesday] Watch video beforehand.

Precept 1. [Thursday/Friday] Meets this week.



Exercise 1. Due via Bb submission at 11pm on Sunday.

Assignment 1. Due via electronic submission at 11pm on Tuesday.

← protip: start early

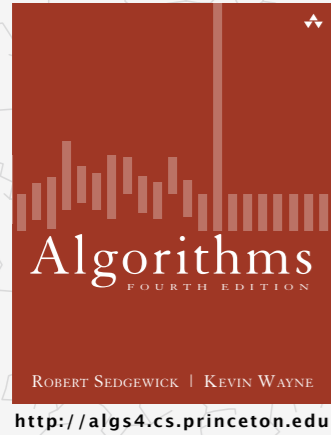
Right course? See me.

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

Not registered? Go to any precept this week.

Change precept? Use SCORE.

← see Colleen Kenny-McGinley  
in CS 210 if the only precepts  
you can attend are closed



## 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.



## 1.5 UNION-FIND

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

## Dynamic connectivity problem

### Given a set of N objects, support two operation:

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

*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?* ✓

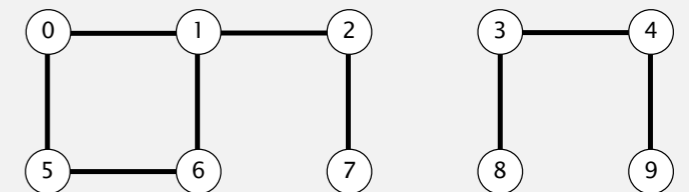
*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

*connect 1 and 0*

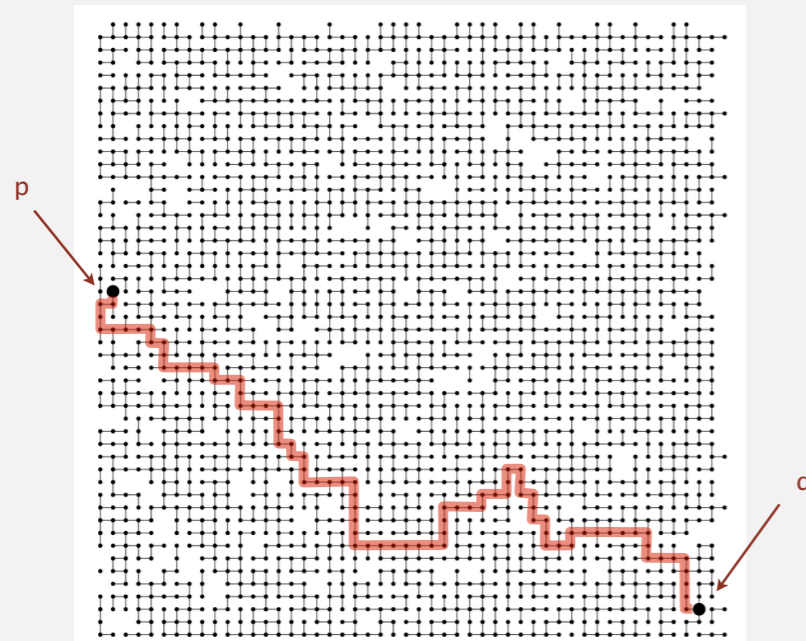
*are 0 and 7 connected?* ✓





## A larger connectivity example

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



A. Yes.

5

## Modeling the objects

Applications involve manipulating objects 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 objects 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)

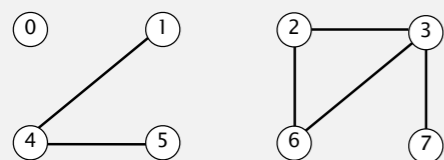
6

## 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 objects that are mutually connected.



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

3 connected components

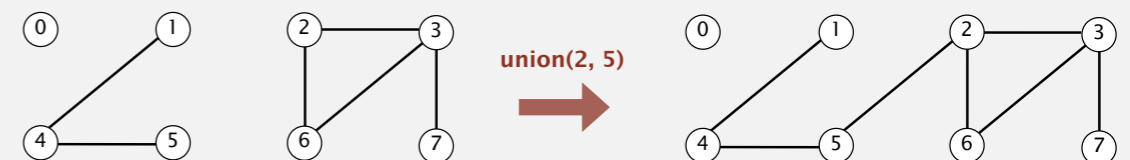
7

## Implementing the operations

Find. In which component is object  $p$ ?

Connected. Are objects  $p$  and  $q$  in the same component?

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



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

3 connected components

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

2 connected components

8

## Union-find data type (API)

**Goal.** Design efficient data structure for union-find.

- Number of objects  $N$  can be huge.
- Number of operations  $M$  can be huge.
- Union and find operations may be intermixed.

```
public class UF
{
    UF(int N)                initialize union-find data structure
                             with N singleton objects (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()

9

## Dynamic-connectivity client

- Read in number of objects  $N$  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

10

## 1.5 UNION-FIND

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



## Quick-find [eager approach]

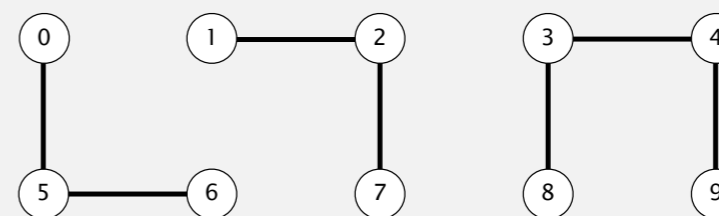
**Data structure.**

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

if and only if

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



12



## 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

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

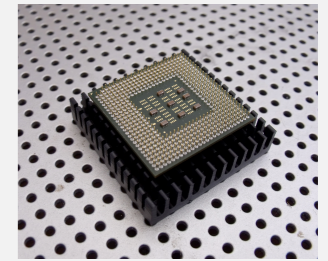
quadratic

## 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!

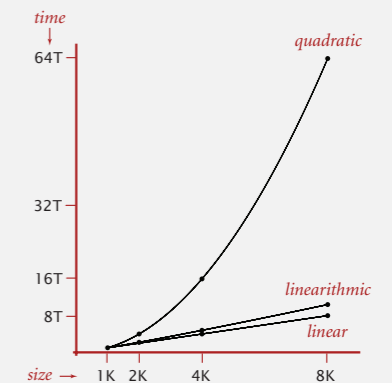


**Ex. Huge problem for quick-find.**

- $10^9$  union commands on  $10^9$  objects.
- 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!



## 1.5 UNION-FIND

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



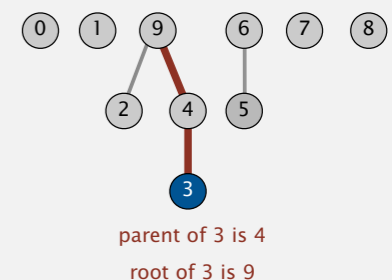
## Quick-union [lazy approach]

**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



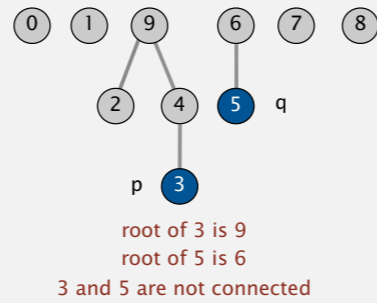
parent of 3 is 4  
root of 3 is 9

## 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
<code>id[]</code>	0	1	9	4	9	6	6	7	8	9



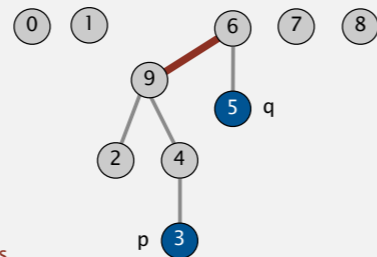
**Find.** What is the root of `p`?

**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
<code>id[]</code>	0	1	9	4	9	6	6	7	8	6

↑  
only one value changes



21

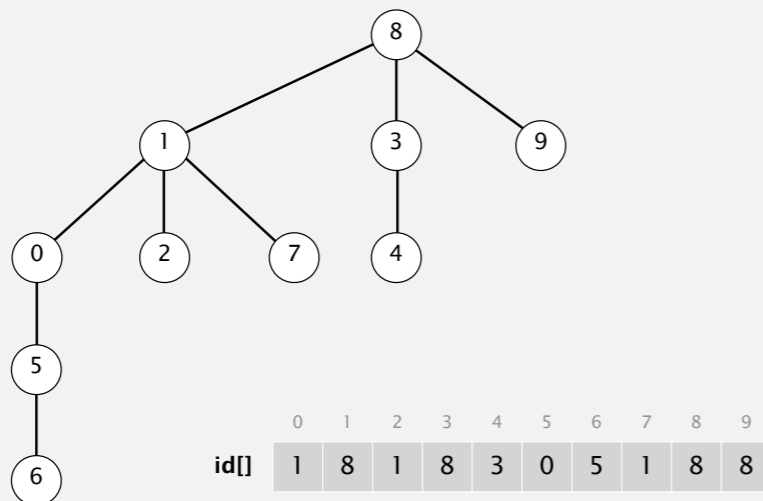
## Quick-union demo



	0	1	2	3	4	5	6	7	8	9
<code>id[]</code>	0	1	2	3	4	5	6	7	8	9

22

## Quick-union demo



## 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 i)
    {
        while (i != id[i]) i = id[i];
        return i;
    }

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

```

← set id of each object to itself (N array accesses)

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

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

24

## 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 roots

### Quick-find defect.

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

### Quick-union defect.

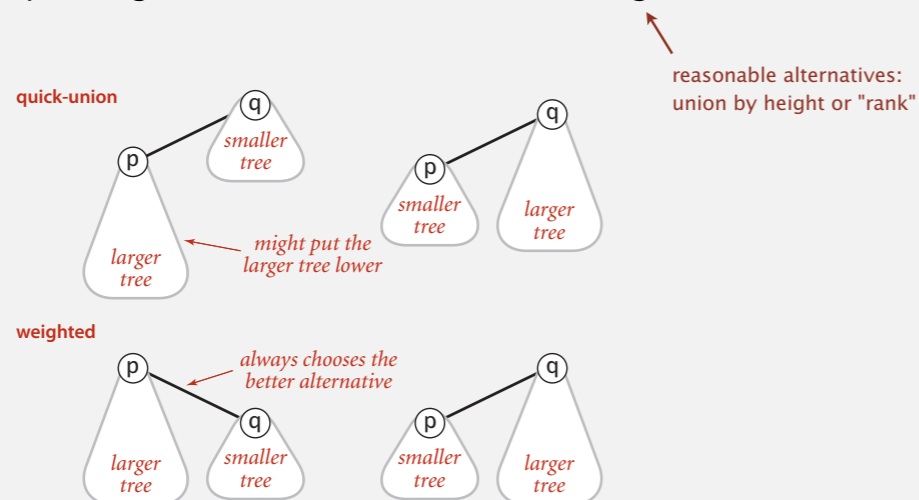
- Trees can get tall.
- Find/connected too expensive (could be  $N$  array accesses).

25

## Improvement 1: weighting

### Weighted quick-union.

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



27

## 1.5 UNION-FIND

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

ROBERT SEDGWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

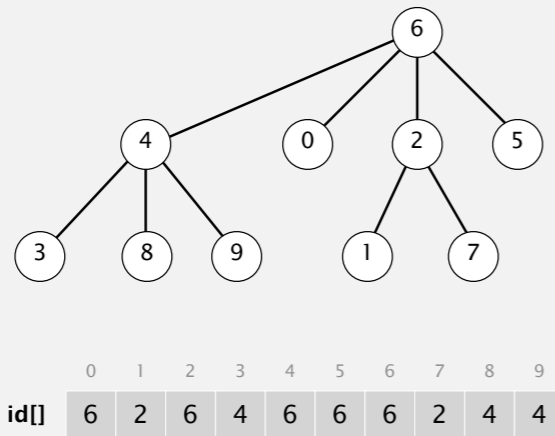
## Weighted quick-union demo



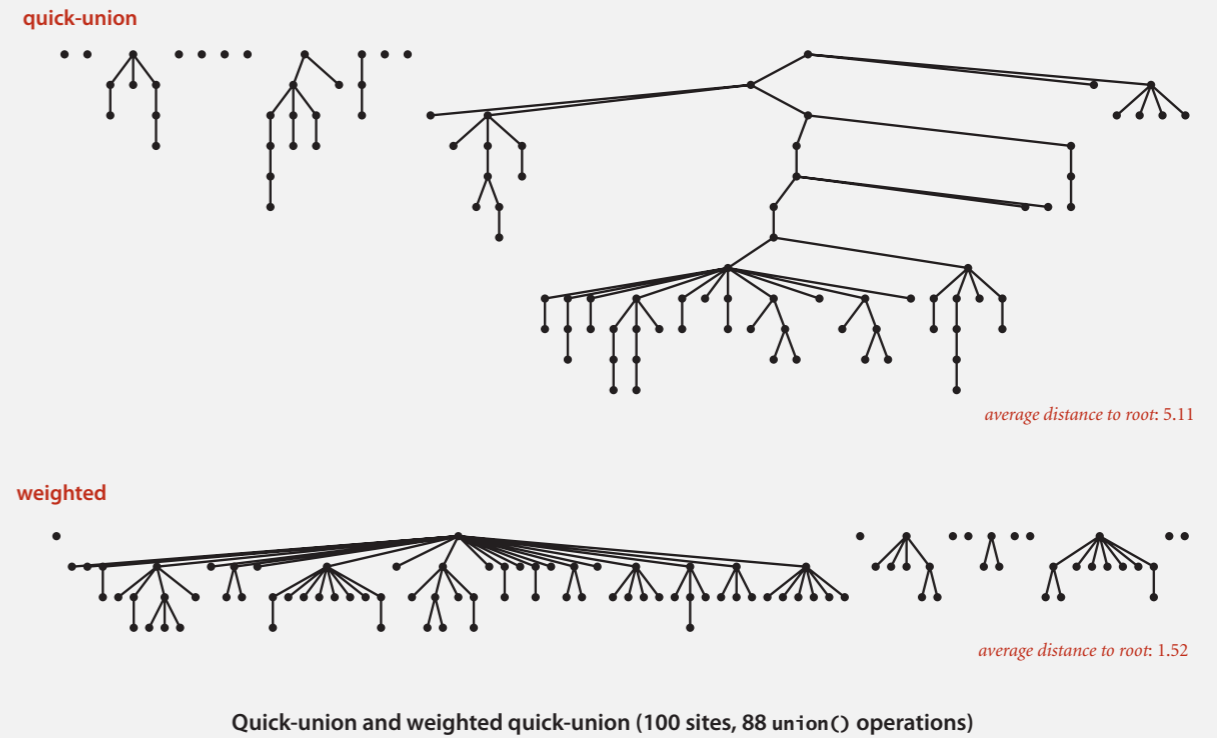
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

28

## Weighted quick-union demo



## Quick-union and weighted quick-union example



## Weighted quick-union: Java implementation

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

**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.

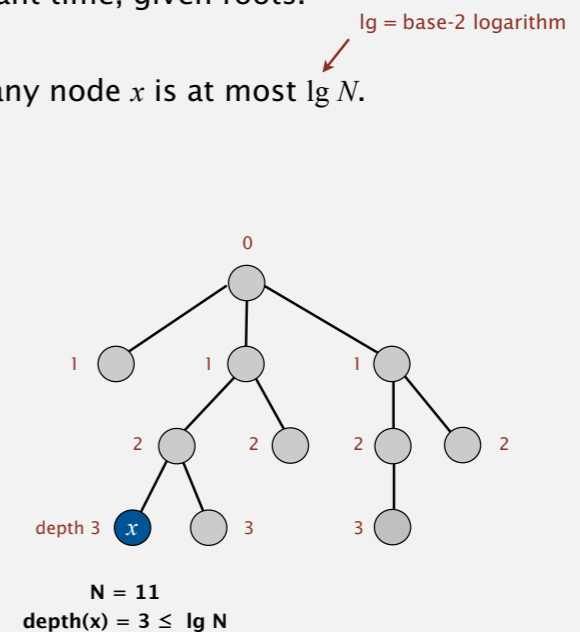
```
int i = find(p);
int j = find(q);
if (i == j) return;
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else { id[j] = i; sz[i] += sz[j]; }
```

## Weighted quick-union analysis

**Running time.**

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

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



## Weighted quick-union analysis

### Running time.

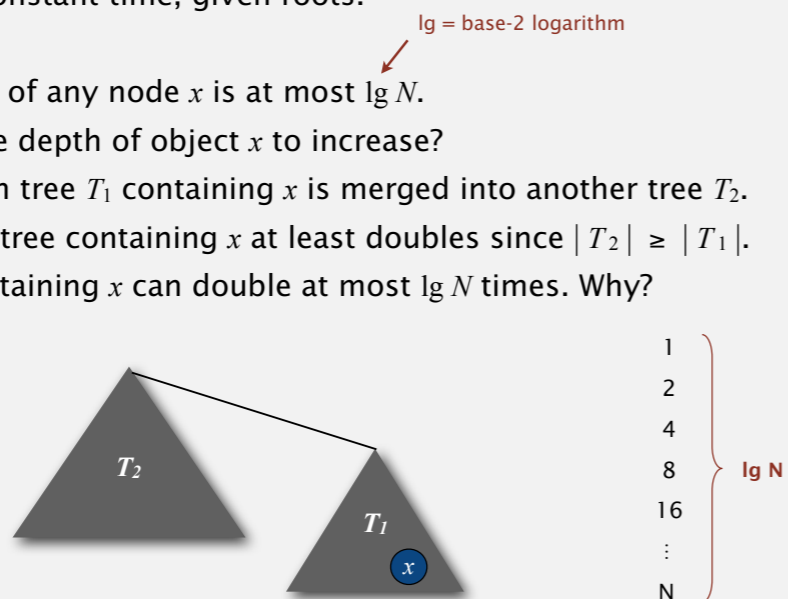
- Find: takes time proportional to depth of  $p$ .
- Union: takes constant time, given roots.

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

**Pf.** What causes the depth of object  $x$  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?



33

## Weighted quick-union analysis

### Running time.

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

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

algorithm	initialize	union	find	connected
<b>quick-find</b>	N	N	1	1
<b>quick-union</b>	N	N †	N	N
<b>weighted QU</b>	N	$\lg N$ †	$\lg N$	$\lg N$

† includes cost of finding roots

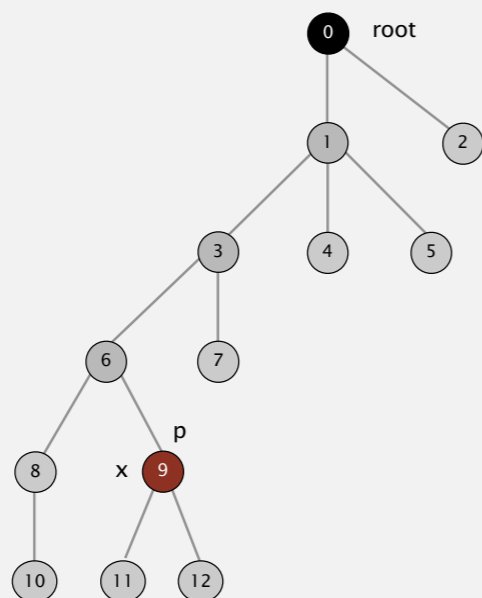
**Q.** Stop at guaranteed acceptable performance?

**A.** No, easy to improve further.

34

## Improvement 2: path compression

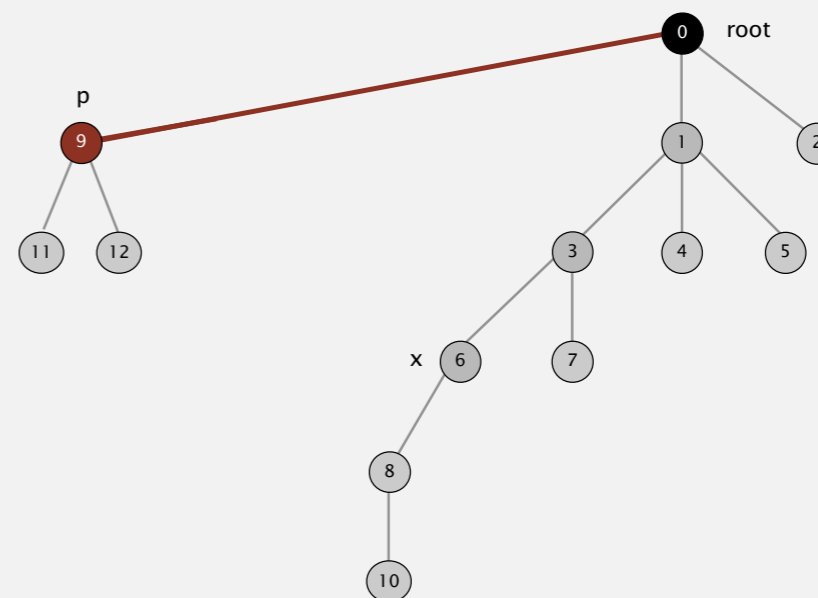
**Quick union with path compression.** Just after computing the root of  $p$ , set the  $\text{id}[]$  of each examined node to point to that root.



35

## Improvement 2: path compression

**Quick union with path compression.** Just after computing the root of  $p$ , set the  $\text{id}[]$  of each examined node to point to that root.

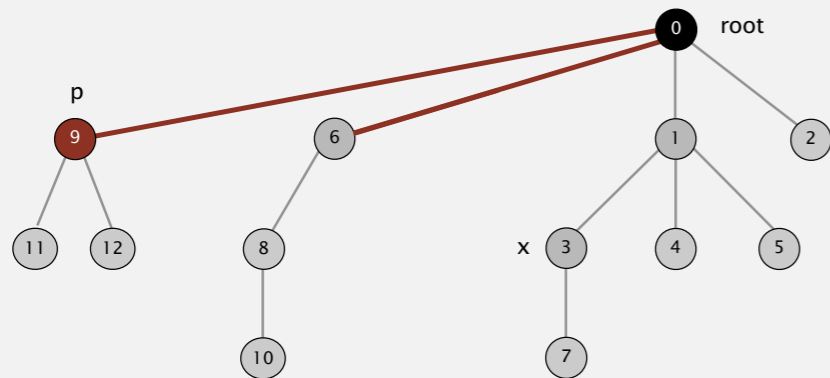


36



## Improvement 2: path compression

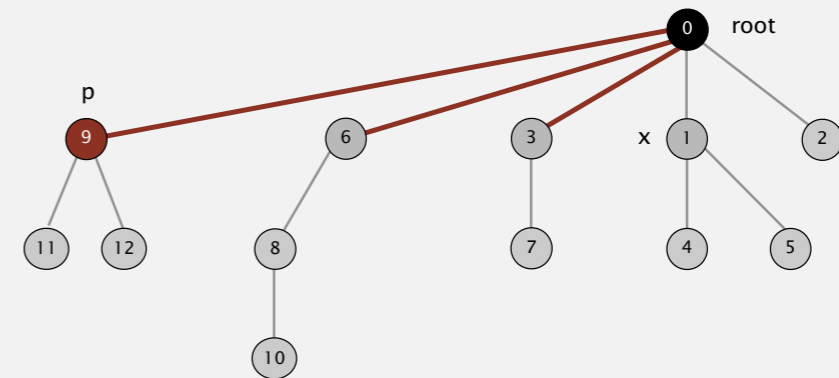
**Quick union with path compression.** Just after computing the root of  $p$ , set the  $id[]$  of each examined node to point to that root.



37

## Improvement 2: path compression

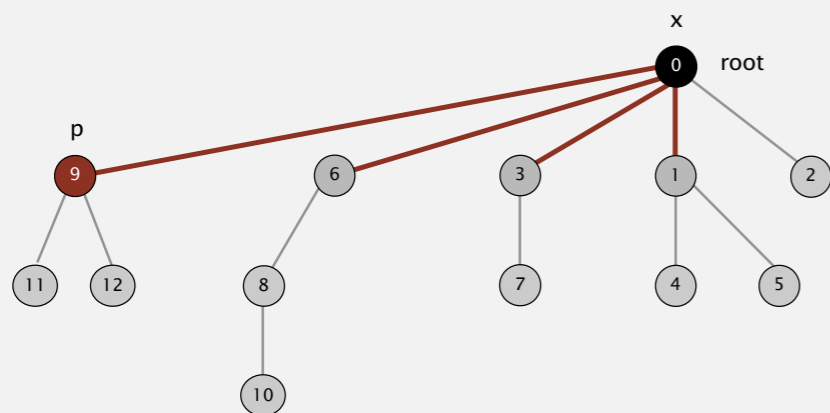
**Quick union with path compression.** Just after computing the root of  $p$ , set the  $id[]$  of each examined node to point to that root.



38

## Improvement 2: path compression

**Quick union with path compression.** Just after computing the root of  $p$ , set the  $id[]$  of each examined node to point to that root.



**Bottom line.** Now,  $find()$  has the side effect of compressing the tree.

39

## Path compression: Java implementation

**Two-pass implementation:** add second loop to  $find()$  to set the  $id[]$  of each examined node to the root.

**Simpler one-pass variant (path halving):** Make every other node in path point to its grandparent.

```
public int find(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]]; ← only one extra line of code!
        i = id[i];
    }
    return i;
}
```

**In practice.** No reason not to! Keeps tree almost completely flat.

40

## Weighted quick-union with path compression: amortized analysis

**Proposition.** [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of  $M$  union-find ops on  $N$  objects makes  $\leq c(N + M \lg^* N)$  array accesses.

- Analysis can be improved to  $N + M \alpha(M, N)$ .
- Simple algorithm with fascinating mathematics.

$N$	$\lg^* N$
1	0
2	1
4	2
16	3
65536	4
$2^{65536}$	5

iterated lg function

Linear-time algorithm for  $M$  union-find ops on  $N$  objects?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

**Amazing fact.** [Fredman-Saks] No linear-time algorithm exists.

↑  
in "cell-probe" model of computation

41

## 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  $N$  objects

**Ex.** [ $10^9$  unions and finds with  $10^9$  objects]

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

42

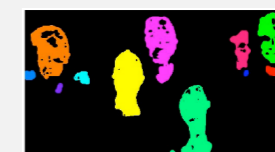
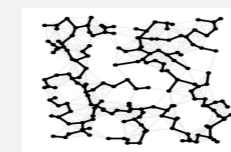
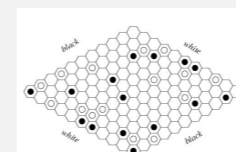
## 1.5 UNION-FIND

- ▶ *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.

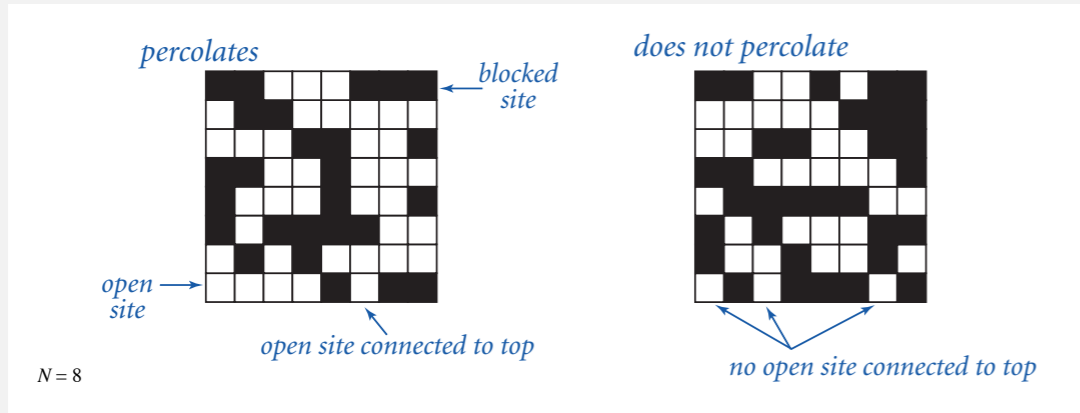


44

## 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.



45

## Percolation

An abstract model for many physical systems:

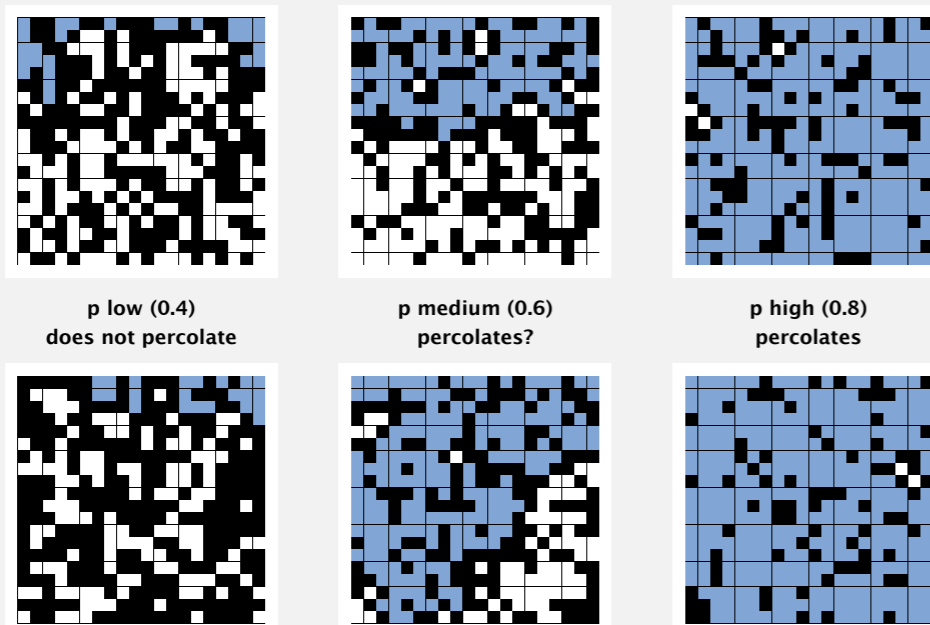
- $N$ -by- $N$  grid of sites.
- Each site is open with probability  $p$  (and blocked with probability  $1 - p$ ).
- System **percolates** iff top and bottom are connected by open sites.

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

46

## Likelihood of percolation

Depends on grid size  $N$  and site vacancy probability  $p$ .



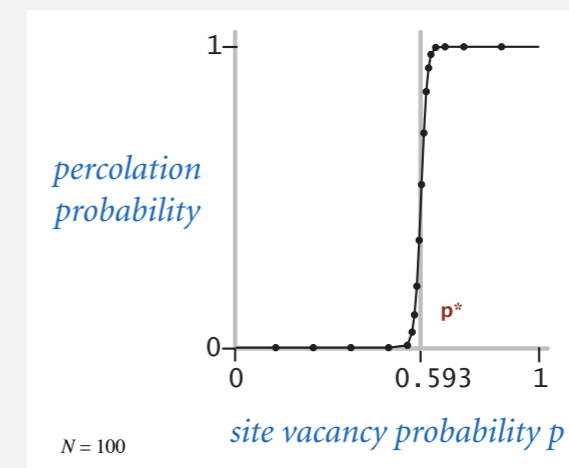
47

## 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.

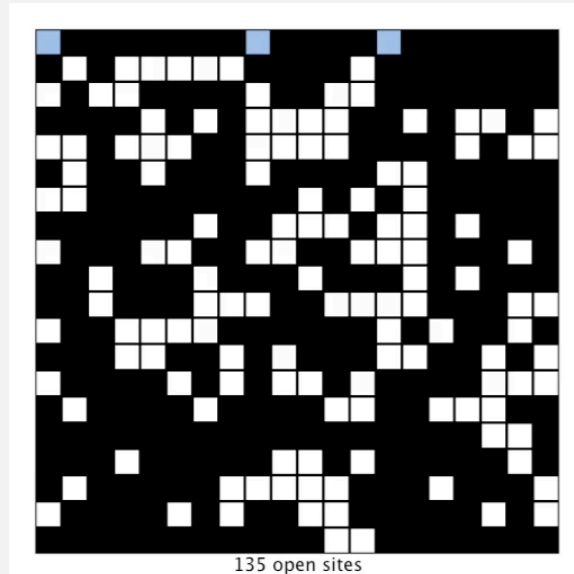
Q. What is the value of  $p^*$  ?



48

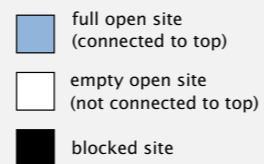
## Monte Carlo simulation

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



$N = 20$

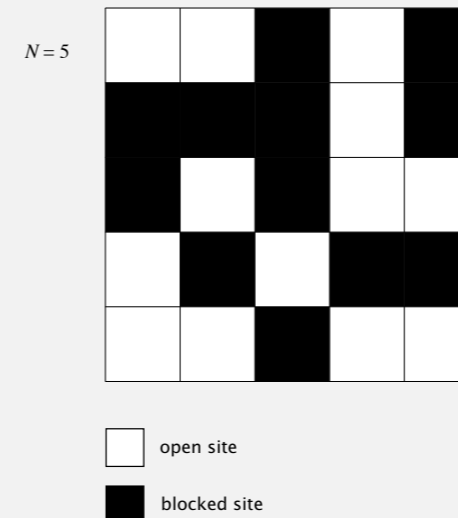
135 open sites



49

## Dynamic connectivity solution to estimate percolation threshold

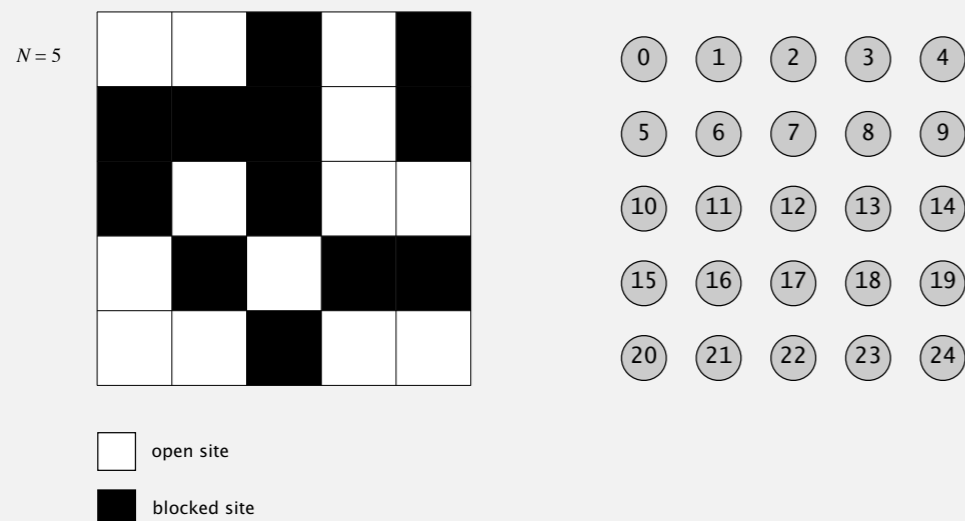
- Q. How to check whether an  $N$ -by- $N$  system percolates?
- A. Model as a **dynamic connectivity** problem and use **union-find**.



50

## Dynamic connectivity solution to estimate percolation threshold

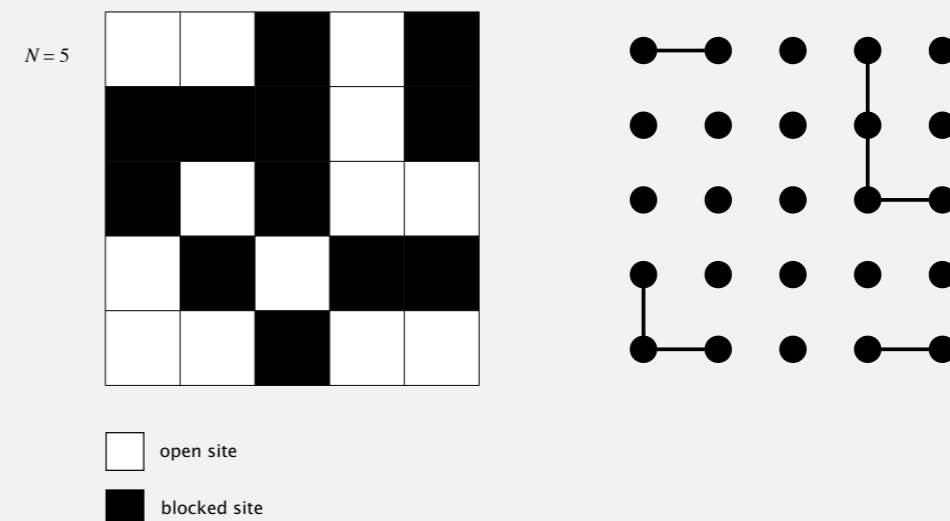
- Q. How to check whether an  $N$ -by- $N$  system percolates?
- Create an object for each site and name them 0 to  $N^2 - 1$ .



51

## Dynamic connectivity solution to estimate percolation threshold

- Q. How to check whether an  $N$ -by- $N$  system percolates?
- Create an object for each site and name them 0 to  $N^2 - 1$ .
  - Sites are in same component iff connected by open sites.



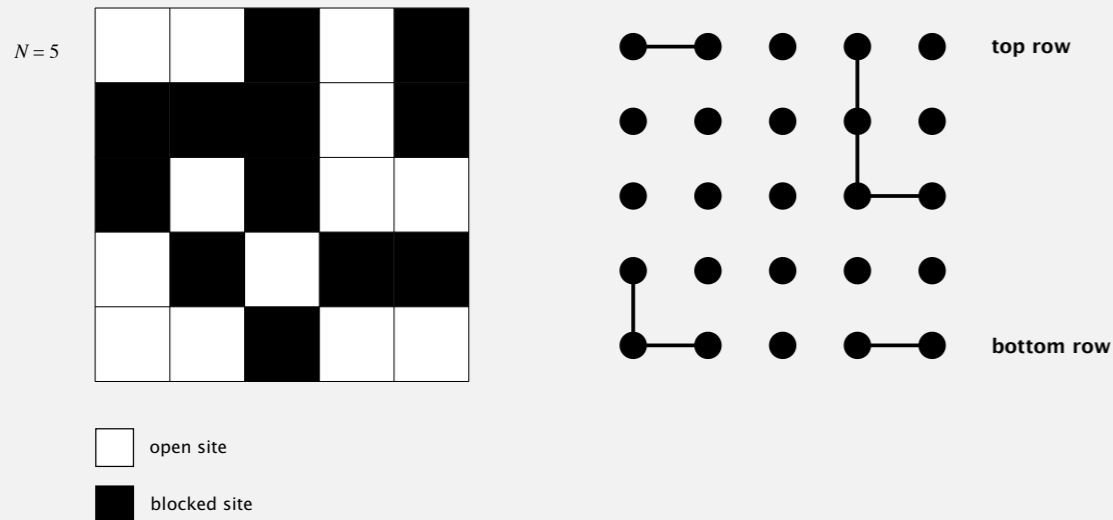
52

## Dynamic connectivity solution to estimate percolation threshold

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

- Create an object 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()`



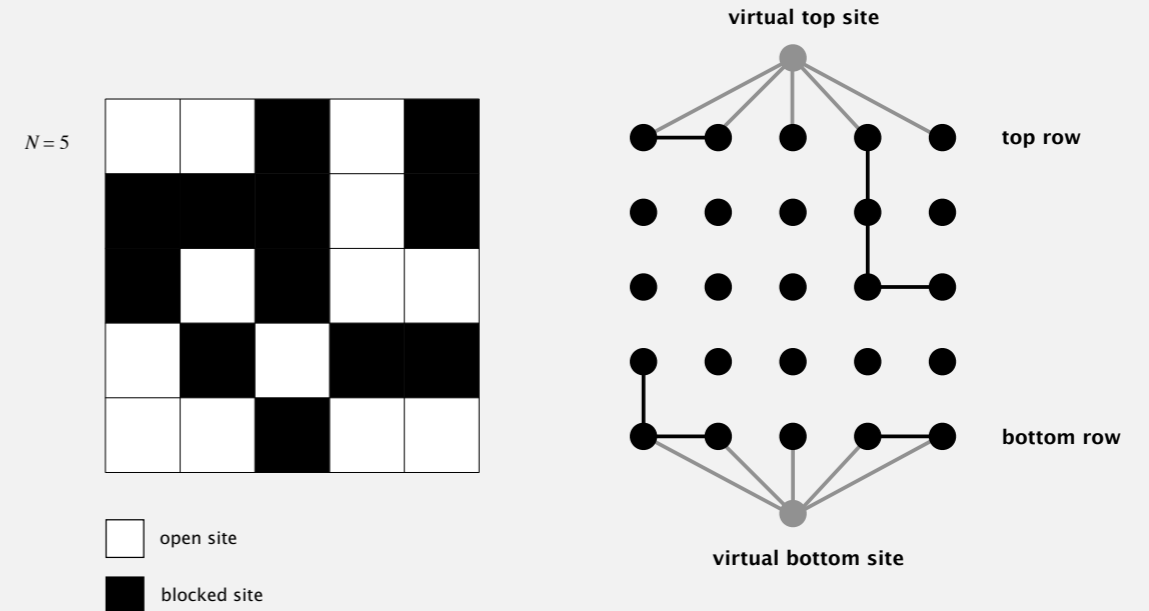
53

## 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.

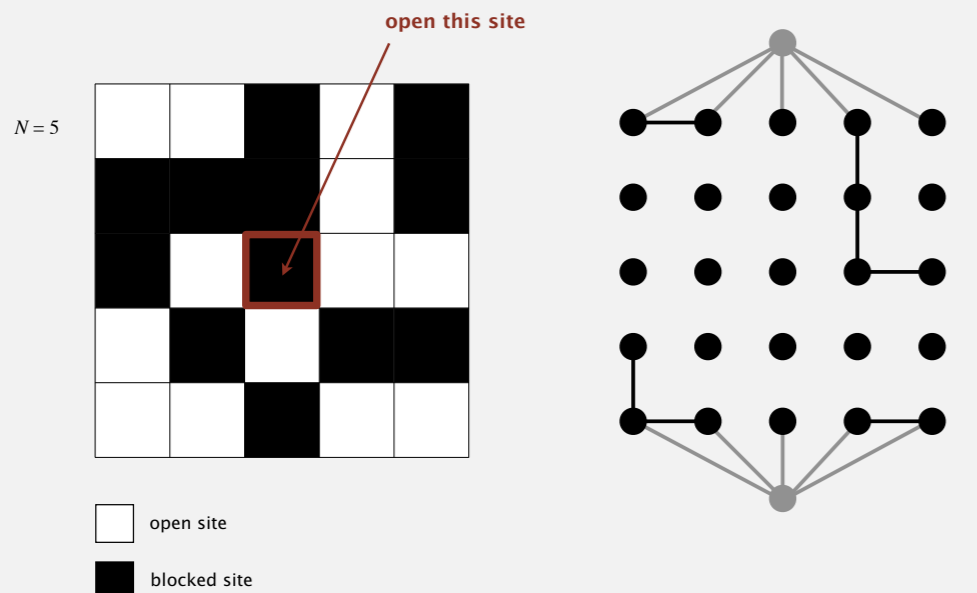
more efficient algorithm: only 1 call to `connected()`



54

## Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?



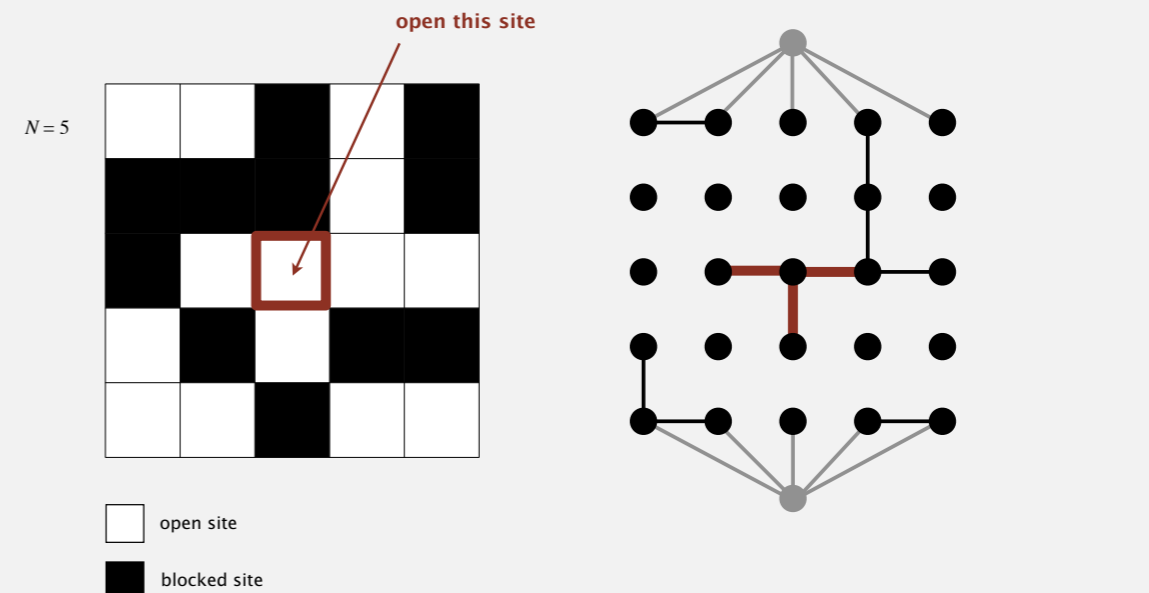
55

## 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()`



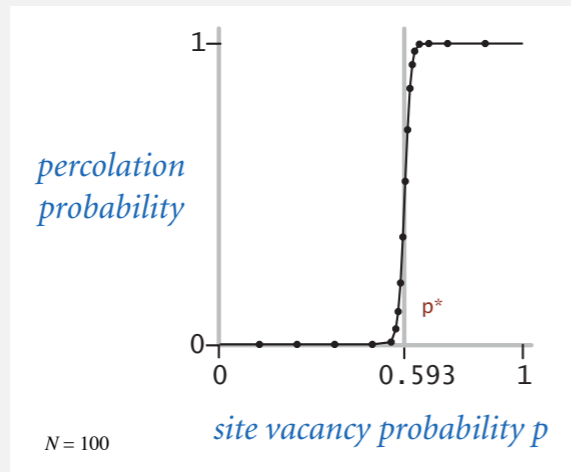
56

## 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.

57

## 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.

58