# COS 226, Fall 2015

# Algorithms and Data Structures

Szymon Rusinkiewicz

Princeton University

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

# COS 226 course overview

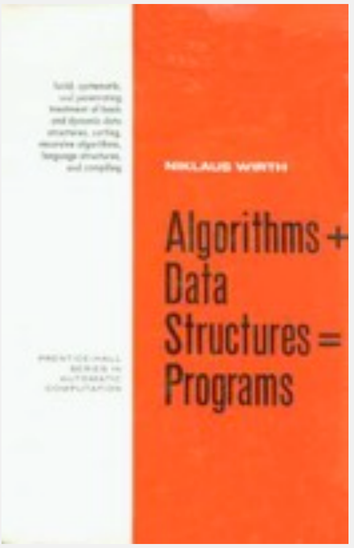
## What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.

- Algorithm: sequence of instructions for solving a problem.
- Data structure: layout + rules for organizing information.

- Application Programming Interface (API): software component with well-defined interfaces, encapsulating algorithms + data structures.

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

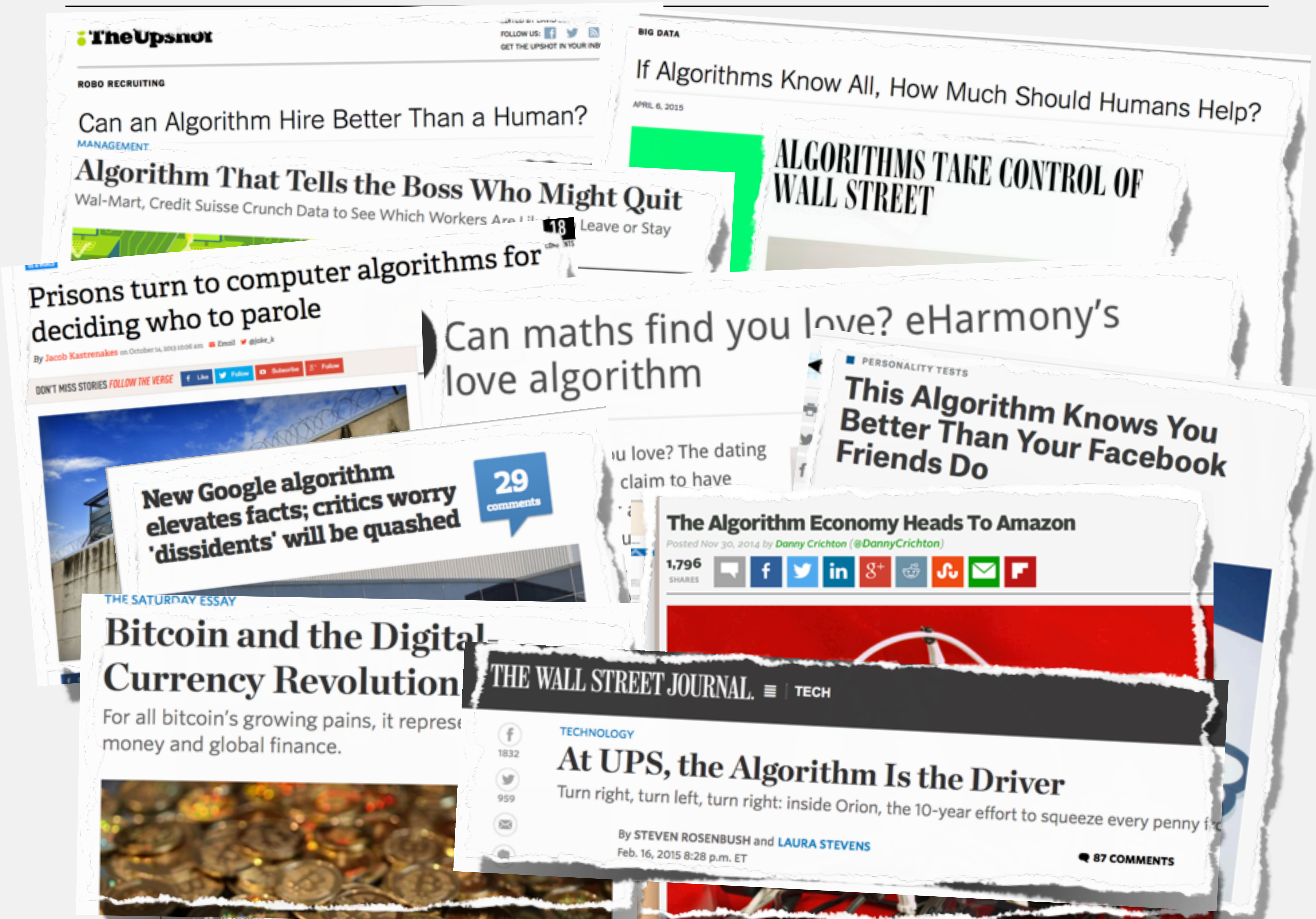


# COS 226 course overview

## What is COS 226?

| 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?

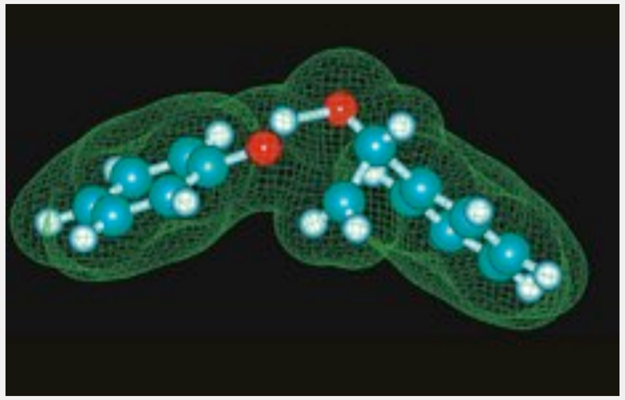


# Why study algorithms?

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

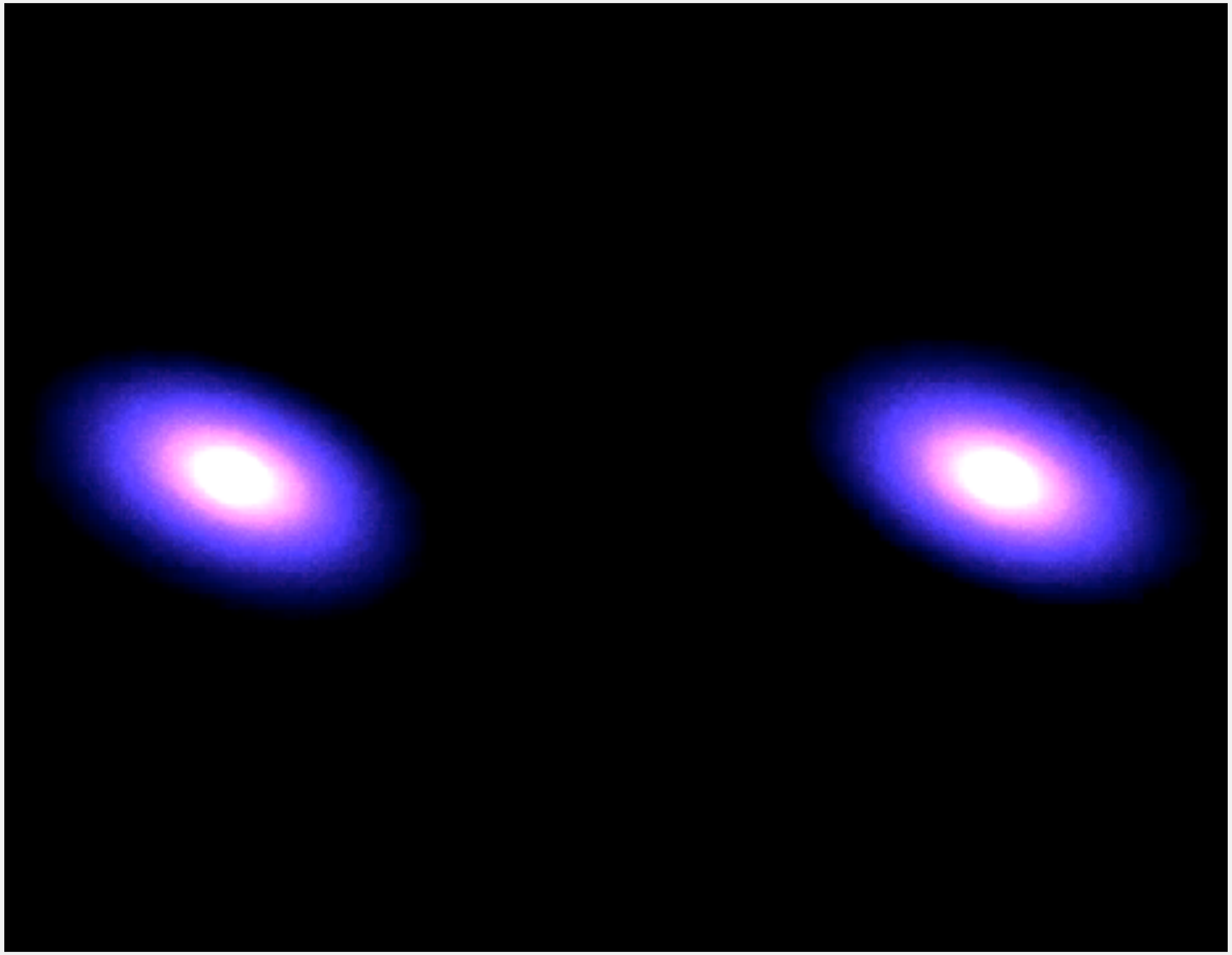
> *" Computer models mirroring real life have become crucial for most advances made in chemistry today…. Today the computer is just as important a tool for chemists as the test tube. "*
>
> — *Royal Swedish Academy of Sciences (Nobel Prize in Chemistry 2013)*

**Martin Karplus, Michael Levitt, and Arieh Warshel**



# Why study algorithms?

To solve problems that could not otherwise be addressed.

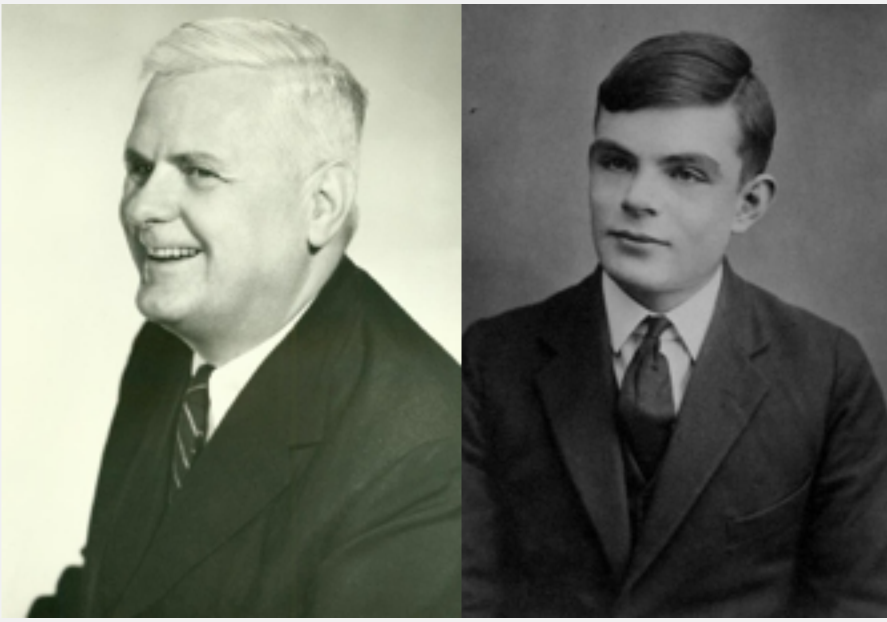


http://www.youtube.com/watch?v=ua7YlN4eL_w

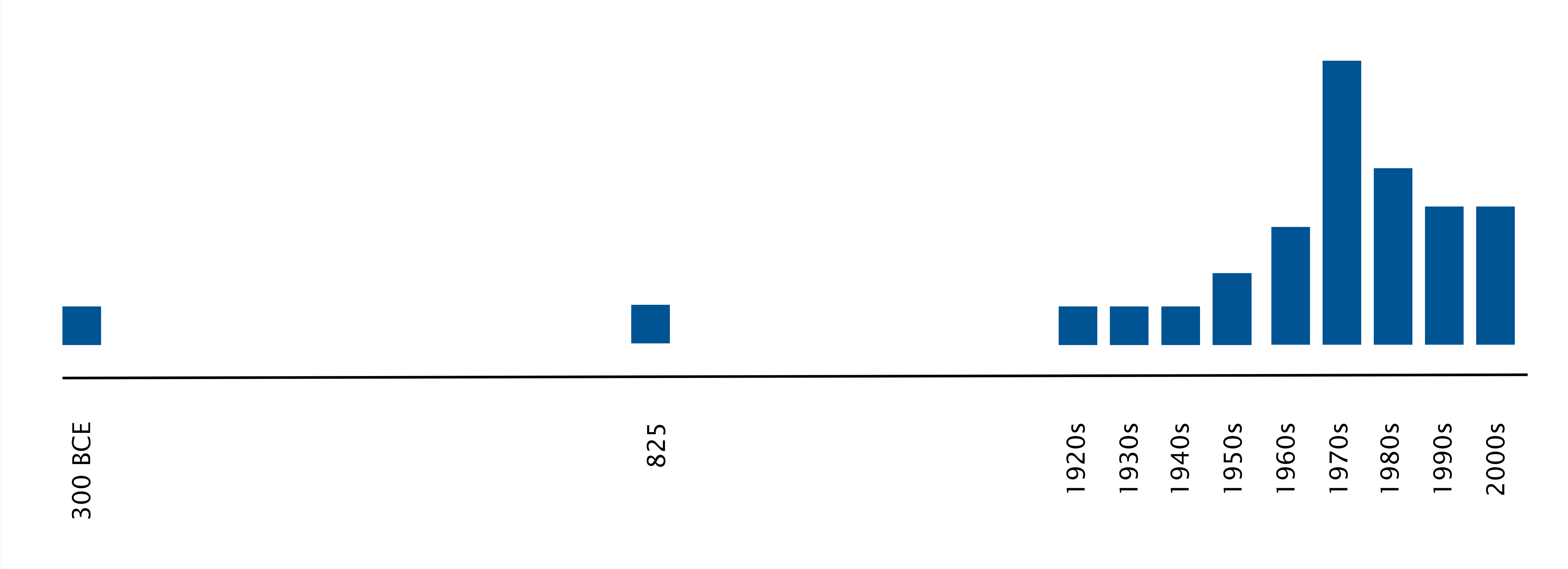
# Why study algorithms?

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!



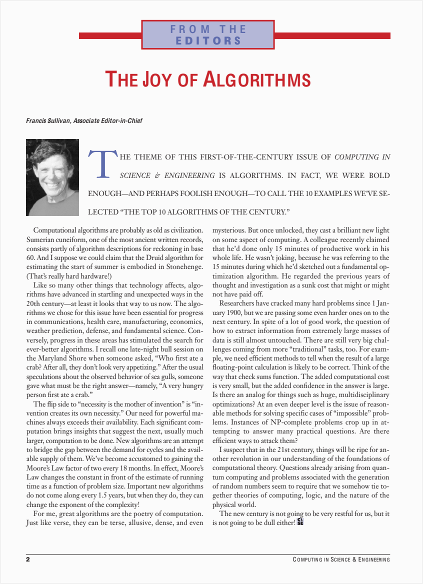
Alonzo Church    Alan Turing



# Why study algorithms and data structures?

## 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

Francis Sullivan, Associate Editor-in-Chief

**DEAR MYSTERY ALGORITHM THAT HOGGED GLOBAL FINANCIAL TRADING LAST WEEK: WHAT DO YOU WANT?**

ON FRIDAY, A SINGLE MYSTERIOUS PROGRAM WAS RESPONSIBLE FOR 4 PERCENT OF ALL STOCK QUOTE TRAFFIC AND SUCKED UP 10 PERCENT OF THE NASDAQ'S TRADING BANDWIDTH. THEN IT DISAPPEARED.

By Clay Dillow   Posted October 10, 2012

0 Shares

# Why study algorithms and data structures?

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)*

# Why study algorithms and data structures?

For fun and profit.

# Lectures

| What | When | Where | Who | Office Hours |
|---|---|---|---|---|
| **L01** | **TTh 11–12:20** | **Friend 101** | **Szymon Rusinkiewicz** | see web |

Traditional lectures. Introduce new material.

Electronic devices. Permitted *only* to enhance lecture (e.g., viewing lecture slides and taking notes).

# Lectures

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

Flipped lectures. Learn at your own pace.

- Video lectures and online learning tools.
- Tuesdays: "film days" with instructor answering questions online in real time.
- Thursdays: "flipped" class sessions to discuss ideas and do collaborative problem solving.
- Same exercises, programming assignments, exams.
- Apply via web by 11:00 PM today, results tomorrow, 2-week shopping.

# Precepts

Discussion, problem-solving, background for assignments.

| What | When | Where | Who | Office Hours |
|---|---|---|---|---|
| **P01** | **F 9–9:50** | **Friend 108** | **Andy Guna †** | see web |
| **P02** | **F 10–10:50** | **Friend 108** | **Andy Guna †** | see web |
| **P02A** | **F 10–10:50** | **Friend 109** | **Elena Sizikova** | see web |
| **P03** | **F 11–11:50** | **Friend 108** | **Maia Ginsburg †** | see web |
| **P03A** | **F 11–11:50** | **Friend 109** | **Nora Coler** | see web |
| **P04** | **F 12:30–1:20** | **Friend 108** | **Maia Ginsburg †** | see web |
| **P04A** | **F 12:30–1:20** | **Friend 109** | **Miles Carlsten** | see web |
| **P05** | **F 1:30–2:20** | **Friend 112** | **Tom Wu** | see web |

† co-lead preceptors

# Coursework and grading

Programming assignments. 45%

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

Exercises. 10%

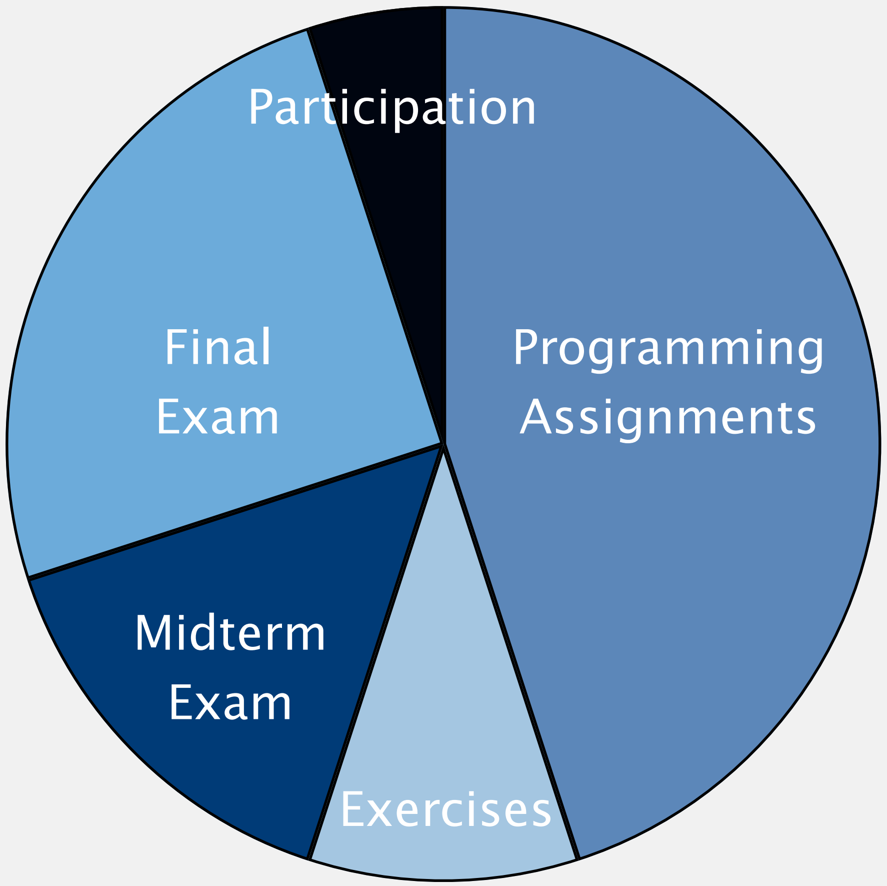
- Due at 11pm on Sundays via Blackboard.
- Collaboration/lateness policies: see web.

Exams. 15% + 25%

- Midterm (in class on Tuesday, Oct 27).
- Final (to be scheduled by the registrar).

Participation. 5%

- Attend and participate in precept/lecture.
- Answer questions on Piazza.



## Required device for lecture.

save serial number to maintain resale value

- Any hardware version of i▸clicker.

  (sorry, insufficient WiFi in this room to support i▸clicker GO)
- Available at Labyrinth Books ($25).
- Use default frequency AA.
- You must register your i▸clicker in Blackboard.

Which model of i▸clicker are you using?

**A.** i▸clicker.

**B.** i▸clicker+.

**C.** i▸clicker 2.

**D.** *I don't know.*

**E.** *I don't have one yet.* *(Ummm.. how are you answering this?)*

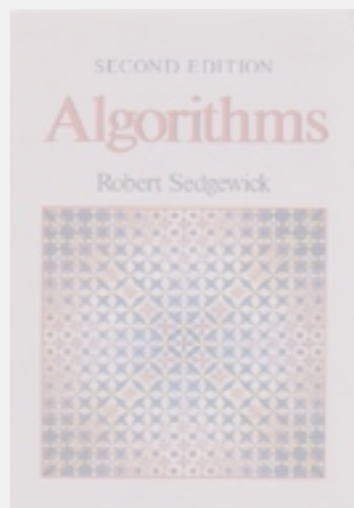


# Resources (textbook)

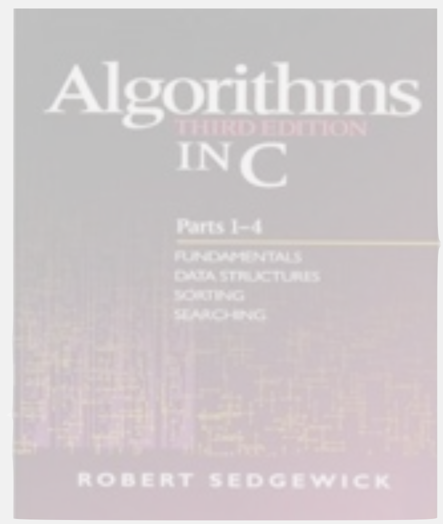
Required reading. Algorithms 4th edition by R. Sedgewick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.




1st edition (1982)




2nd edition (1988)




3rd edition (1997)


**4th edition (2011)**

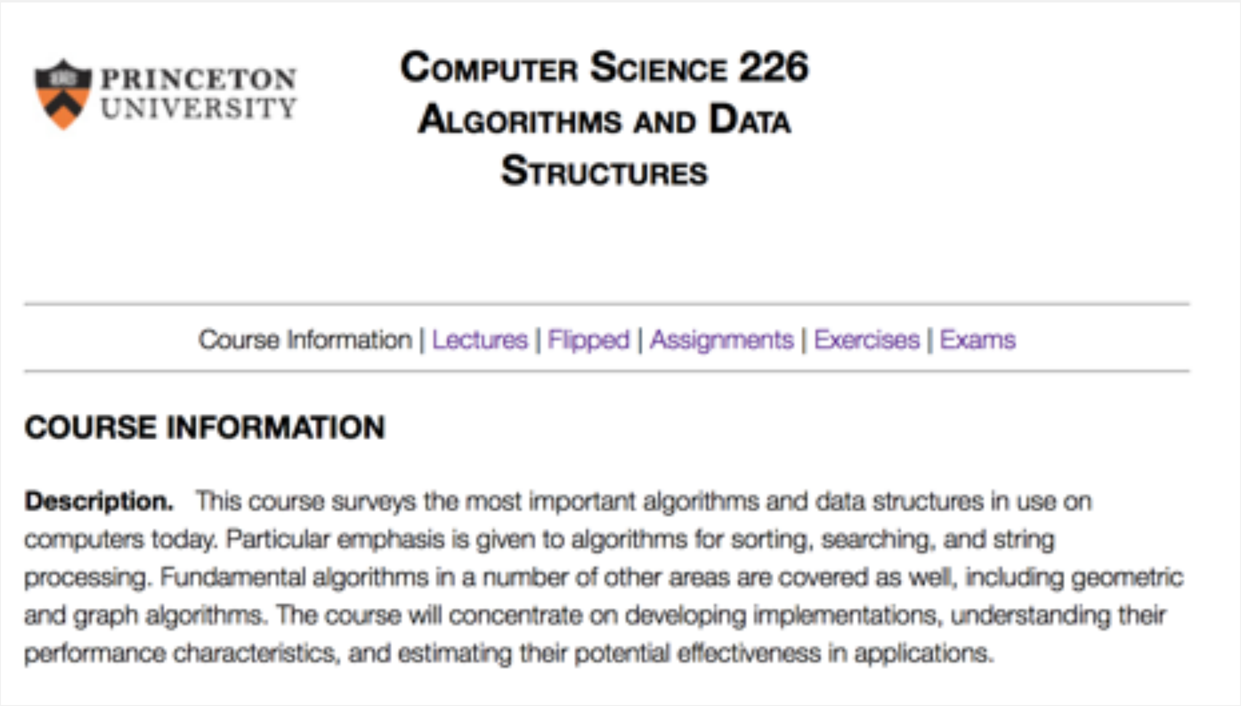
Available in hardcover and Kindle.

- Online: Amazon ($60 hardcover, $55 Kindle, $50 rent), ...
- Brick-and-mortar: Labyrinth Books (122 Nassau St.).
- On reserve: Engineering library.

# Resources (web)

## Course content.

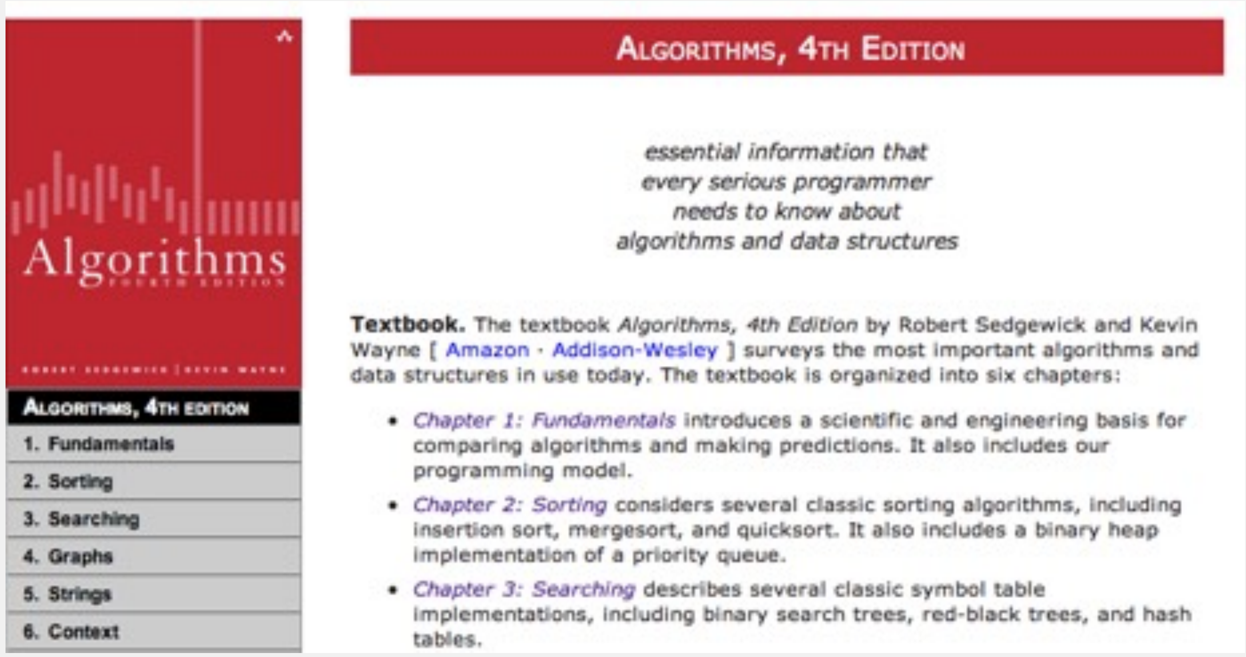
- Course info.
- Lecture slides.
- Flipped lectures.
- Programming assignments.
- Exercises.
- Exam archive.




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

## Booksite.

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




**http://algs4.cs.princeton.edu**

# Resources (people)

## Piazza discussion forum.

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

http://piazza.com/princeton/fall2015/cos226

## Office hours.

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


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

## Lab TAs.

- For help with debugging.
- See web for schedule.

http://labta.cs.princeton.edu

# What's ahead?


Today. Attend lecture.

Friday. Attend precept.

```
for (int week = 1; week < reading_period; week++) {
   if (week != fall_break) {
      Sunday: two sets of exercises due.
      Tuesday: traditional/flipped lecture.
      Wednesday: programming assignment due.
      Thursday: traditional/flipped lecture.
      Friday: precept.
   }
}
```

yes, even for week == 1

yes, even for week == 1

be sure to start early!

this means you!

really!

start early!

# Q+A

Not registered? Go to any precept this week.

Registered but not continuing? Drop as soon as possible.

Change precept? Use TigerHub.

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.

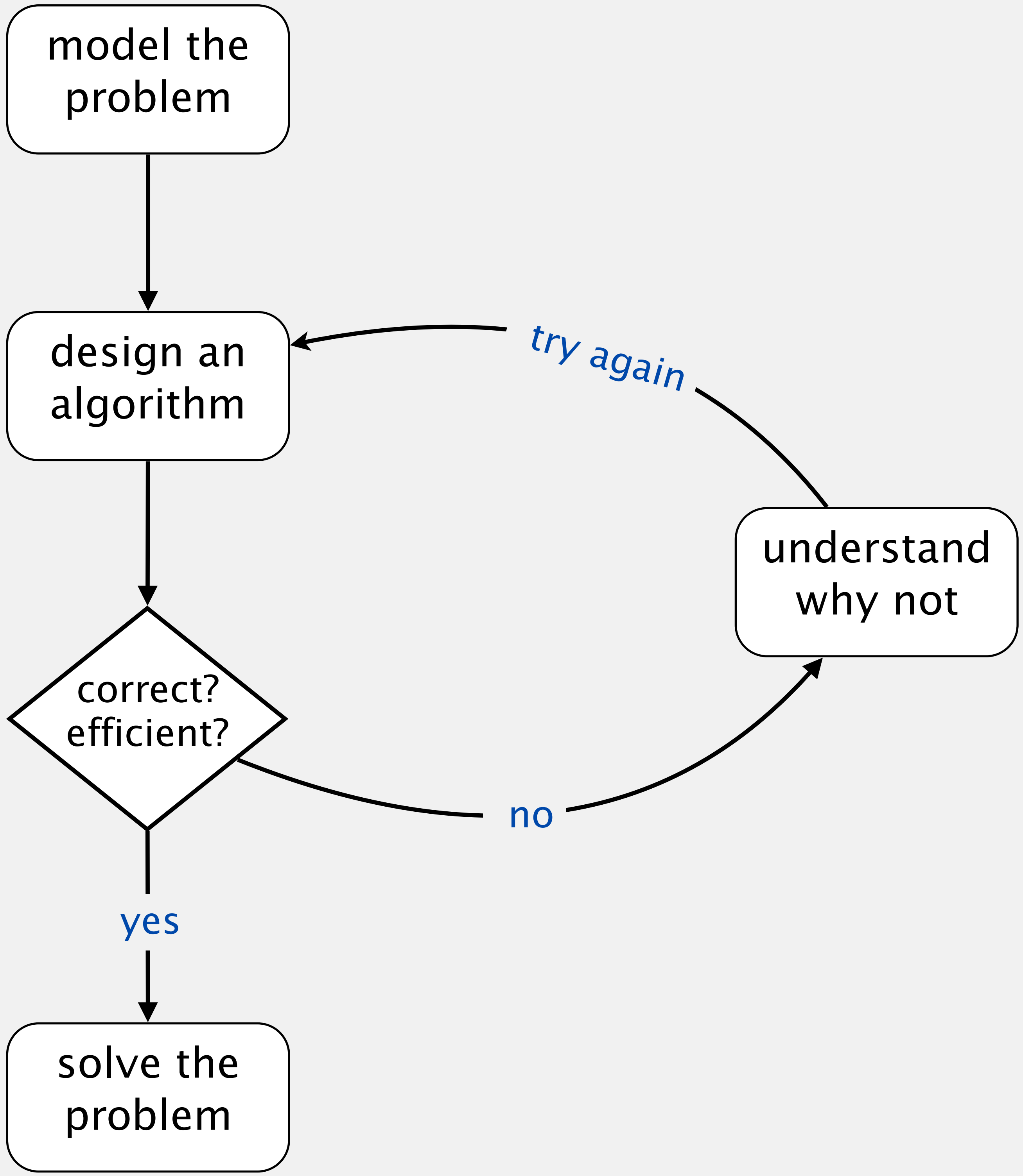

http://algs4.cs.princeton.edu

# 1.5 UNION-FIND

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

# Subtext of today's lecture (and this course)

Steps to developing a usable algorithm to solve a computational problem.




http://algs4.cs.princeton.edu

# 1.5 Union-Find

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

# Dynamic-connectivity problem

Given a set of N elements, support two operations:

- Connect two elements with an edge.
- Query: is there a path connecting two elements?

*connect 4 and 3*

*connect 3 and 8*

*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*

*are 8 and 9 connected?* ✔

*are 5 and 7 connected?* ✕

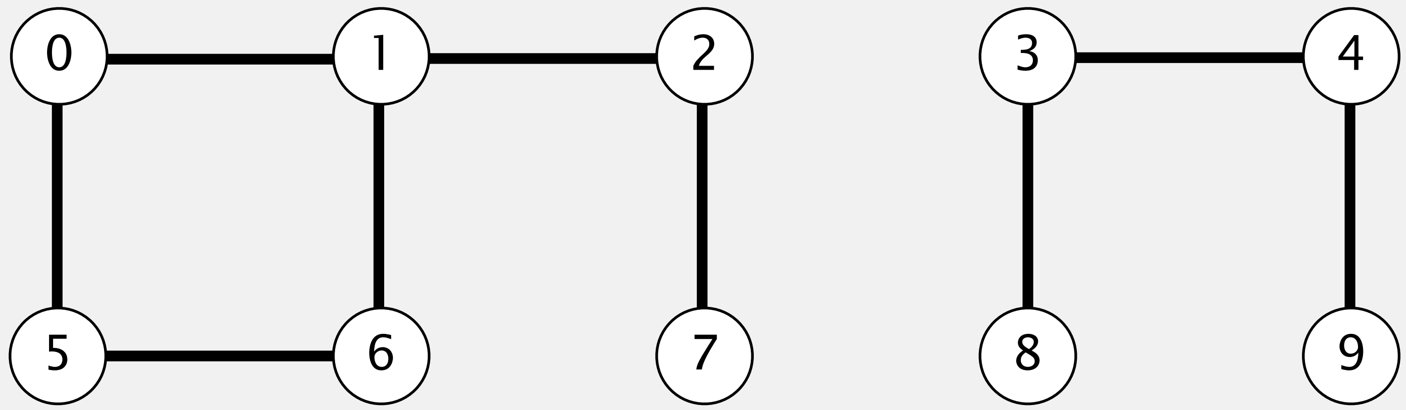
*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

*connect 1 and 0*

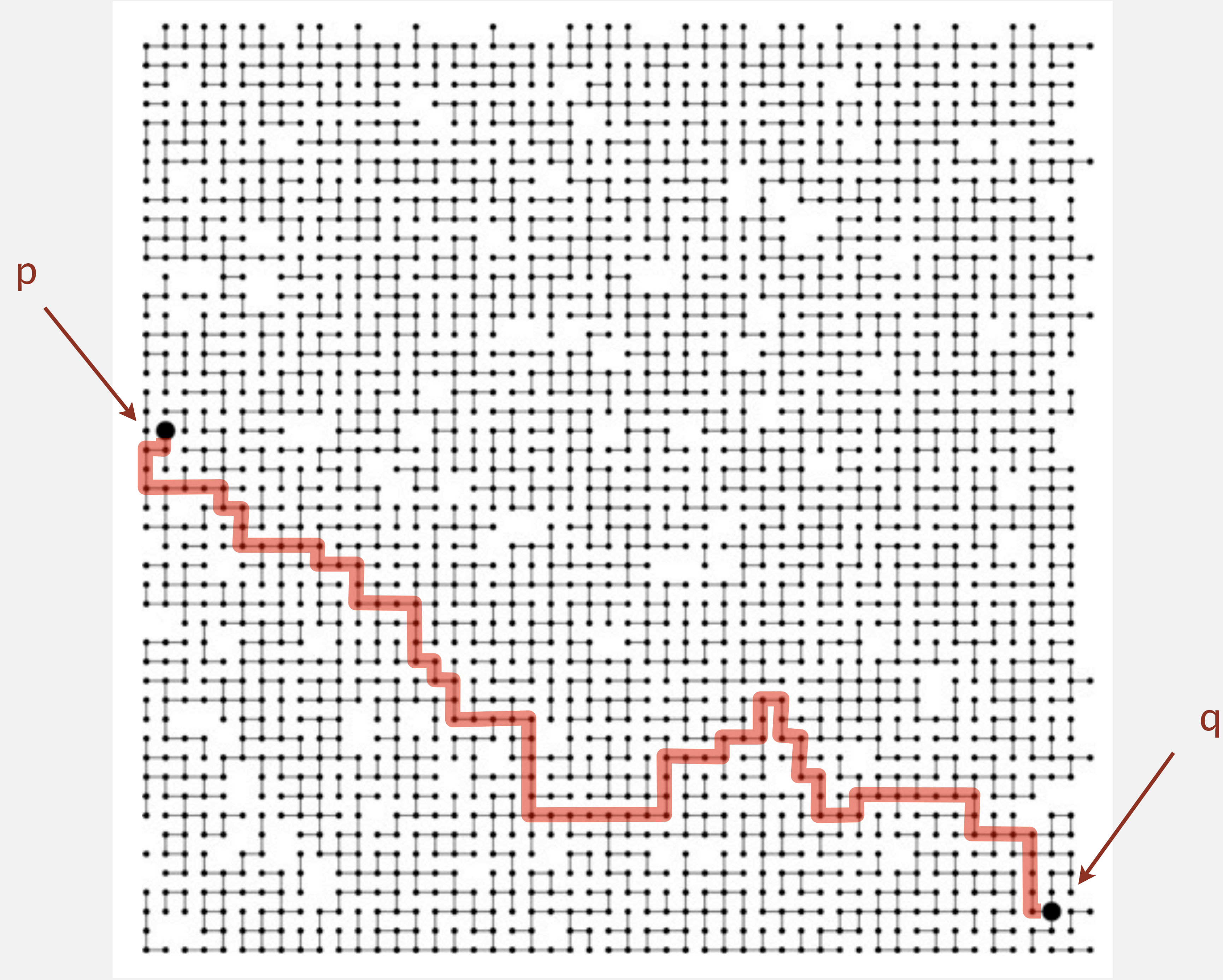
*are 5 and 7 connected?* ✔



# A larger connectivity example

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

A. Yes.




(finding the path explicitly is a harder problem:
stay tuned for graph algorithms in a few weeks)

# 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.

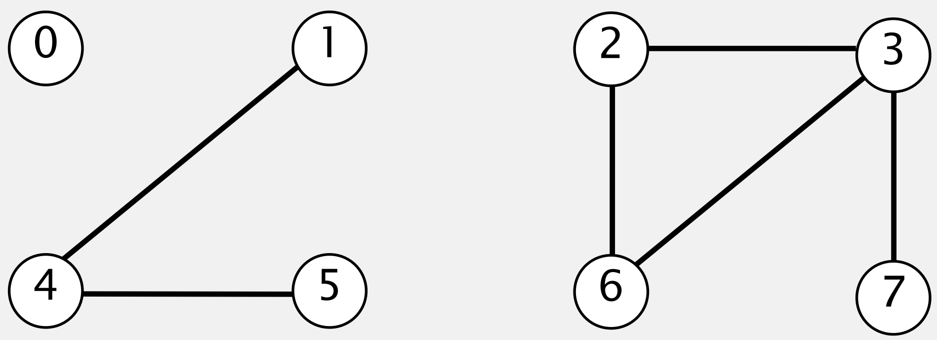
can use symbol table to translate from site names to integers (stay tuned for Chapter 3)

# Modeling the connections

We model "is connected to" as 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 mutually-connected elements.




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

**3 disjoint sets**

**(connected components)**

# Two core operations on disjoint sets

Union. Replace set $p$ and $q$ with their union.

Find. In which set is element $p$ ?

**union(2, 5)**

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

**3 disjoint sets**

**find(5) == find(6)** ✔

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

**2 disjoint sets**

# Modeling the dynamic-connectivity problem using union-find

Q. How to model the dynamic-connectivity problem using union-find?

A. Maintain disjoint sets that correspond to connected components.

- Connect elements $p$ and $q$: **union**.
- Are elements $p$ and $q$ connected? **find**.

**union(2, 5)**

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

**3 disjoint sets**

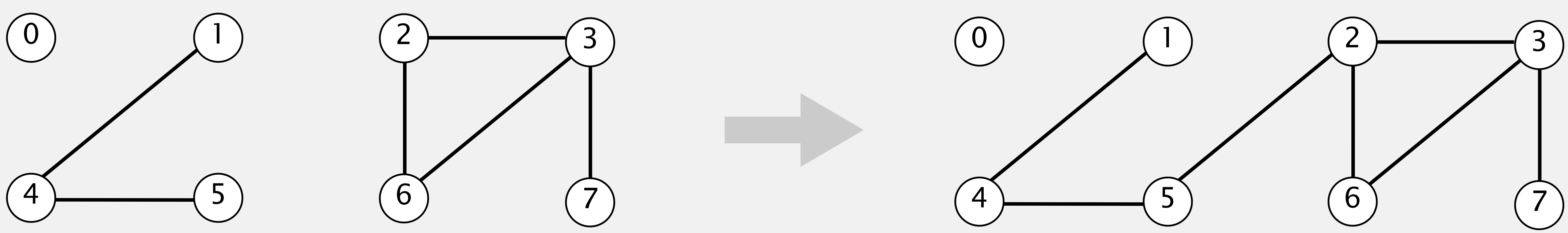
**find(5) == find(6)** ✔

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

**2 disjoint sets**

**connect 2 and 5**

**are 5 and 6 connected?**




**3 connected components**

**2 connected components**

# Union-find data type (API)

Goal. Design a union-find data type.

| public class UF | |
|---|---|
| `UF(int N)` | *initialize union-find data structure with N singleton sets (0 to N – 1)* |
| `void union(int p, int q)` | *merge sets containing elements p and q* |
| `int find(int p)` | *identifier for set containing element p (0 to N – 1)* |

# Union-find data type (API)

Goal. Design an efficient union-find data type.

- 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 sets (0 to N – 1)* |
| `void union(int p, int q)` | *merge sets containing elements p and q* |
| `int find(int p)` | *identifier for set containing element p (0 to N – 1)* |

# 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 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.find(p) != uf.find(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 (don't print these)


http://algs4.cs.princeton.edu

# 1.5 Union-Find

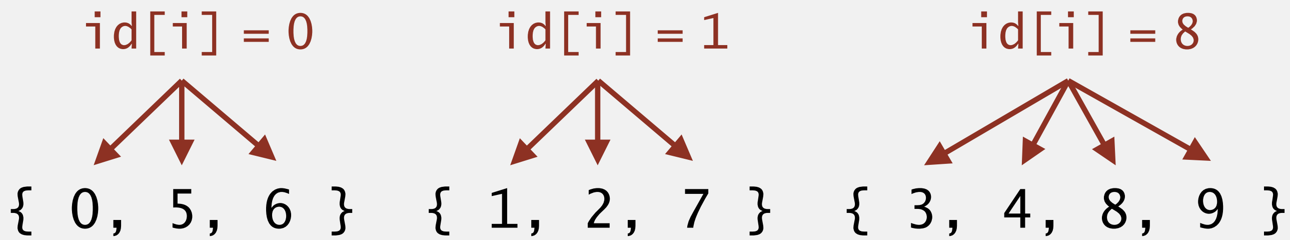
- *dynamic-connectivity problem*
- ***quick find***
- *quick union*
- *improvements*
- *applications*

# Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` identifies the set containing element `p`.

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




**3 disjoint sets**

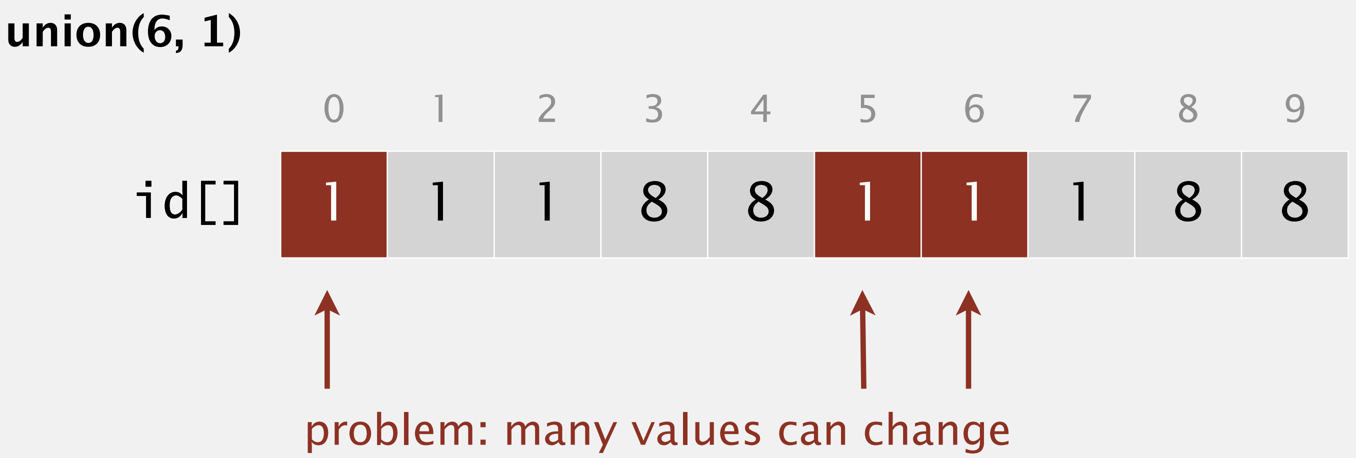
Q. How to implement `find(p)`?

A. Easy, just return `id[p]`.

# Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` identifies the set containing element `p`.

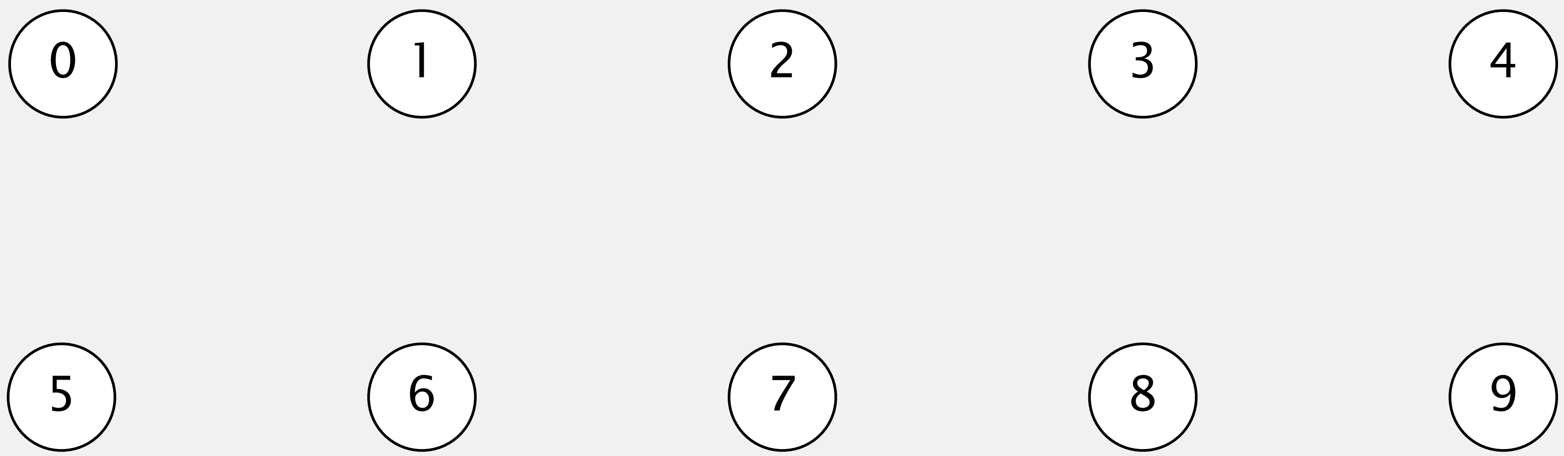



Q. How to implement `union(p, q)`?

~~A? Change `id[p]` to `q`.~~

A. Change all entries whose identifier equals `id[p]` to `id[q]`.

# 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: 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]
(N+2 to 2N+2 array accesses)

# Quick-find is too slow

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

| algorithm | initialize | union | find |
|---|---|---|---|
| **quick-find** | $N$ | $N$ | 1 |

**number of array accesses (ignoring leading constant)**

Union is too expensive. Processing a sequence of $N$ union operations on $N$ elements takes more than $N^2$ array accesses.

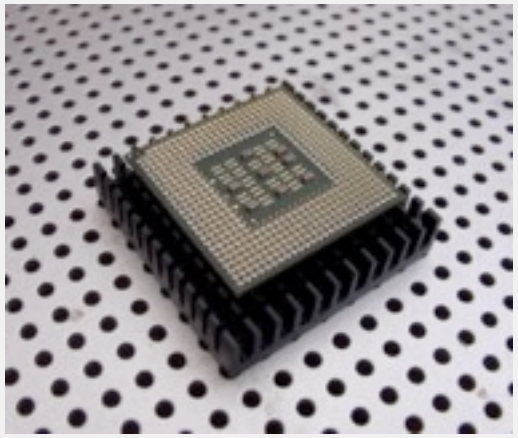
quadratic

# 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!

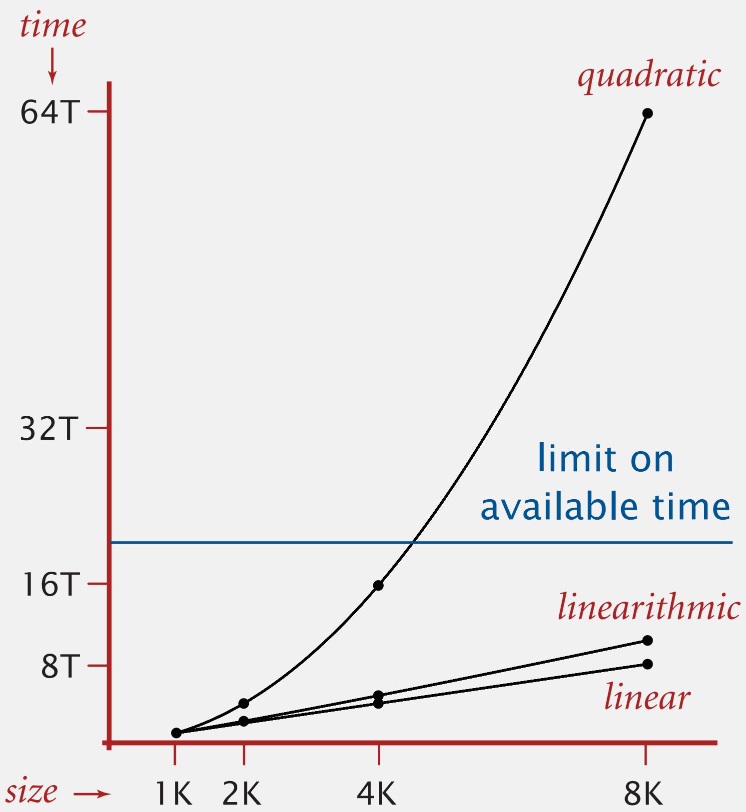


Ex. Huge problem for quick-find.

- $10^9$ union commands on $10^9$ 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 ⇒ 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

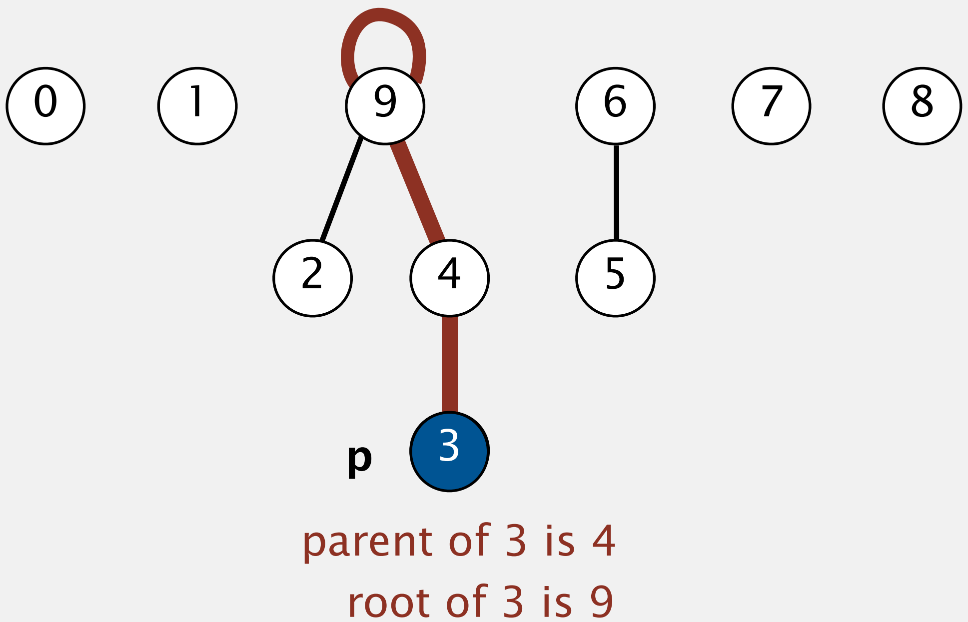
- *dynamic-connectivity problem*
- *quick find*
- *quick union*
- *improvements*
- *applications*

# Quick-union [lazy approach]

## Data structure.

- Integer array `parent[]` of length `N`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in a tree corresponding to a set.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |




`find(i) = 9`

`{ 0 } { 1 } { 2, 3, 4, 9 } { 5, 6 } { 7 } { 8 }`

**3 disjoint sets (3 trees)**

Q. How to implement `find(p)` operation?

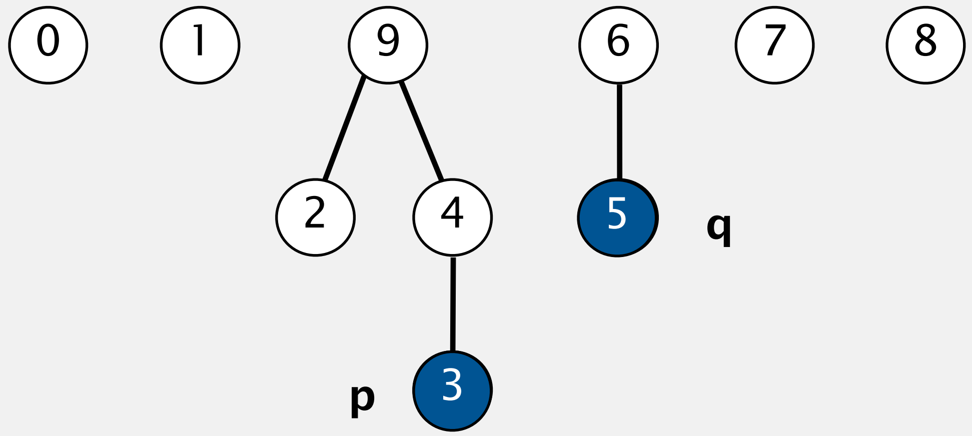
A. Return root of tree containing `p`.

# Quick-union [lazy approach]

Data structure.

- Integer array `parent[]` of length `N`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in a tree corresponding to a set.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **union(3, 5)** | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |




Q. How to implement `union(p, q)`?

A. Set parent of `p`'s root to parent of `q`'s root.

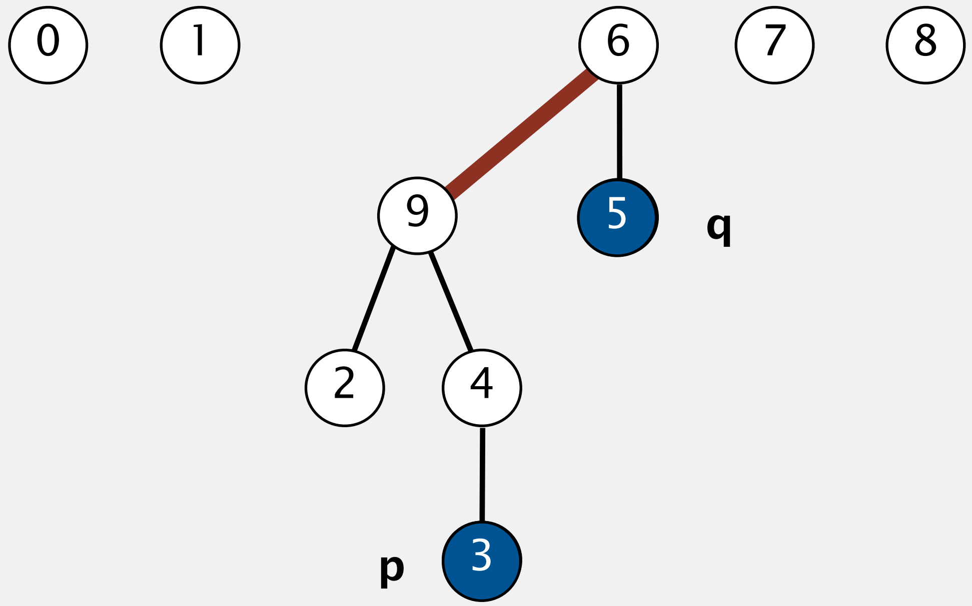
# Quick-union [lazy approach]

## Data structure.

- Integer array `parent[]` of length `N`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in a tree corresponding to a set.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **union(3, 5)** | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 6 |

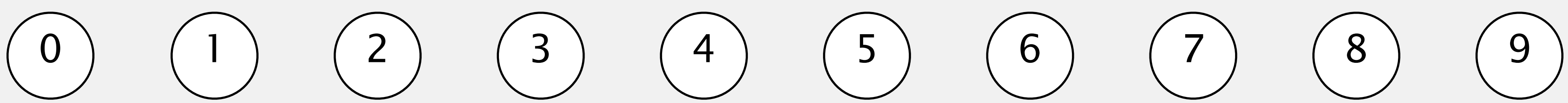
only one value changes




Q. How to implement `union(p, q)`?

A. Set parent of `p`'s root to parent of `q`'s root.

# Quick-union demo




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Quick-union: Java implementation

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

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

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

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

set parent 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

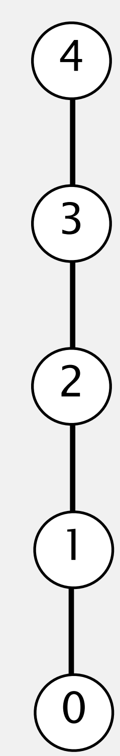
Cost model. Number of array accesses (for read or write).

| algorithm | initialize | union | find |
|---|---|---|---|
| **quick-find** | $N$ | $N$ | 1 |
| **quick-union** | $N$ | $N$ † | $N$ |

← worst case

† includes cost of finding two roots

**worst-case input**




**union(0, 1)**
**union(0, 2)**
**union(0, 3)**
**union(0, 4)**

Quick-find defect.

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

Quick-union defect.

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


http://algs4.cs.princeton.edu

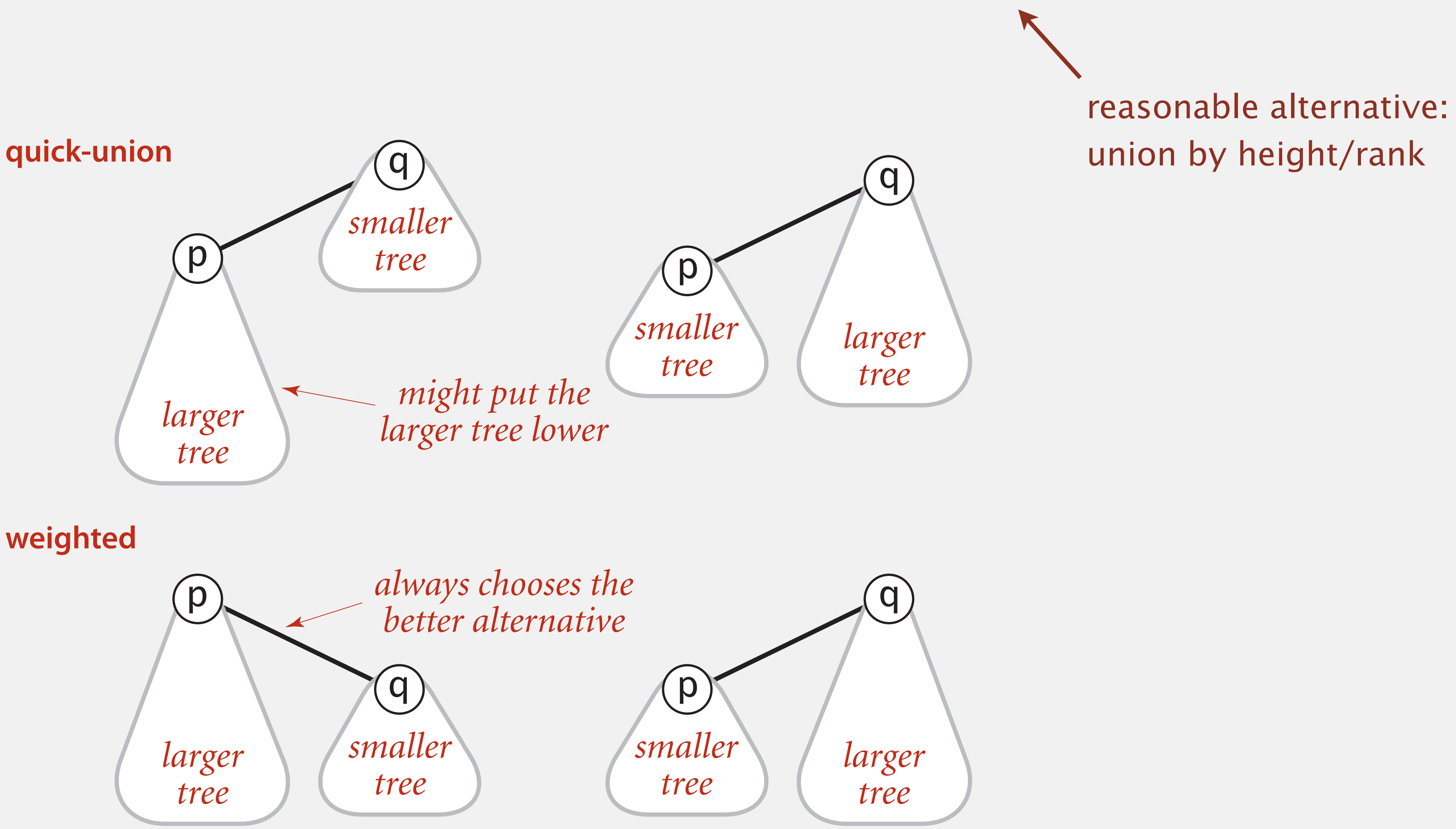
# 1.5 Union-Find

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

# Weighting

Weighted quick-union.

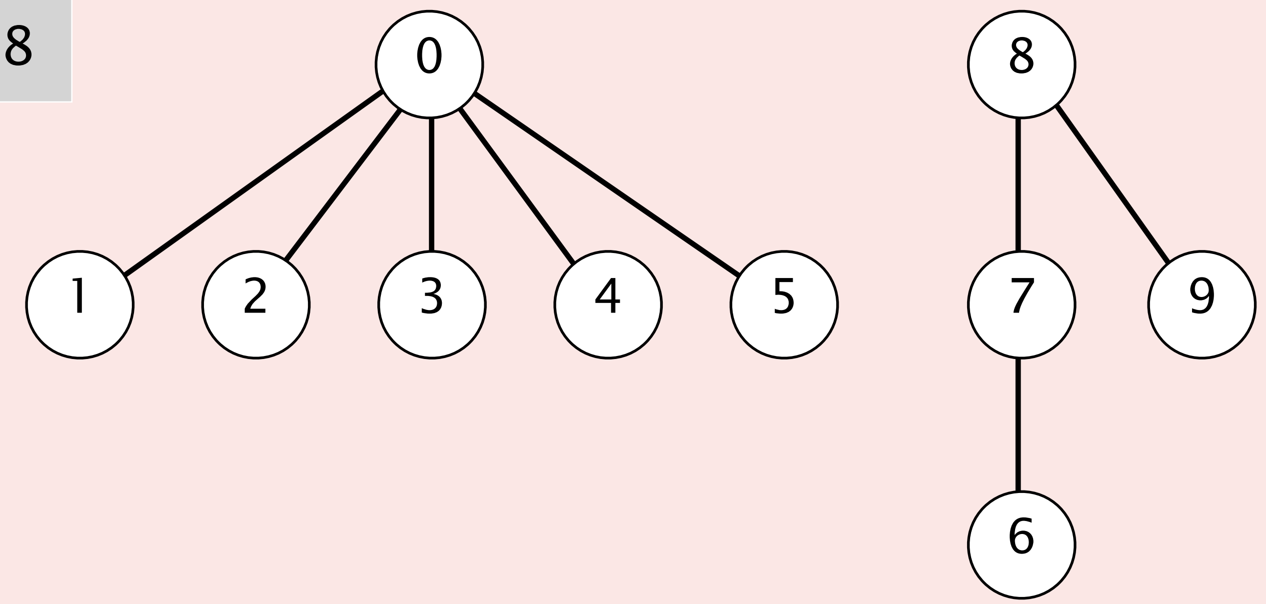
- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of elements).
- Always link root of smaller tree to root of larger tree.



# Weighted quick-union quiz

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

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **parent[]** | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 8 | 8 | 8 |




Which `parent[]` entry changes during `union(2, 6)`?

**A.** `parent[0]`

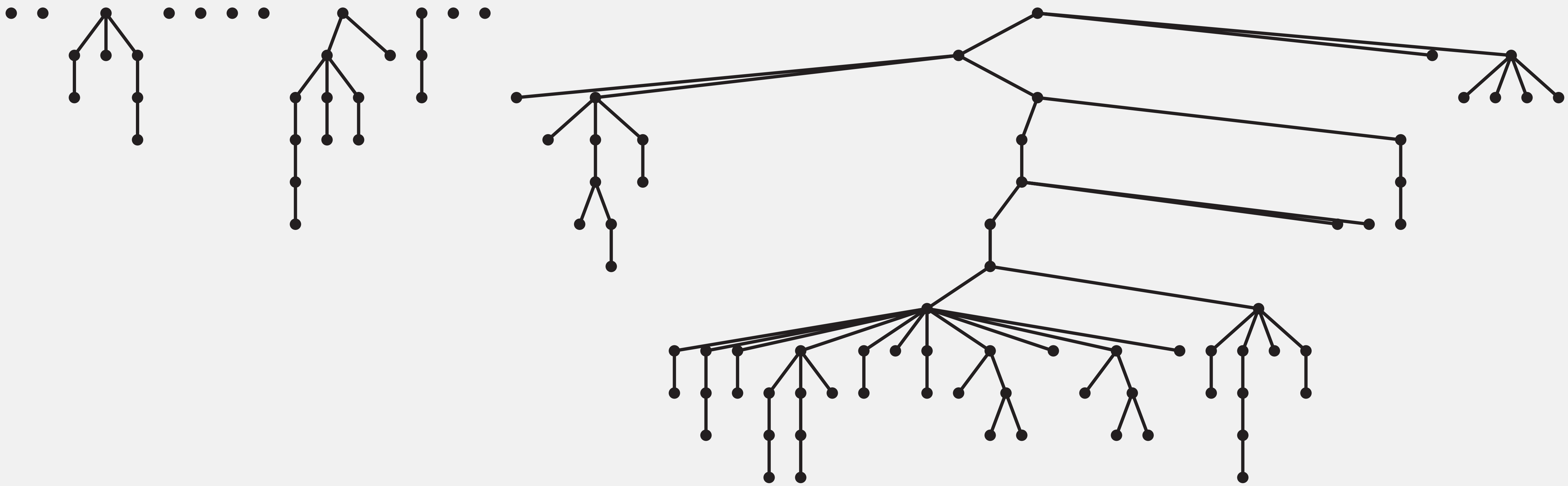
**B.** `parent[2]`

**C.** `parent[6]`

**D.** `parent[8]`

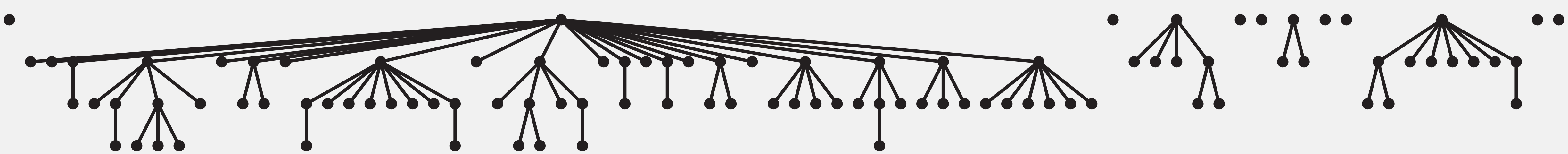
# Quick-union vs. weighted quick-union: larger example

**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 `size[i]` to count number of elements in the tree rooted at `i`, initially `1`.

Find. Identical to quick-union.

Union. Modify quick-union to:

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

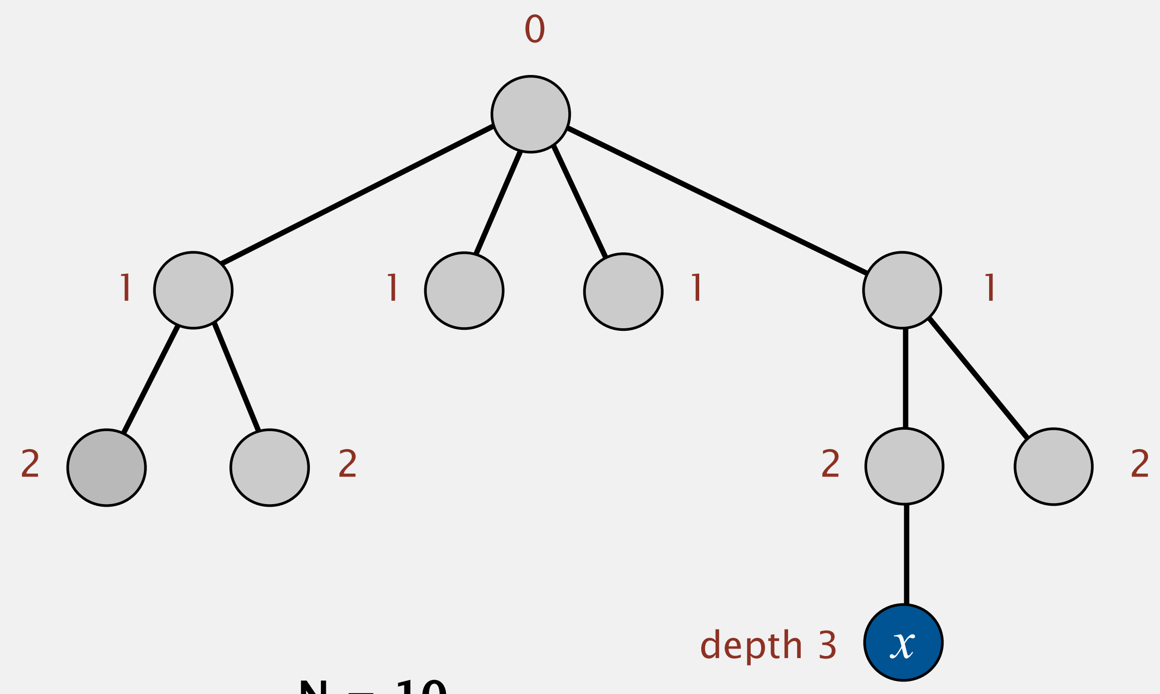
```
int i = find(p);
int j = find(q);
if (i == j) return;
if  (size[i] < size[j]) { parent[i] = j; size[j] += size[i]; }
else                    { parent[j] = i; size[i] += size[j]; }
```

# 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 $\lg N$. ← lg means base-2 logarithm




N = 10
depth(x) = 3 ≤ lg N

# Weighted quick-union analysis

Running time.

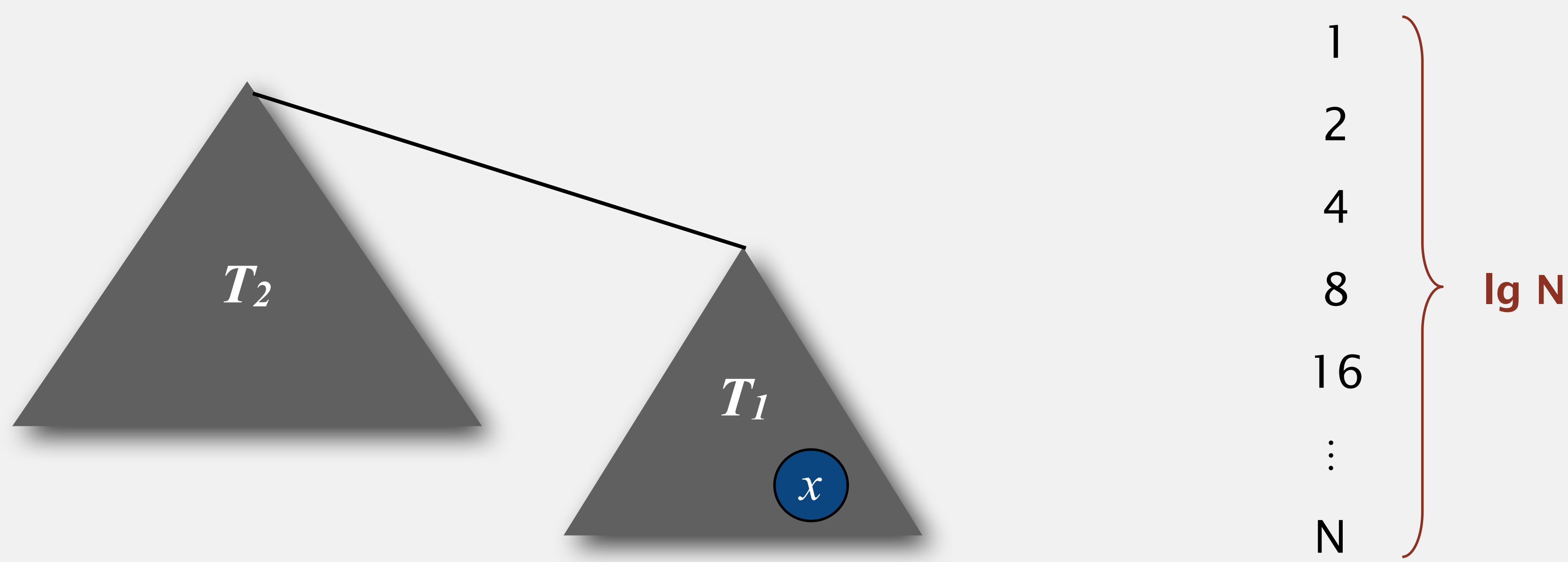
- Find: takes time proportional to depth of $p$.
- Union: takes constant time, given two roots.

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

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

Increases by $1$ when root of tree $T_1$ containing $x$ is linked to root of 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?



# 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 |
|---|---|---|---|
| **quick-find** | $N$ | $N$ | $1$ |
| **quick-union** | $N$ | $N$ † | $N$ |
| **weighted QU** | $N$ | $\log N$ † | $\log N$ |

† includes cost of finding two roots

# Summary

Key point. Weighted quick union 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.

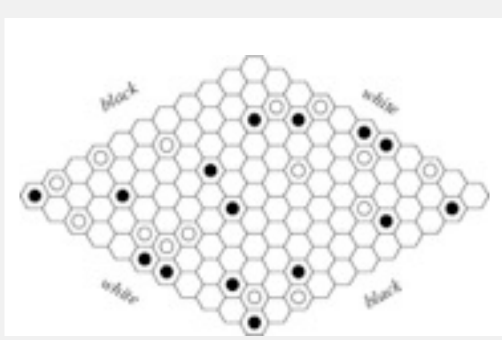

http://algs4.cs.princeton.edu

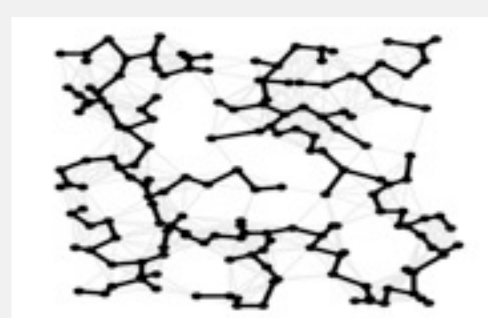
# 1.5 Union-Find

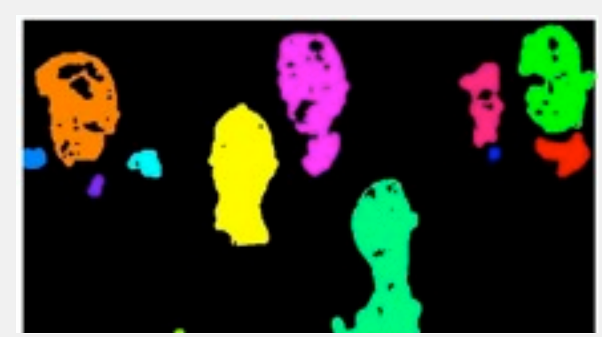
- *dynamic-connectivity problem*
- *quick find*
- *quick union*
- *improvements*
- *applications*

# Union-find applications

- Percolation.
- Games (Go, Hex).
- Least common ancestor.
- ✓ Dynamic-connectivity problem.
- 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.





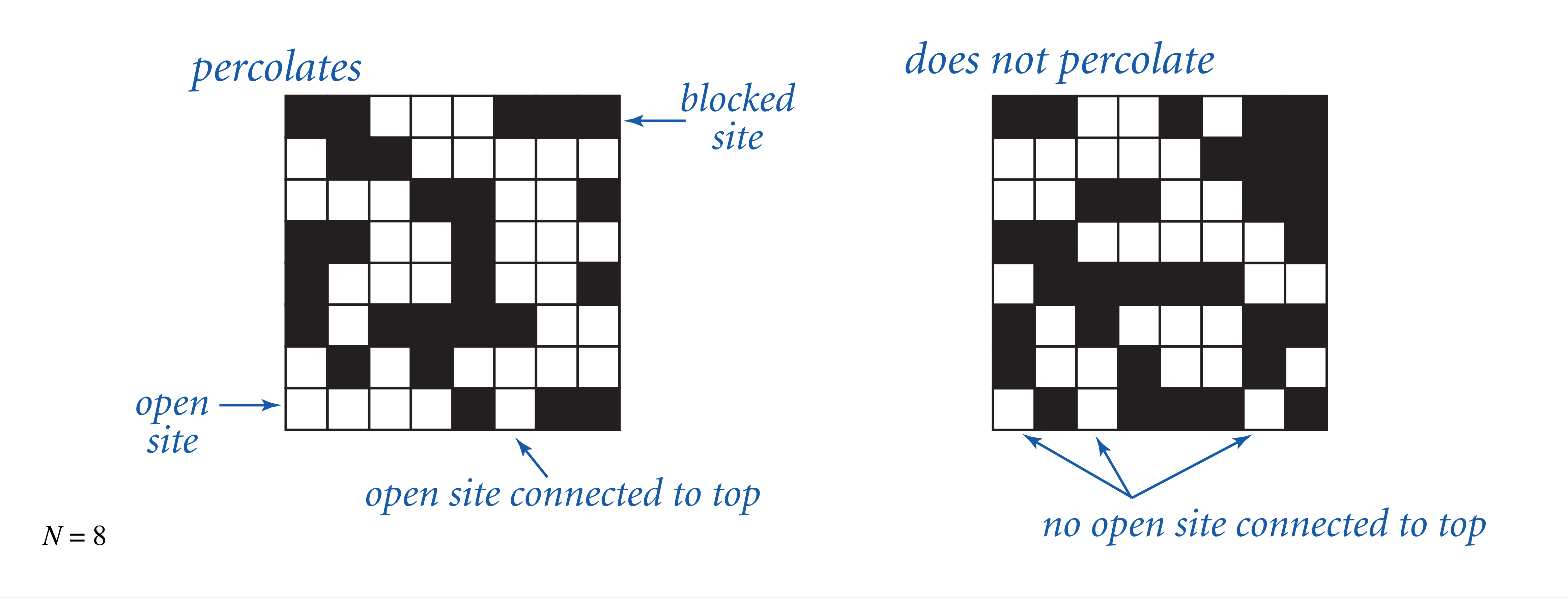


# Percolation

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

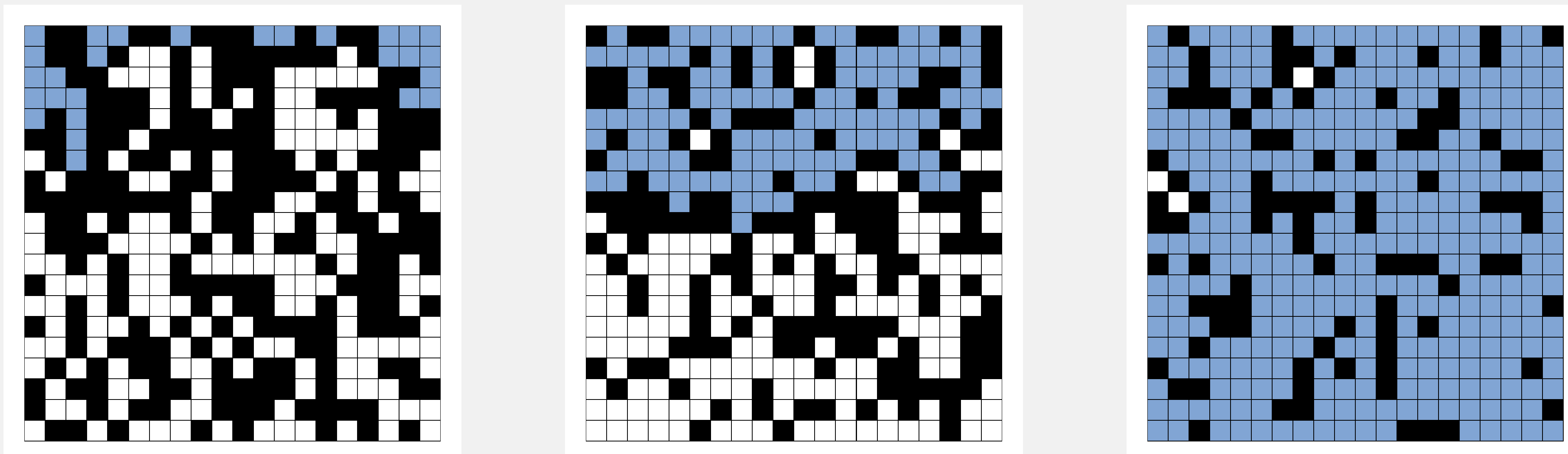
An abstract model for many physical systems:

- $N$-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

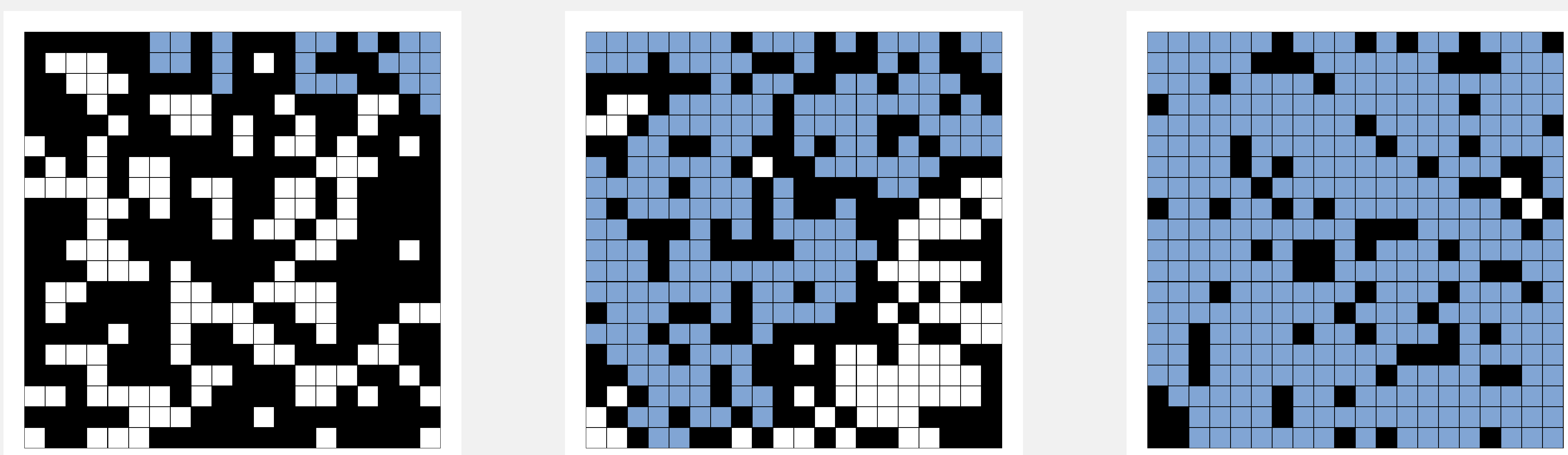
Depends on grid size $N$ and site vacancy probability $p$.



**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)

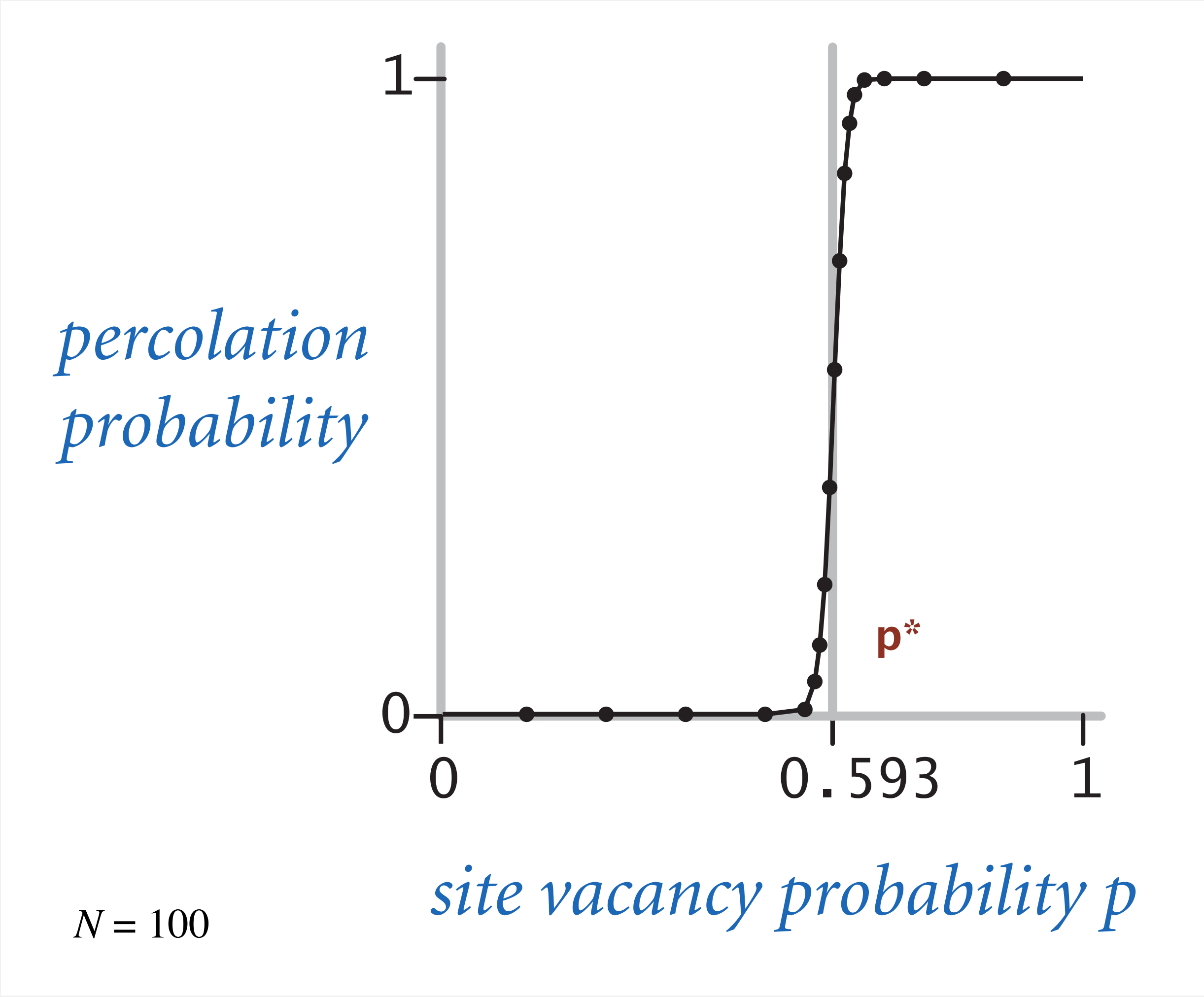
blocked site

# 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 $p^*$ ?



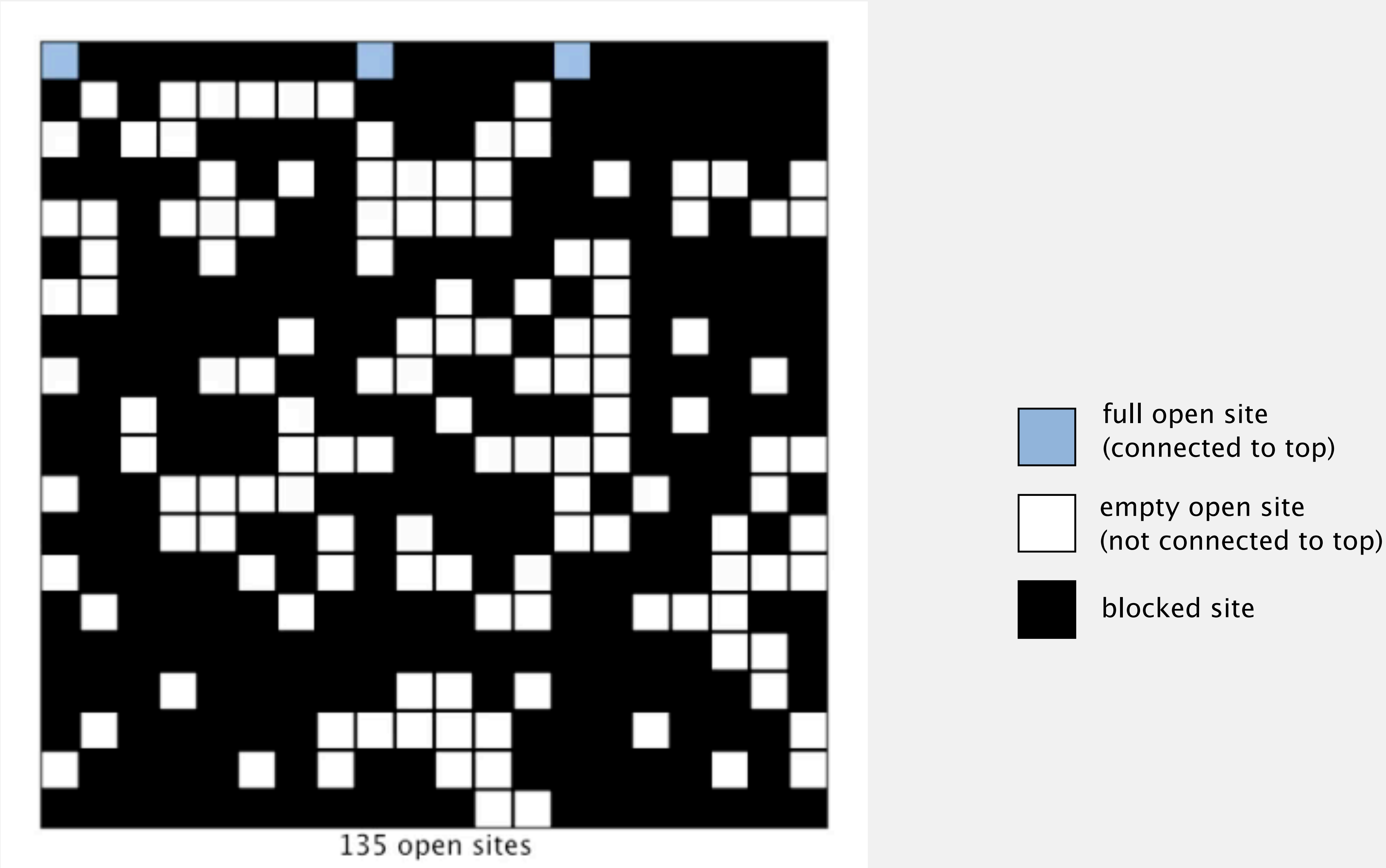
# Monte Carlo simulation

- Determining the threshold $p^*$ is difficult in theory
- Instead, conduct many random simulations, compile statistics.

Le Casino de Monte-Carlo

# 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 $p^*$.
- Repeat many times to get more accurate estimate.




$$\hat{p} = \frac{204}{400} = 0.51$$

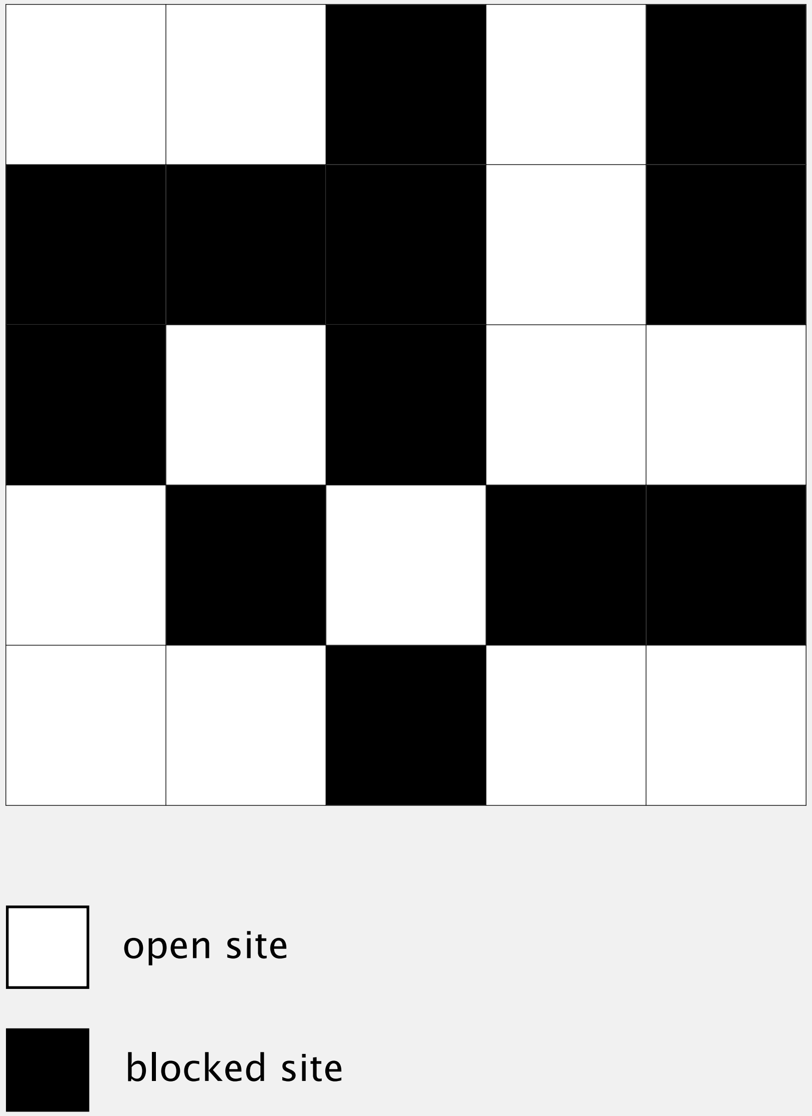
$N = 20$

# 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 problem and use union-find.

$N = 5$

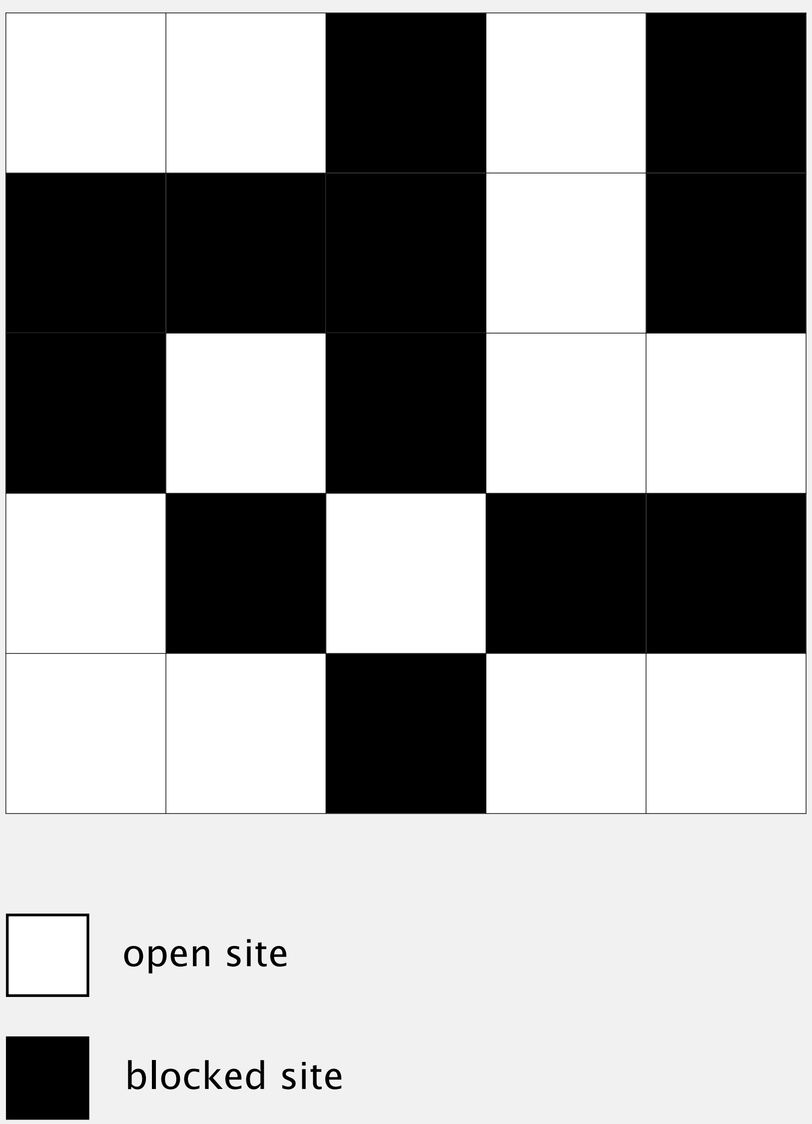


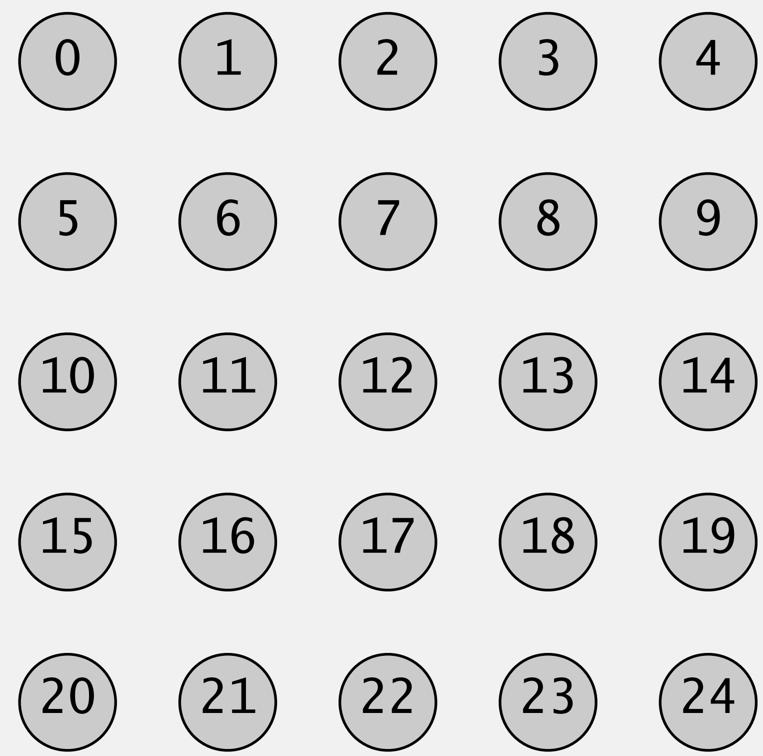
# Dynamic-connectivity solution to estimate percolation threshold

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

- Create an element for each site, named $0$ to $N^2 - 1$.

$N = 5$



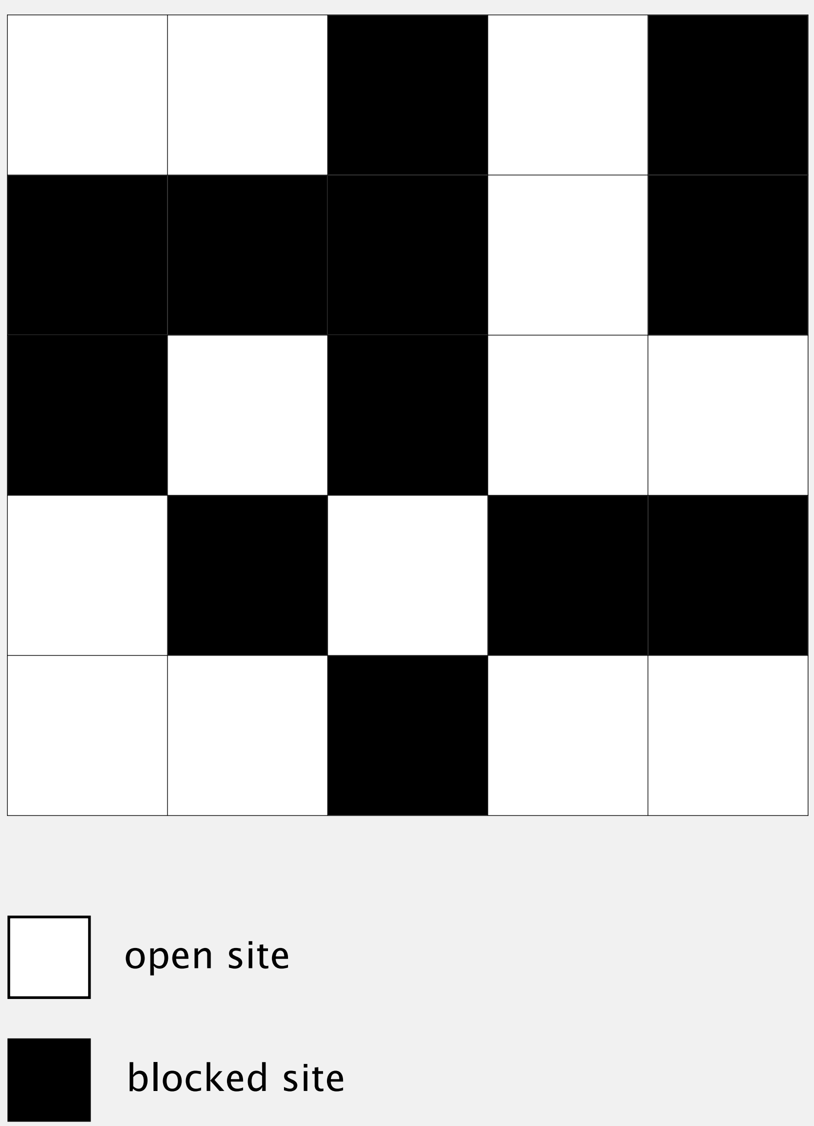
# Dynamic-connectivity solution to estimate percolation threshold

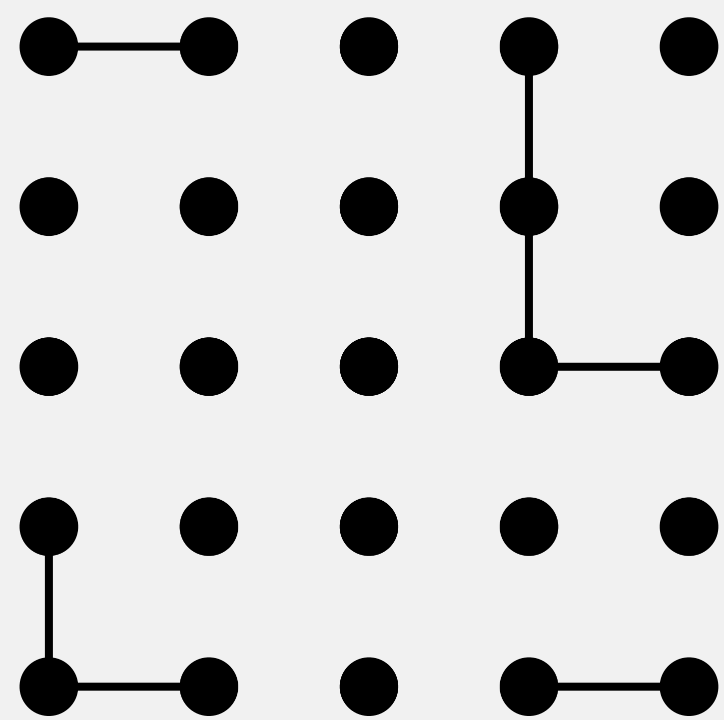
Q. How to check whether an $N$-by-$N$ system percolates?

- Create an element for each site, named $0$ to $N^2 - 1$.
- Add edge between two adjacent sites if both open.

4 possible neighbors: left, right, top, bottom

$N = 5$



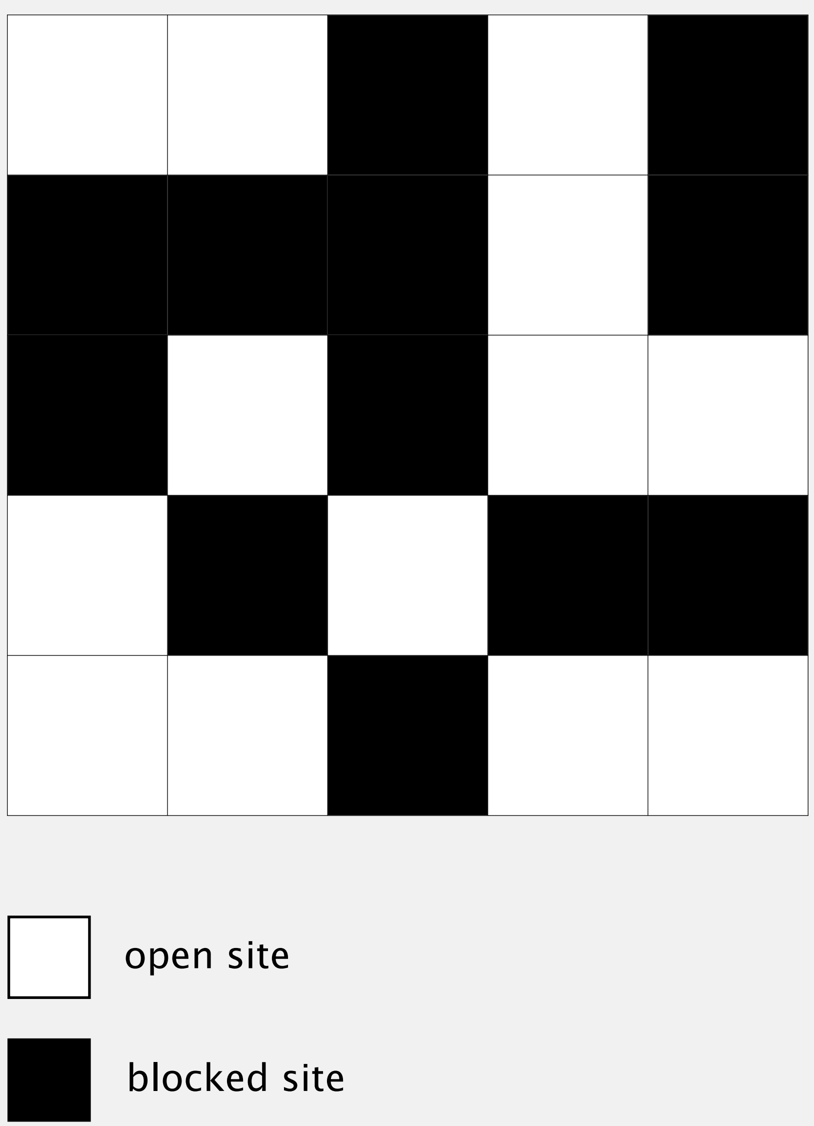
# Dynamic-connectivity solution to estimate percolation threshold

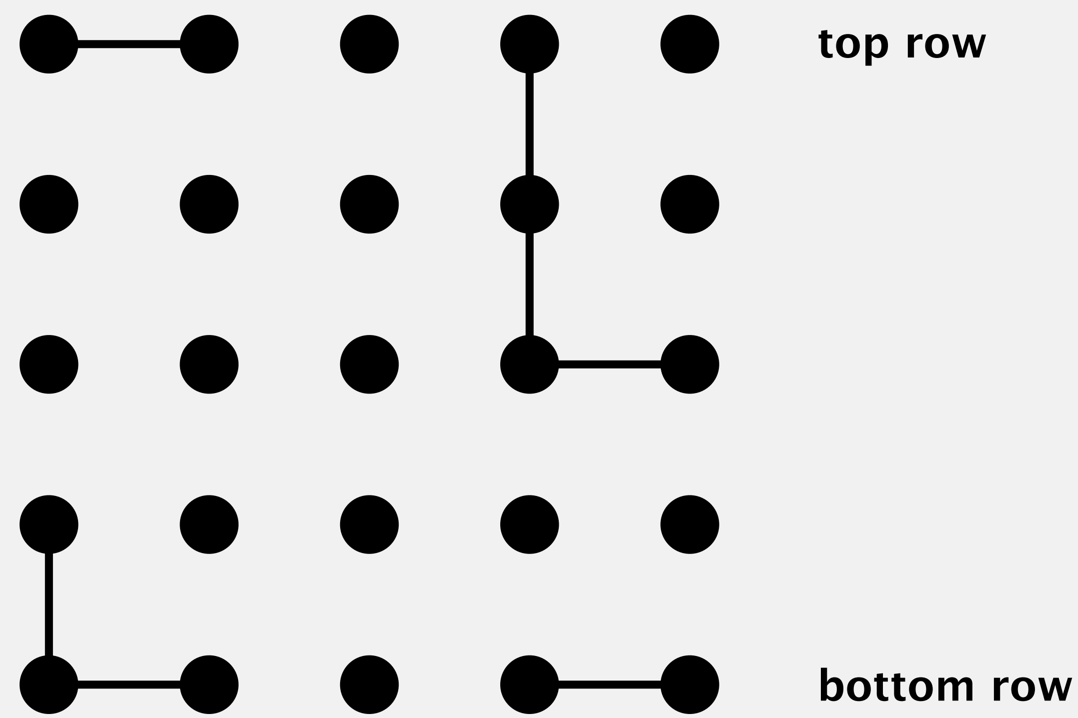
Q. How to check whether an $N$-by-$N$ system percolates?

- Create an element for each site, named $0$ to $N^2 - 1$.
- Add edge between two adjacent sites if both open.
- Percolates iff any site on bottom row is connected to any site on top row.

brute-force algorithm: N 2 connected queries

$N = 5$



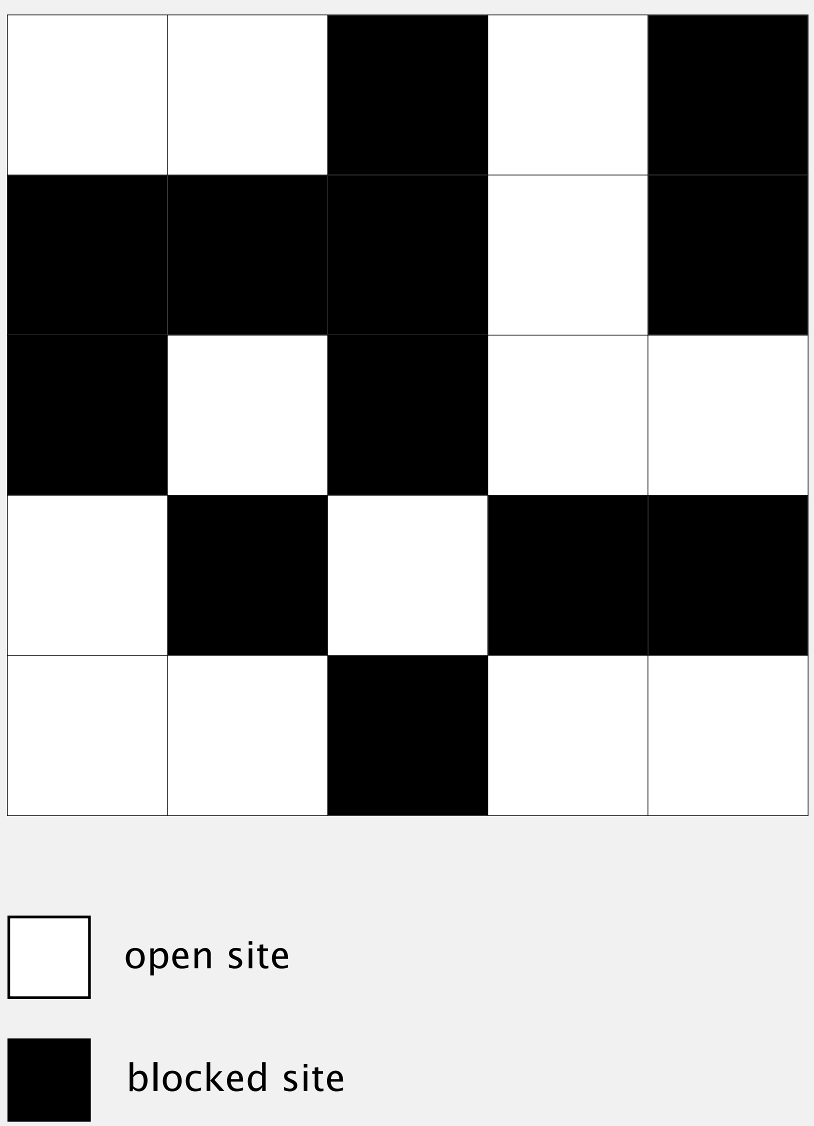
# Dynamic-connectivity solution to estimate percolation threshold

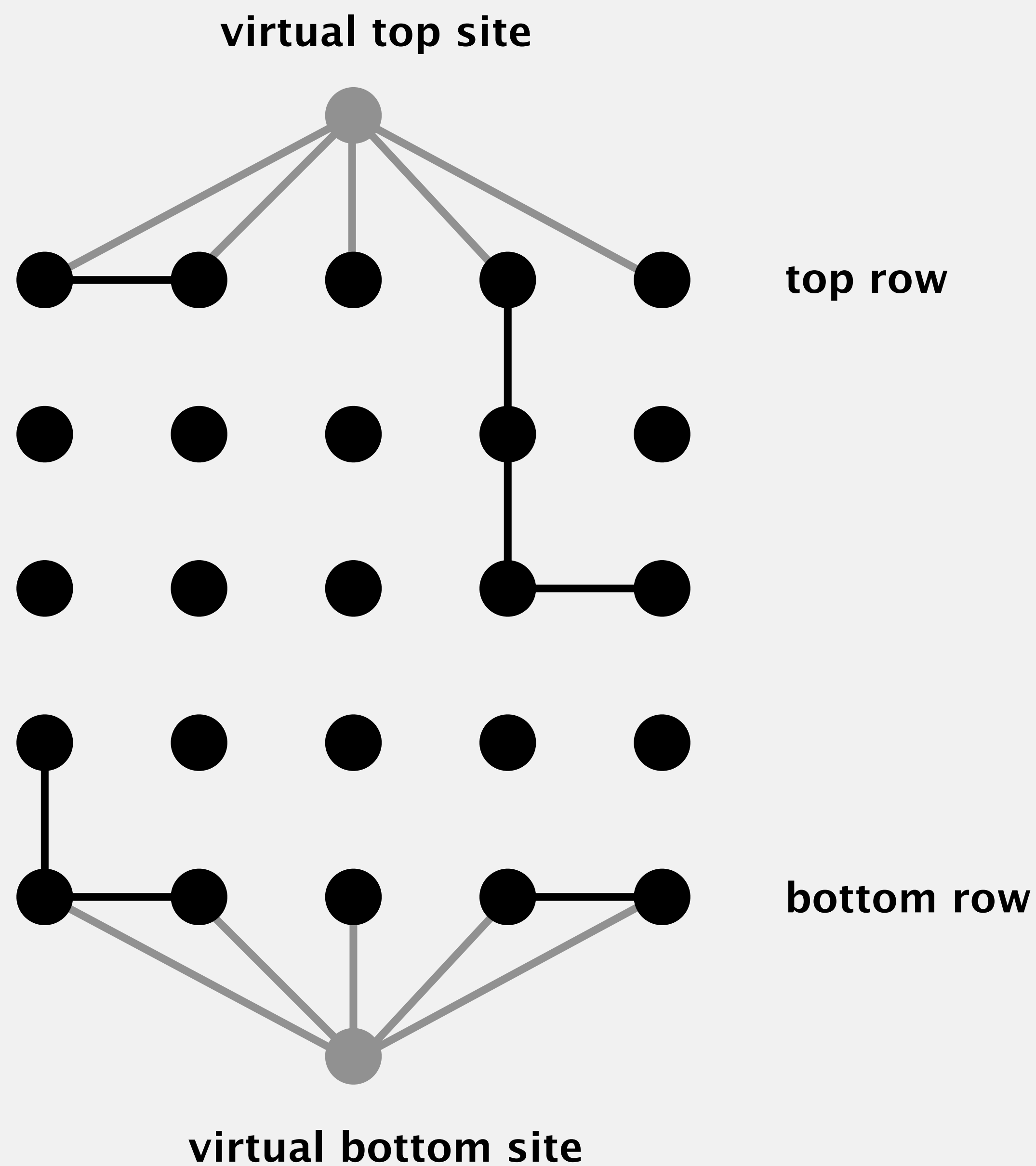
Clever trick. Introduce 2 virtual sites (and edges to top and bottom).

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

more efficient algorithm: only 1 connected query

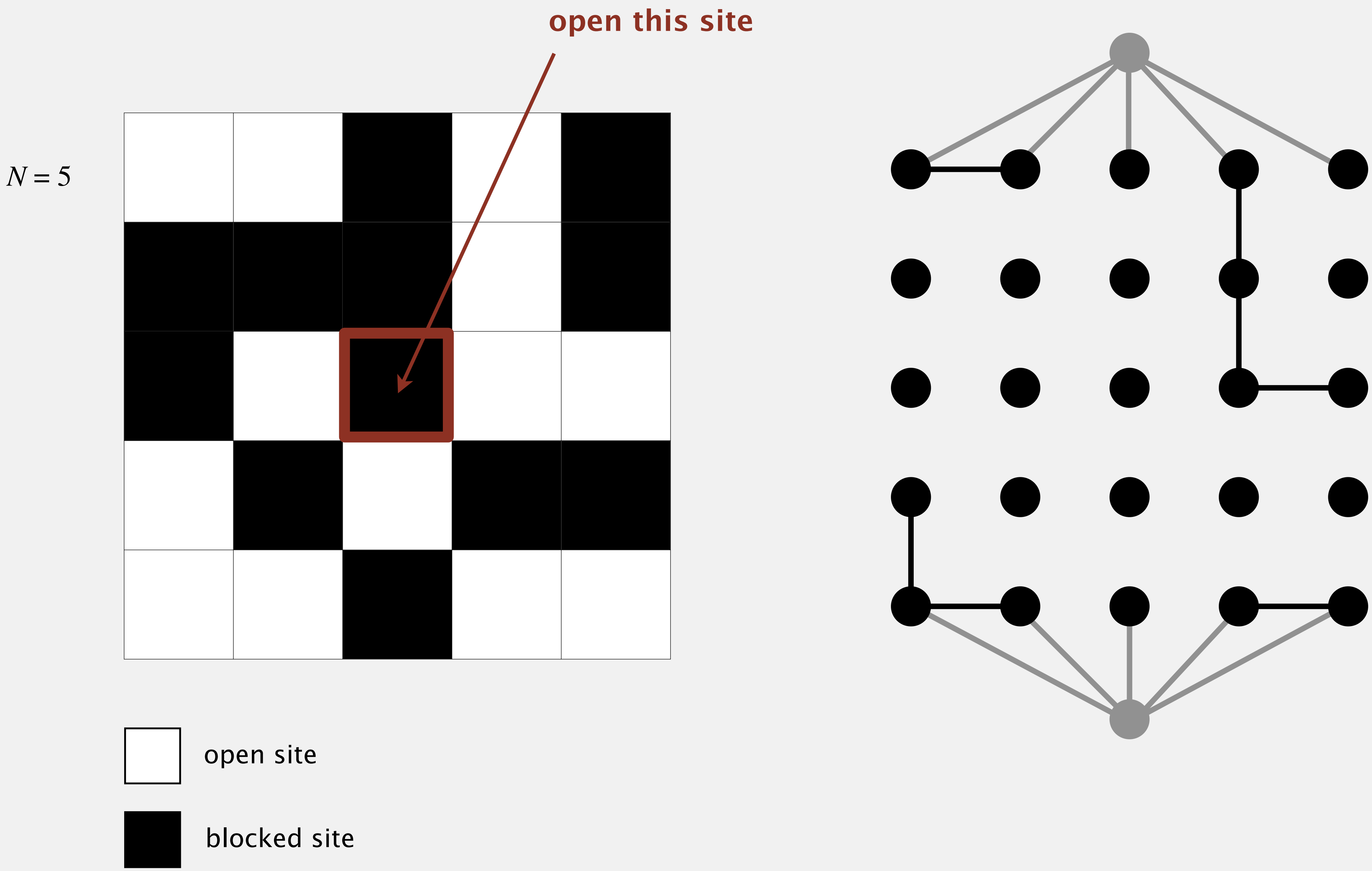
$N = 5$



# Dynamic-connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

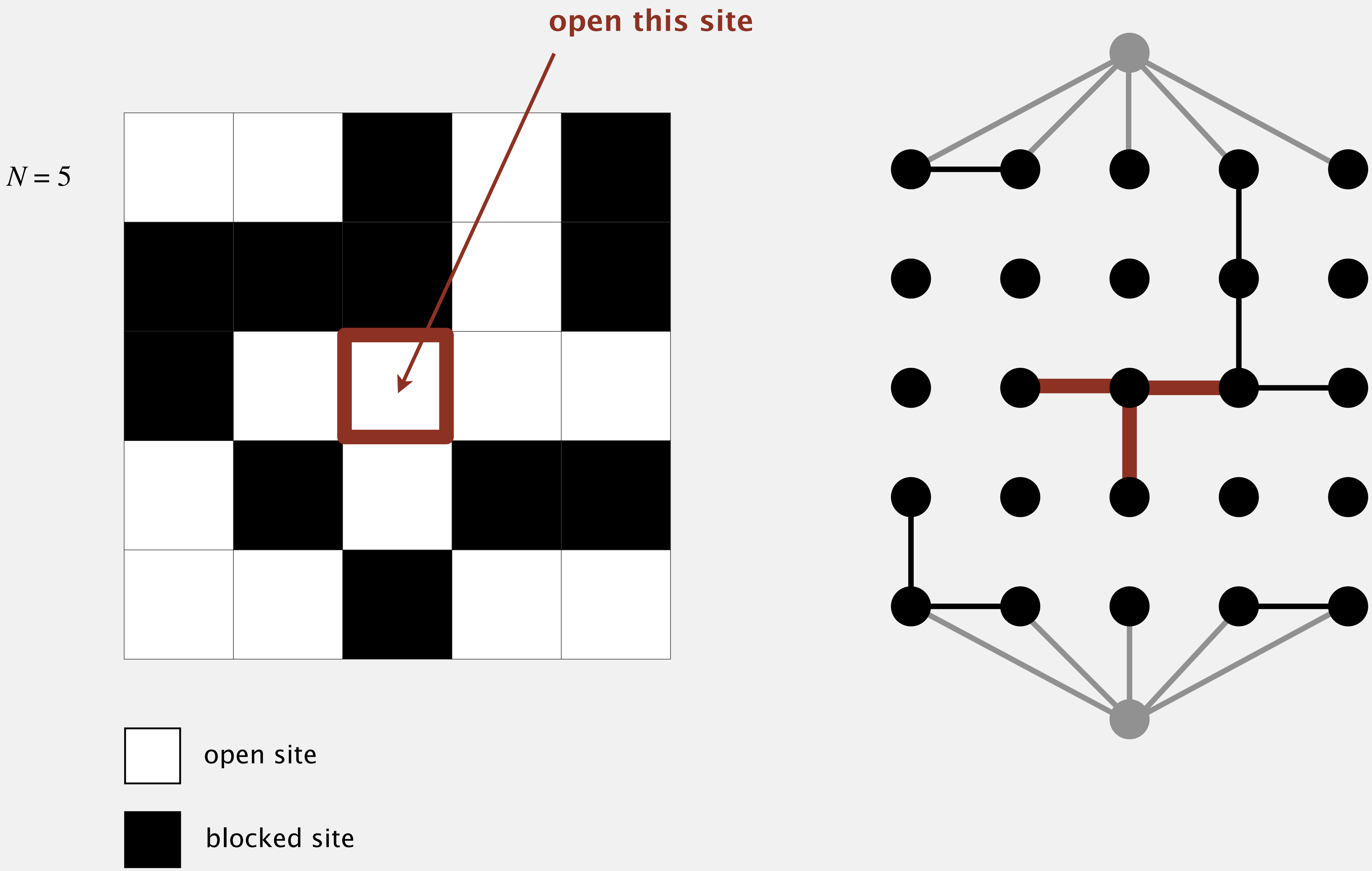


# Dynamic-connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

A. Mark new site as open; add edge to any adjacent site that is open.

adds up to 4 edges

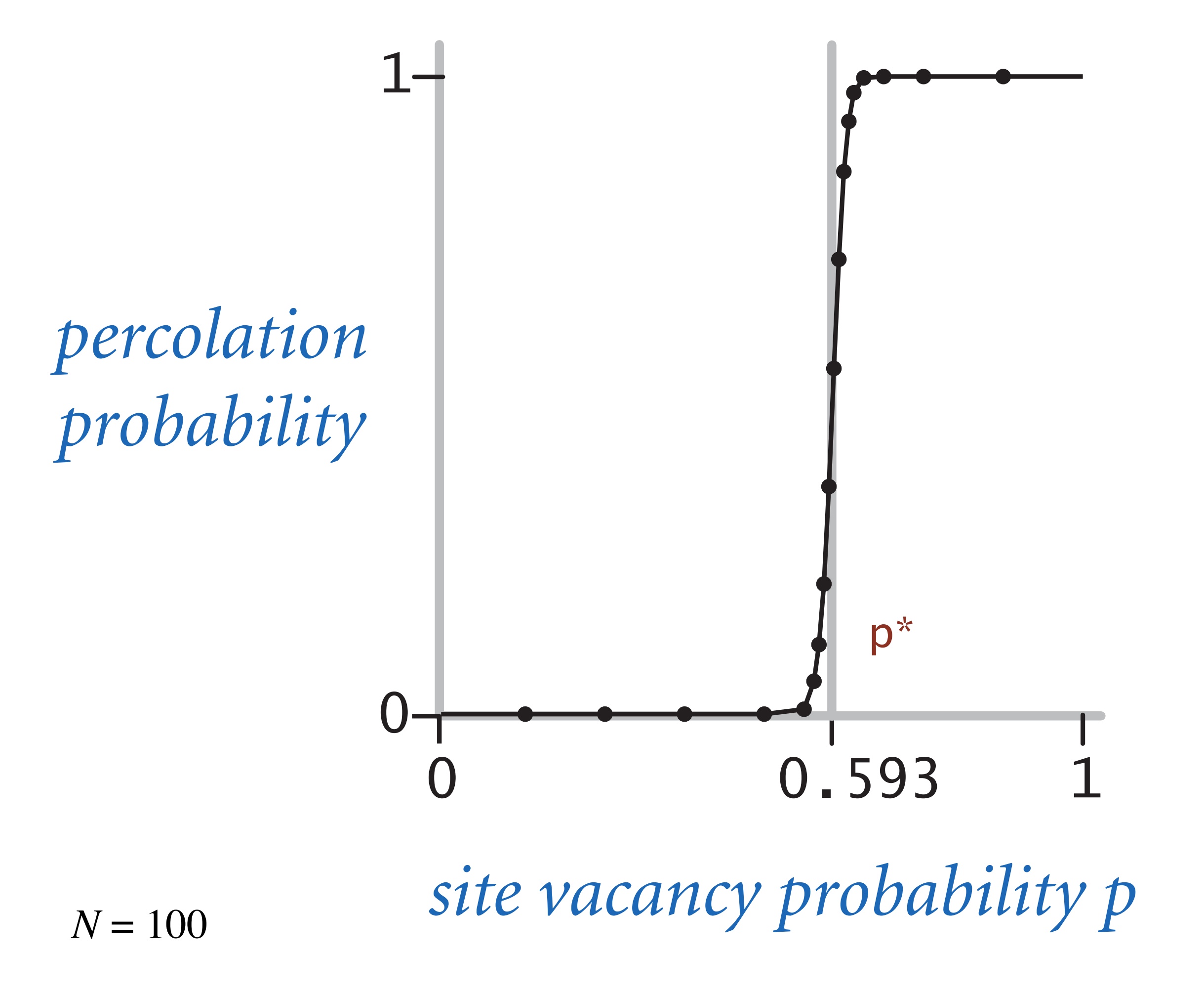


# Percolation threshold

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.