COS 226, Spring 2016

Algorithms and Data Structures

Arvind Narayanan

Princeton University

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

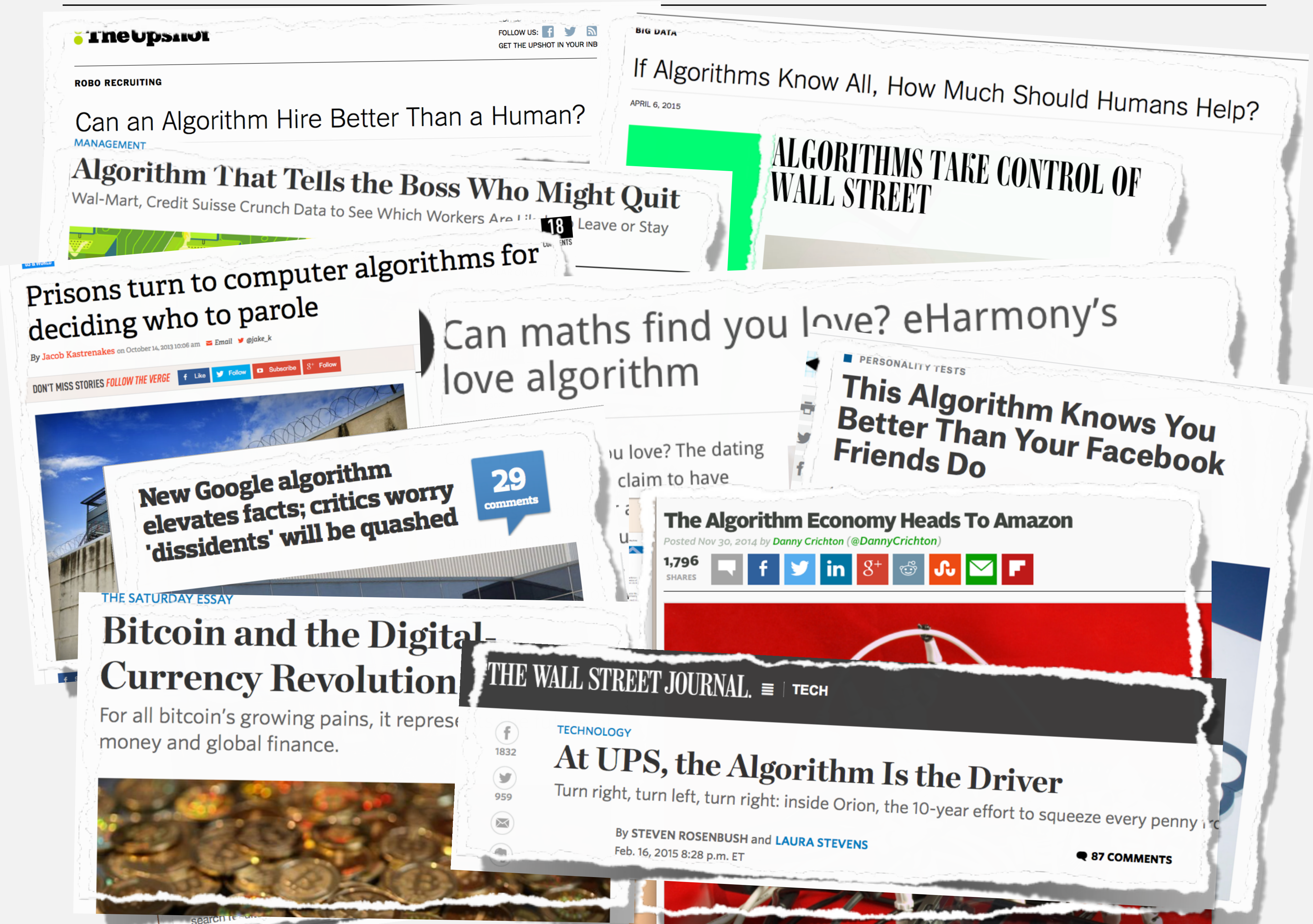
COS 226 course overview

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?

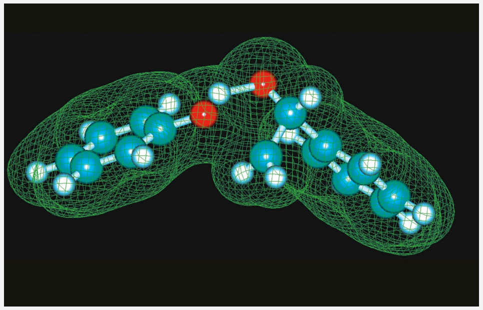


Why study algorithms?

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

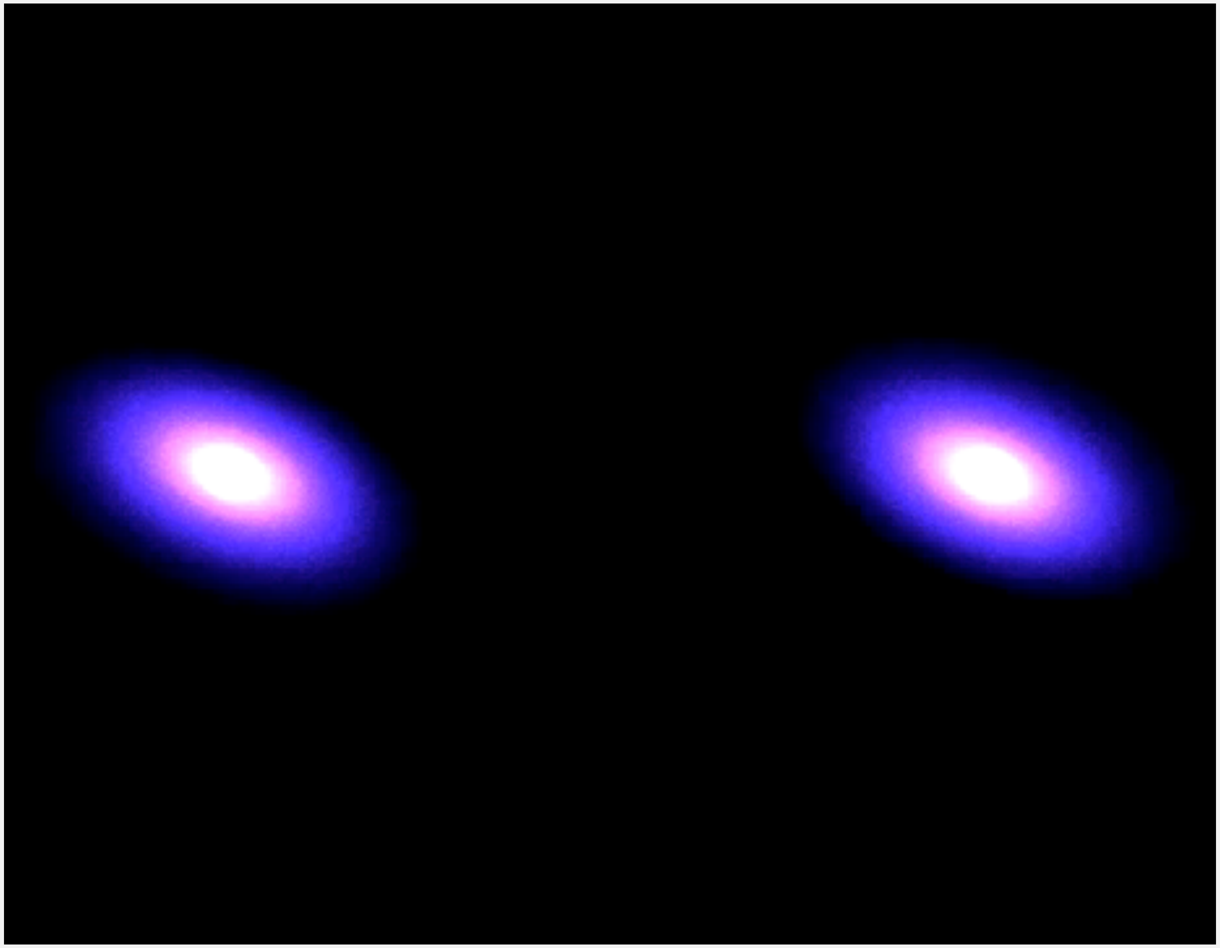
> *" Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube. "*
>
> — *Royal Swedish Academy of Sciences (Nobel Prize in Chemistry 2013)*

Martin Karplus, Michael Levitt, and Arieh Warshel



Why study algorithms?

To solve problems that could not otherwise be addressed.

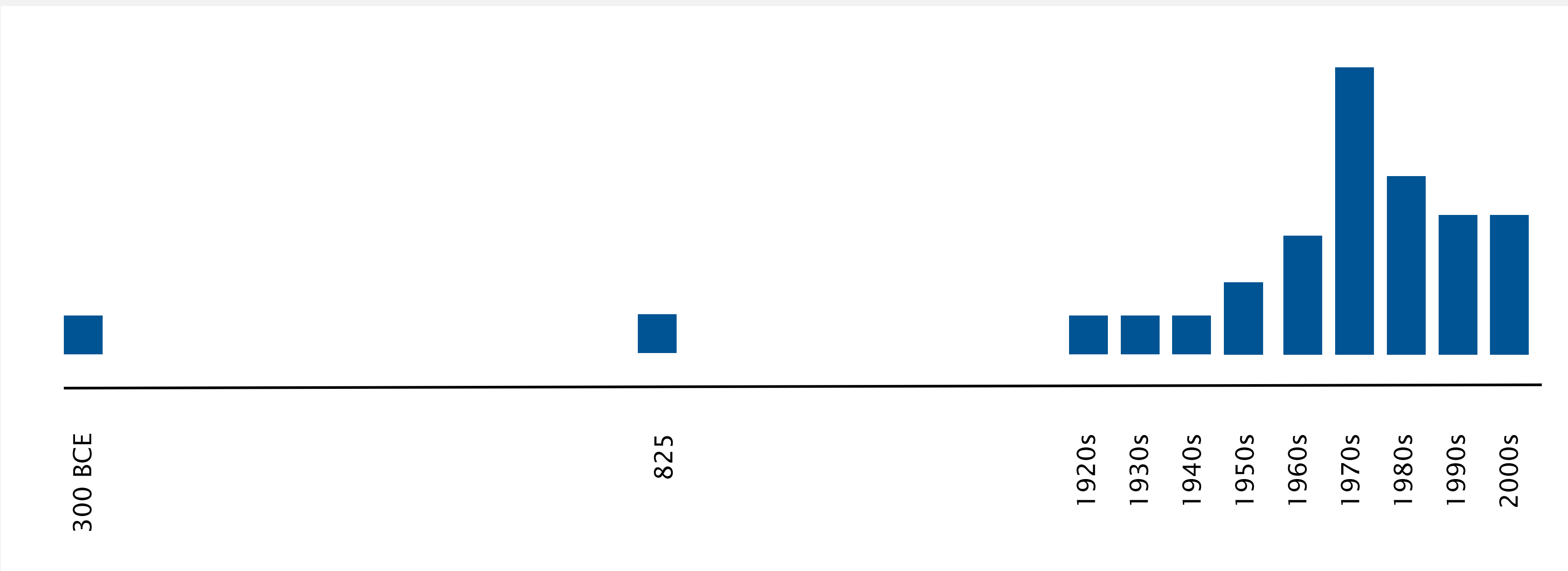


http://www.youtube.com/watch?v=ua7YlN4eL_w

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!



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

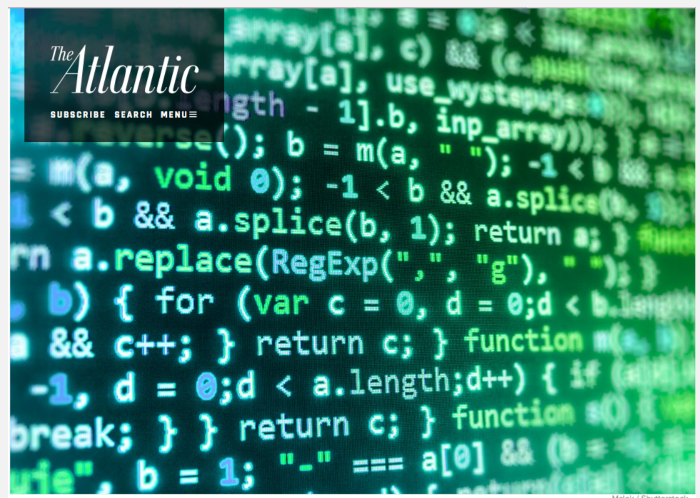
THE JOY OF ALGORITHMS

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



Mclek / Shutterstock

TECHNOLOGY

Not Even the People Who Write Algorithms Really Know How They Work

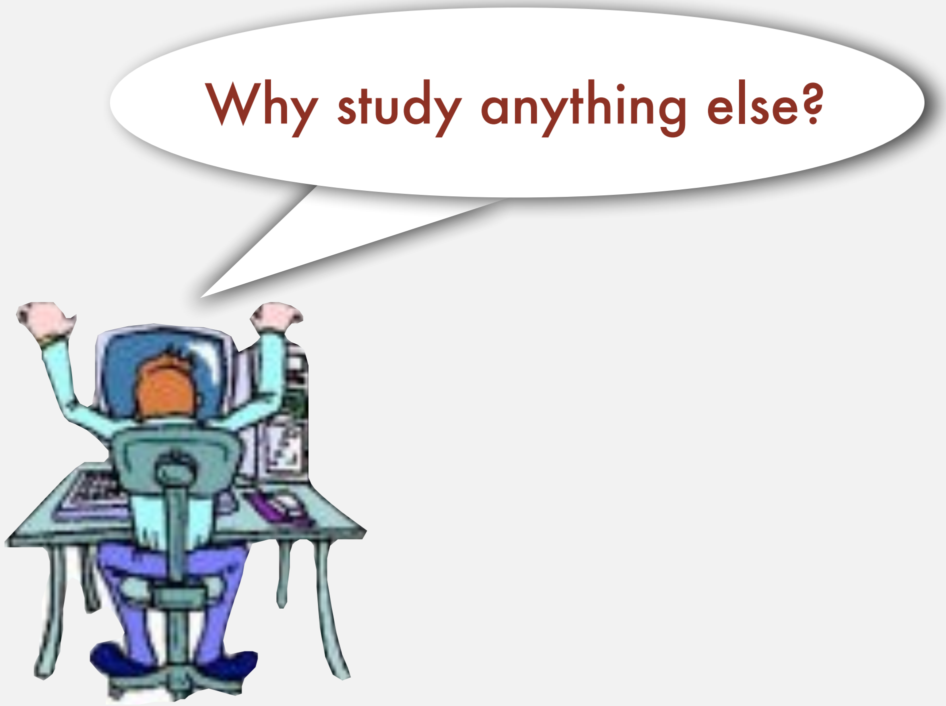
The web's information filters are making assumptions about you based on details that you might not even notice yourself.

Why study algorithms?

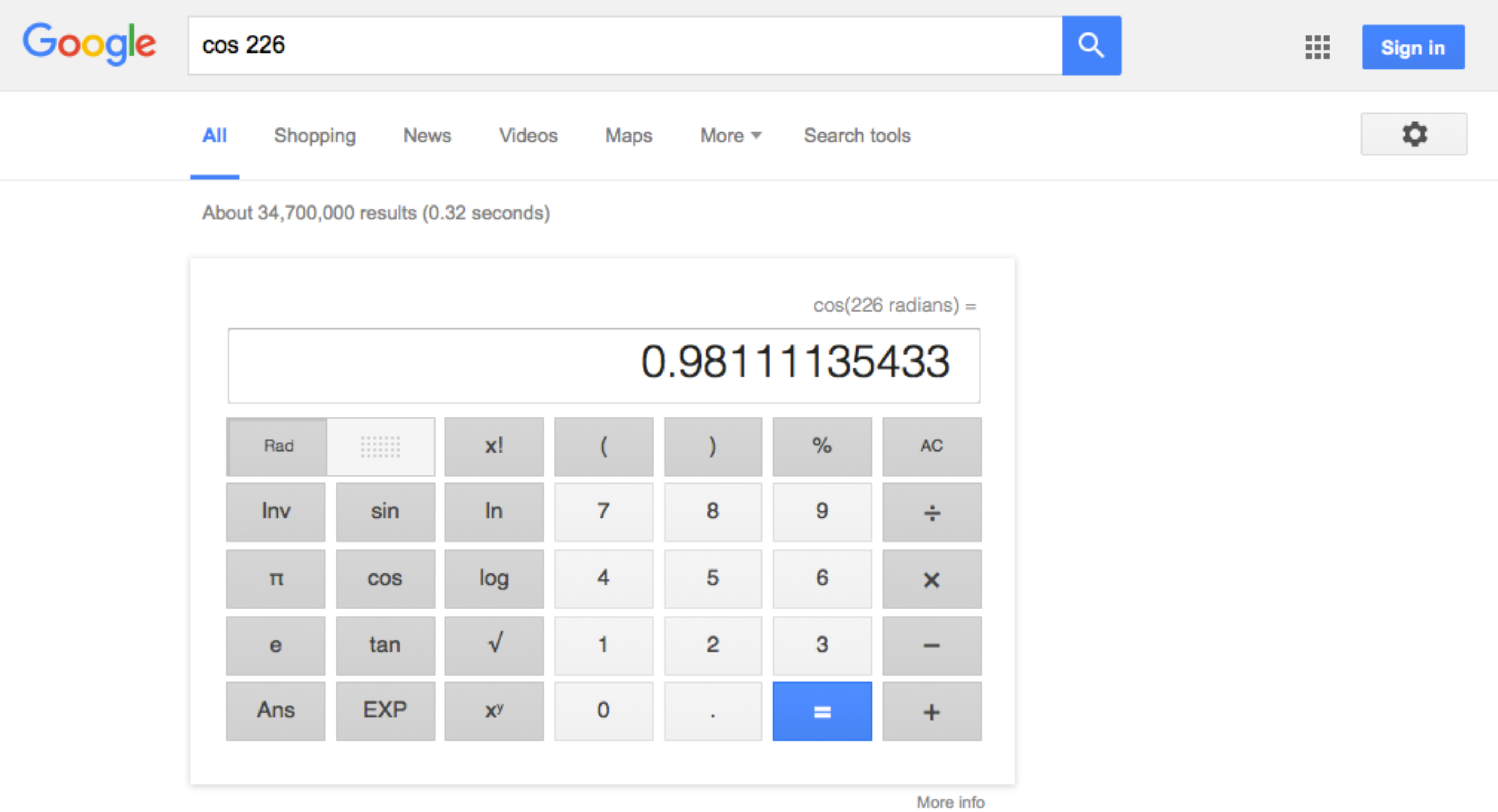
For fun and profit.

Why study algorithms?

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

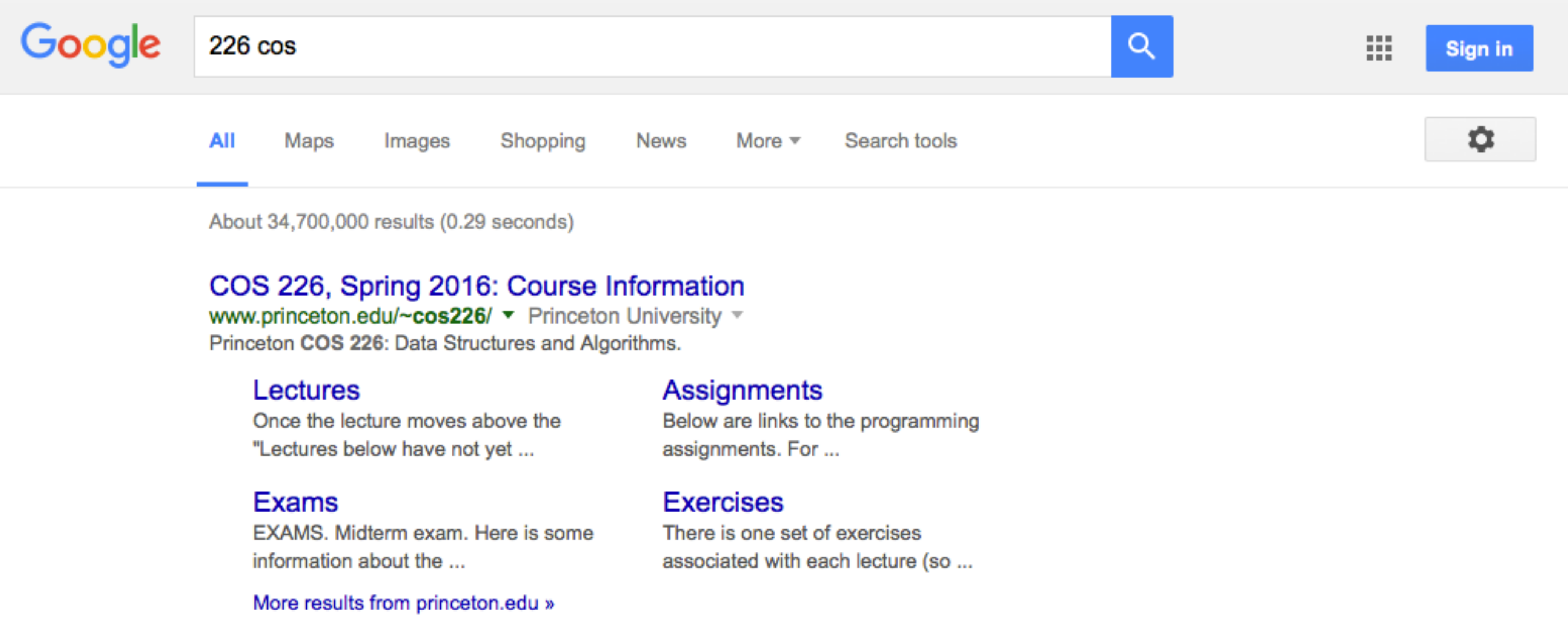


Resources (web)



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

Resources (web)



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

Precepts

Discussion, problem-solving, background for assignments.

	TIME	ROOM	PERSON	OFFICE	HOURS
L01	M W 11–12:20pm	McCosh 10	Arvind Narayan	Sherrerd 308	Wed 2–4pm
L02	M W 11–12:20pm	Jadwin A10	Andy Guna	221 Nassau St. Room 103	Mon 1:00–3:00pm
P01	Th 9–9:50am	Friend 108	Maia Ginsburg †	CS Room 205	Tue 12:30–2:30pm
P02	Th 10–10:50am	Friend 108	Shivam Agarwal	Sherrerd 3rd Floor Common Area	Tue 5–7pm
P02A	Th 10–10:50am	Friend 109	Marc Leef	CS 001B	Mon 6-8pm
P03	Th 11–11:50am	Friend 108	Maia Ginsburg †	CS Room 205	Tue 12:30-2:30pm
P03A	Th 11–11:50am	Friend 109	Ming-Yee Tsang	TBA TBA	Mon 8–10pm
P04	Th 12:30pm–1:20pm	Friend 108	Miles Carlsten	Sherrerd 3rd Floor Common Area	Mon 4–6pm
P05	Th 1:30pm–2:20pm	Friend 112	Sergiy Popovych	CS 241 (front)	Sun 4:30–6:30pm
P06	F 10–10:50am	Friend 108	Andy Guna †	221 Nassau St. Room 103	Mon 1:00–3:00pm
P07	F 11–11:50am	Friend 108	Andy Guna †	221 Nassau St. Room 103	Mon 1:00–3:00pm
P07A	F 11–11:50am	Friend 109	Harry Kalodner	CS 241 (front)	Tues 3–5pm
P99	M 11:00–11:50pm	221 Nassau St. Conference room	Andy Guna	221 Nassau St. Room 103	Mon 1:00–3:00pm

† co-lead preceptors

Coursework and grading

Programming assignments. 45%

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

Exercises. 10%

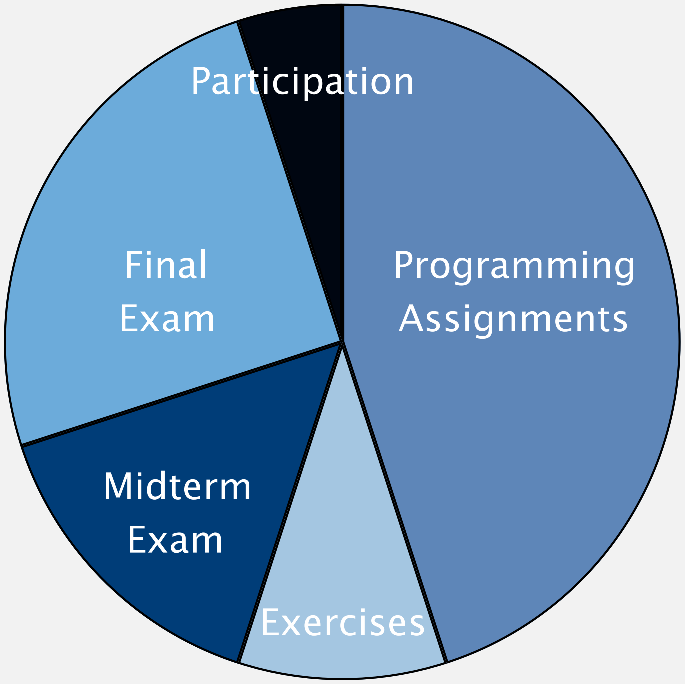
- Due at 11pm on Sundays via Blackboard.
- Collaboration/lateness policies: see web.

Exams. 15% + 25%

- Midterm (in class on Wednesday, March 11).
- Final (to be scheduled by Registrar).

Participation. 5%

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



i▸clicker

Required device for lecture.

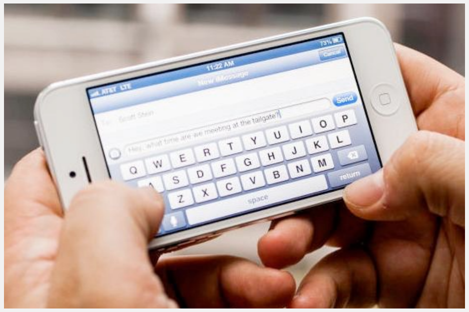
- Any hardware version of i▸clicker.
- Use default frequency AA.
- Available at Labyrinth Books ($25).
- You must register your i▸clicker in Blackboard.

  (sorry, insufficient WiFi in this room to support i▸clicker GO)

save serial number
to maintain resale value

We'll start using them on Wednesday.

Electronic devices: Permitted, but only to enhance lecture.



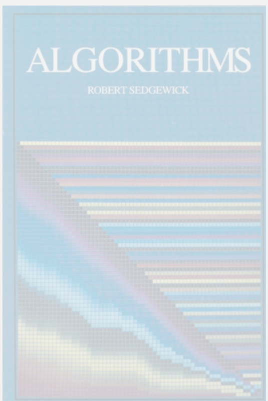
no

no

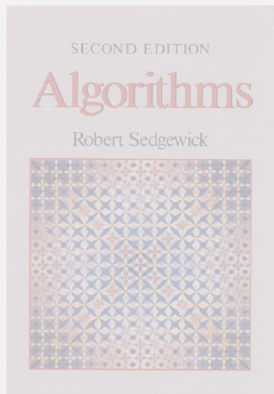
no

Resources (textbook)

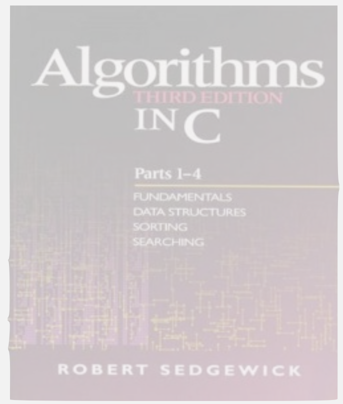
Required reading. Algorithms 4th edition by R. Sedgewick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.



1st edition (1982)



2nd edition (1988)



3rd edition (1997)

4th edition (2011)

Available in hardcover and Kindle.

- Online: Amazon ($60 hardcover, $50 Kindle, $20 rent), ...
- Brick-and-mortar: Labyrinth Books (122 Nassau St.).
- On reserve: Engineering library.

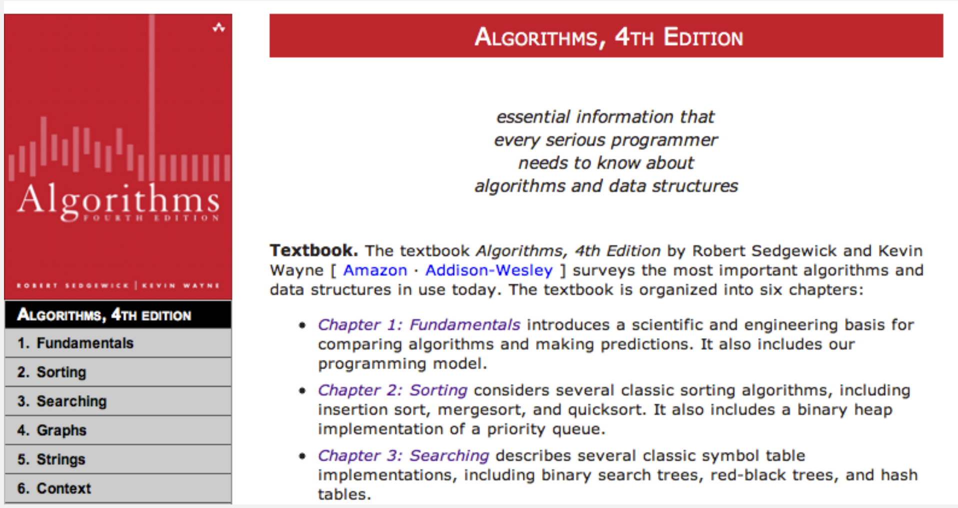
Resources (web)

Course content.

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

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.

piazza

http://piazza.com/princeton/spring2015/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).

Wednesday. Attend traditional/flipped lecture.

Thursday/Friday. Attend precept (everyone).

FOR i = 1 to N

- Sunday: two sets of exercises due.
- Monday: traditional/flipped lecture.
- Tuesday: programming assignment due.
- Wednesday: traditional/flipped lecture.
- Thursday/Friday: precept.

protip: start early

Q+A

Not registered? Go to any precept this week.

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.

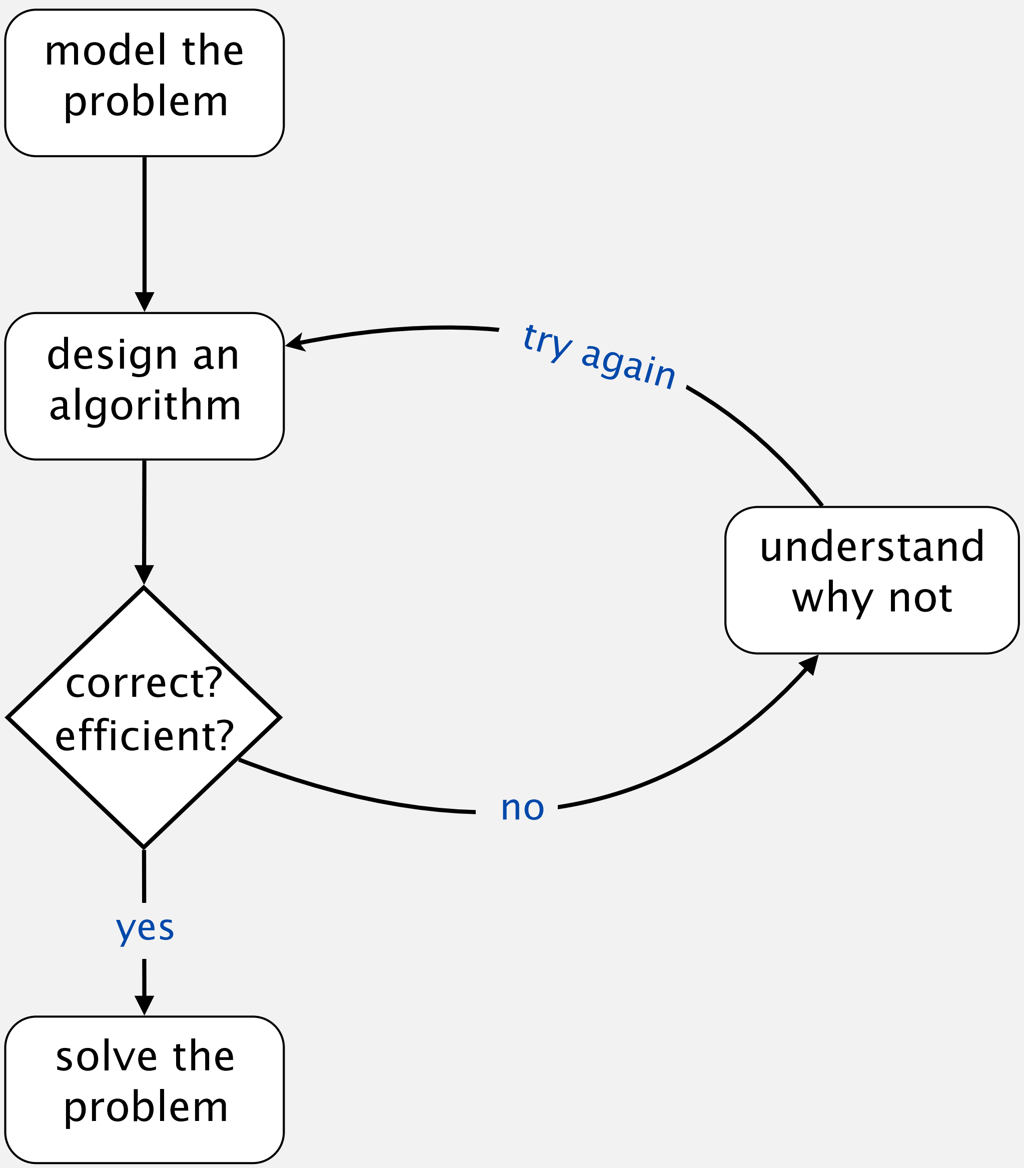
http://algs4.cs.princeton.edu

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:

- Connection command: directly connect two elements with an edge.
- Connection 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? ✕

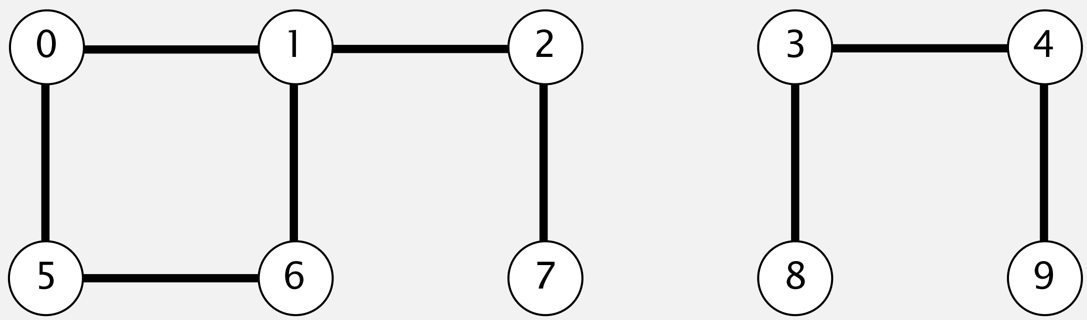
connect 5 and 0

connect 7 and 2

connect 6 and 1

connect 1 and 0

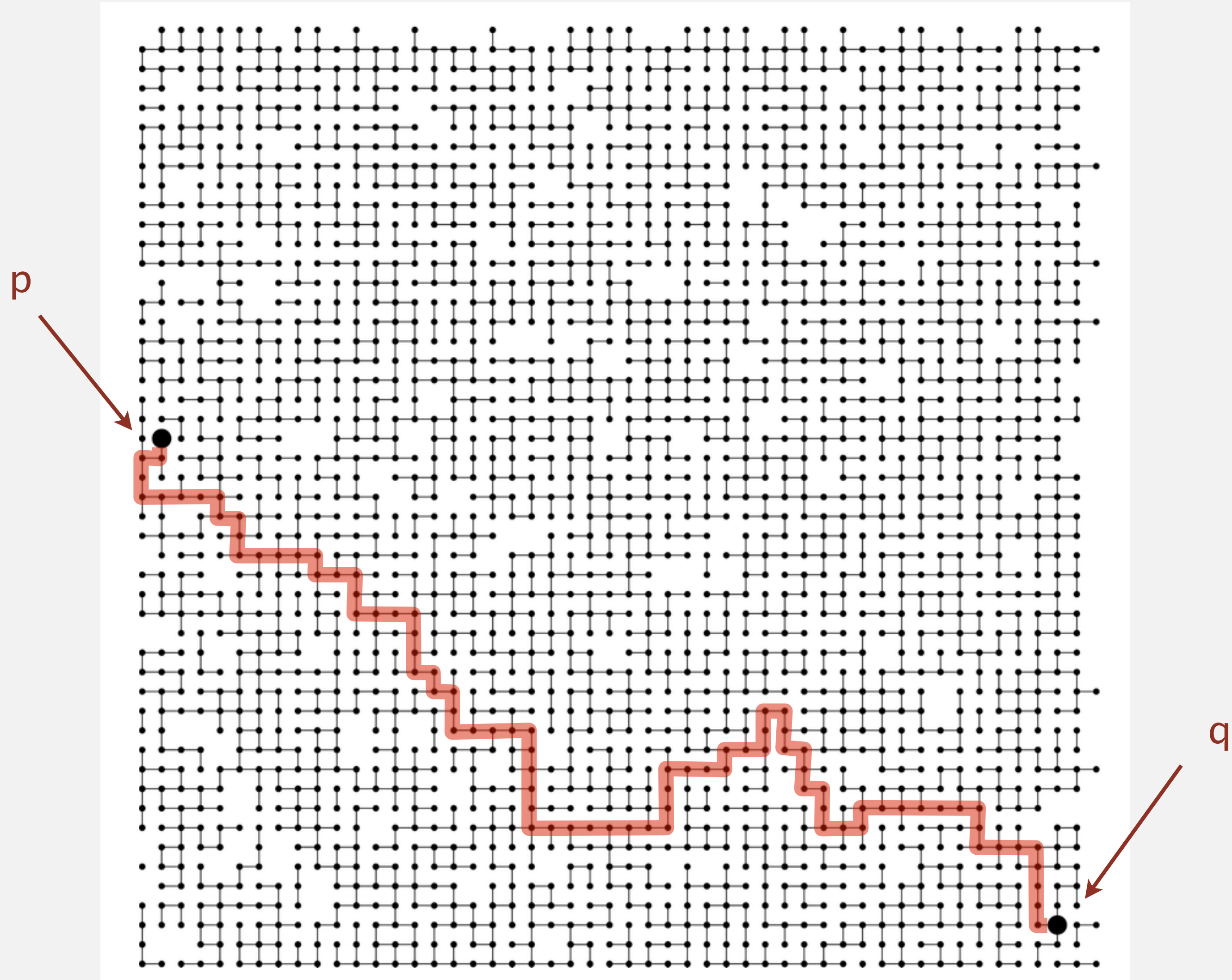
are 5 and 7 connected? ✔



A larger connectivity example

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

finding the path explicitly is a harder problem
(second half of the course)



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.

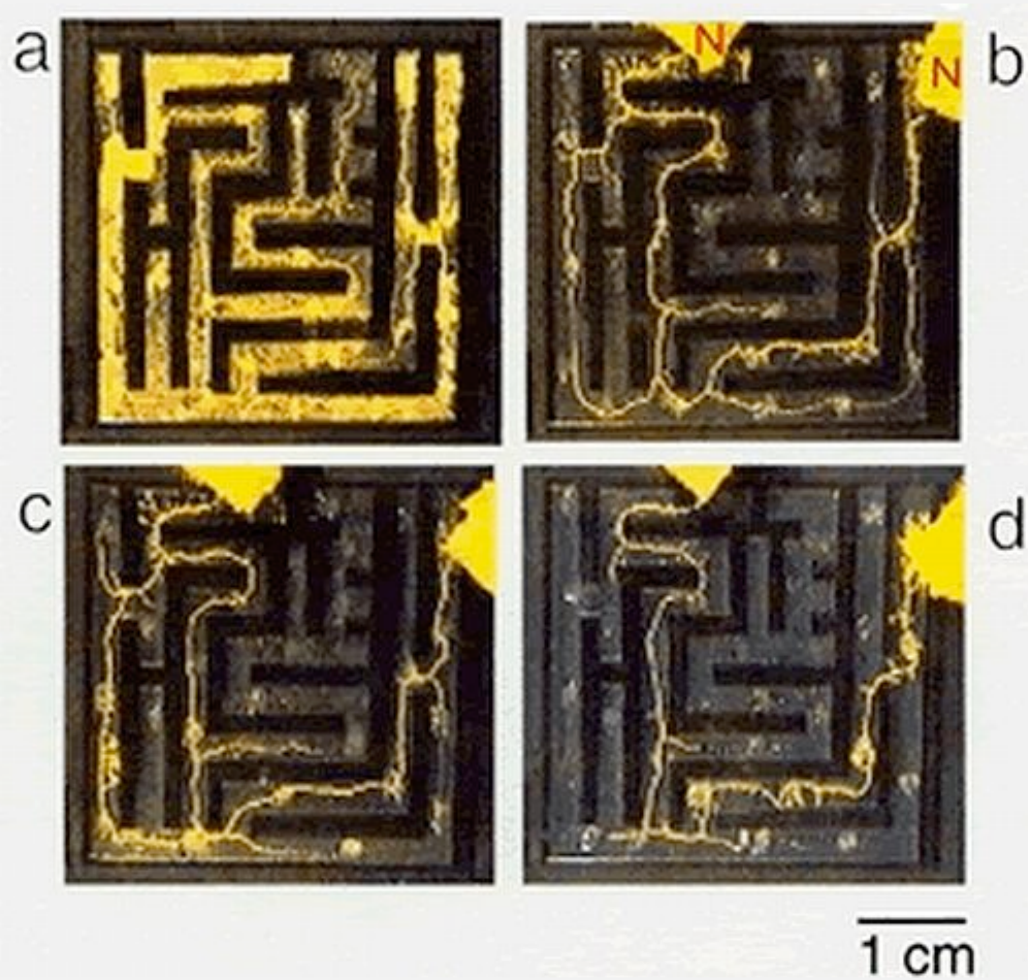
When programming, convenient to name elements 0 to n – 1.

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

Later in the course:
how to translate from names to integers

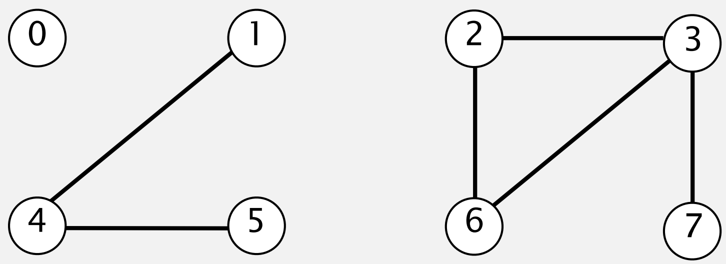
Algorithms in nature

Slime mold in a maze, with food placed at the start and end points



Modeling the connections

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



{ 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.
- Are elements p and q connected?

union(2, 5)

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

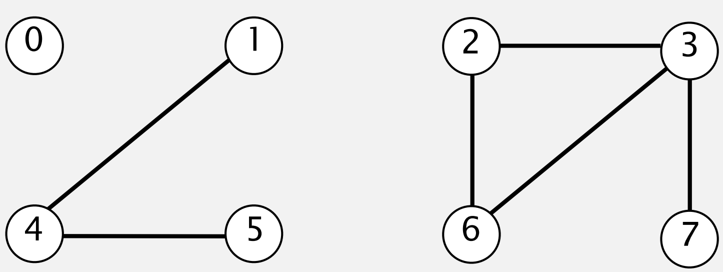
3 disjoint sets

find(5) == find(6) ✔

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

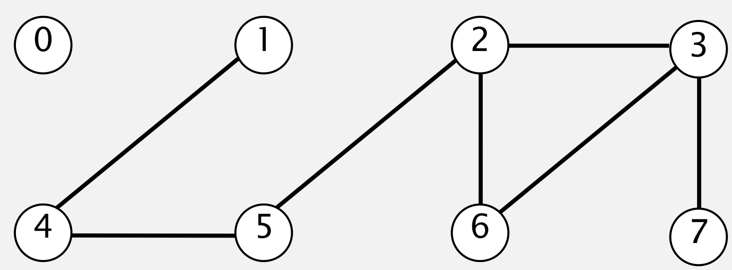
2 disjoint sets

connect 2 and 5



3 connected components

are 5 and 6 connected?



2 connected components

Union–find data type (API)

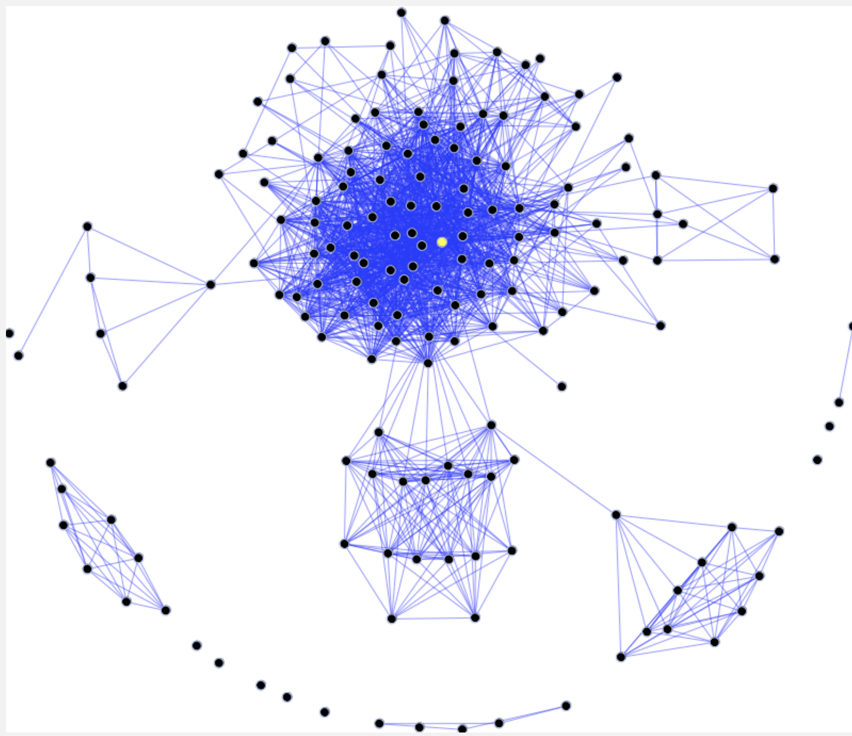
public class UF	
`UF(int n)`	*initialize union–find data structure with n singleton sets (0 to n – 1)*
`void union(int p, int q)`	*merge sets containing elements p and q*
`int find(int p)`	*identifier for set containing element p (0 to n – 1)*

Relatively straightforward expression of problem statement in Java

Up next: two simple union-find algorithms

Difficulty:

- Number of elements n can be huge.
- Number of operations m can be huge.
- Union and find operations can be intermixed.



Data could come from large social network with billions of nodes

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

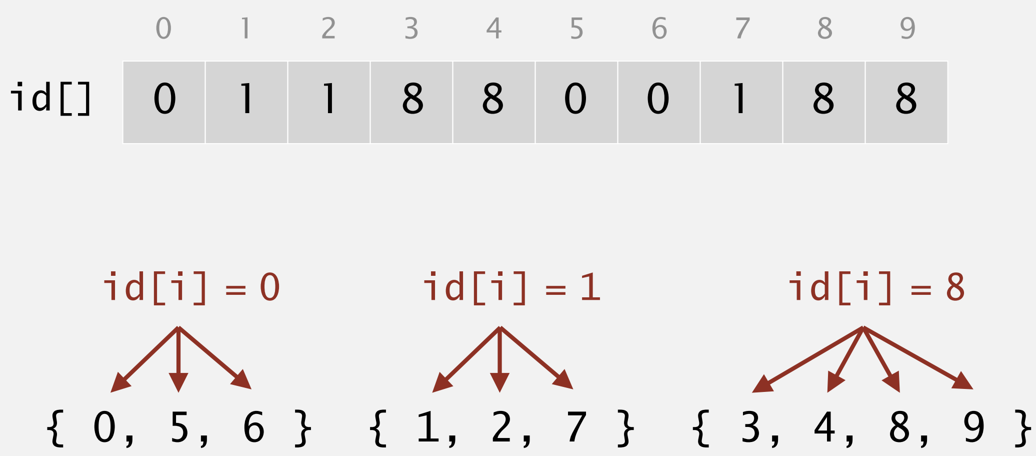
1.5 Union–Find

- 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`.



3 disjoint sets

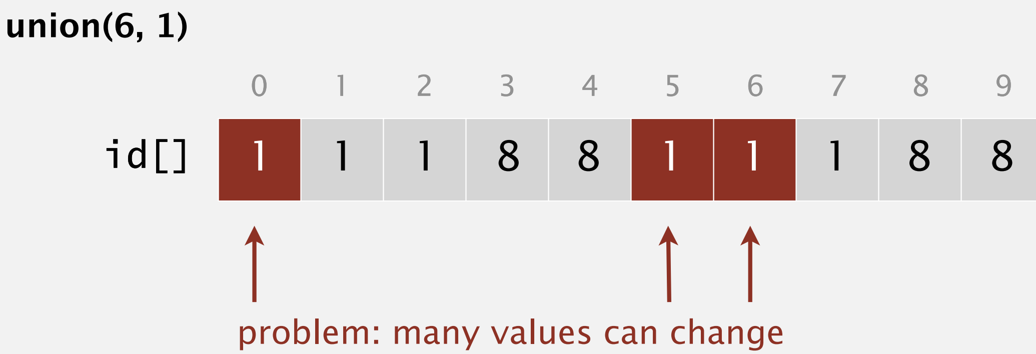
Q. How to implement `find(p)`?

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`.

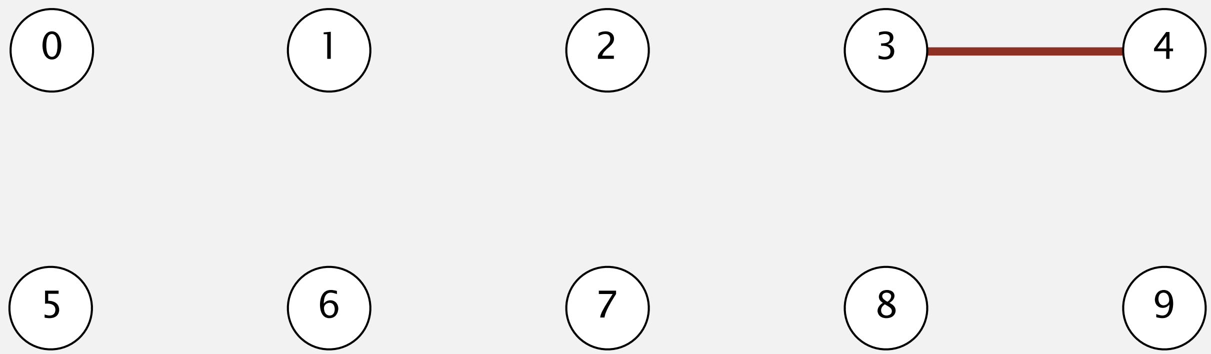


Q. How to implement `union(p, q)`?

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

Quick-find demo

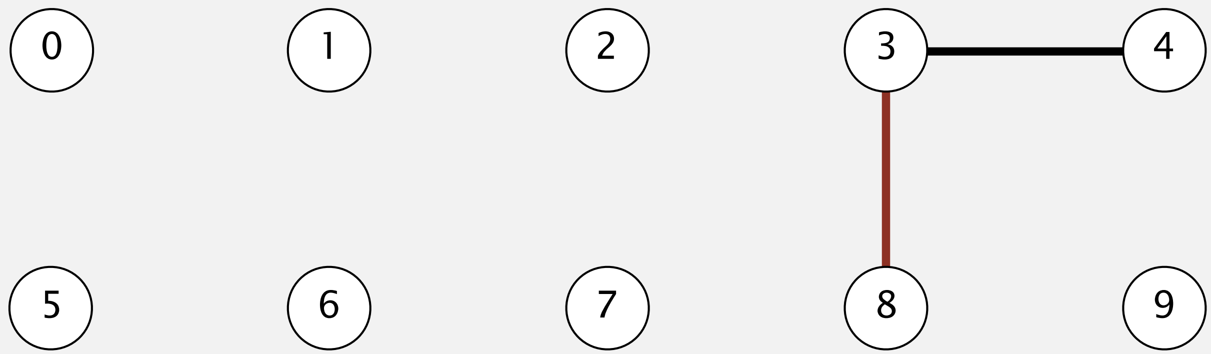
union(4, 3)



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

Quick-find demo

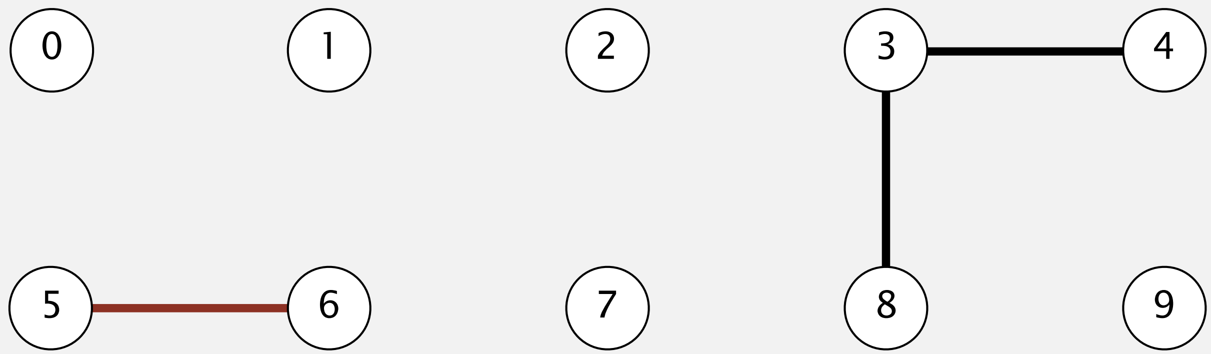
union(3, 8)



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

Quick-find demo

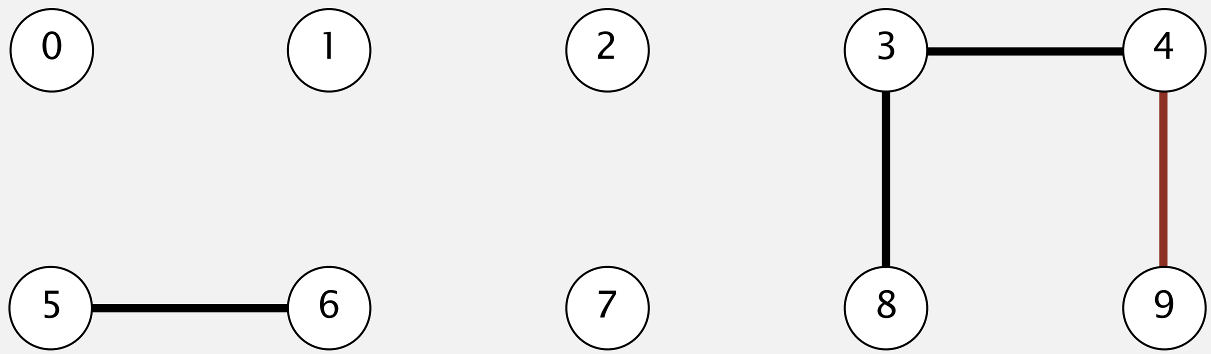
union(6, 5)



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

Quick-find demo

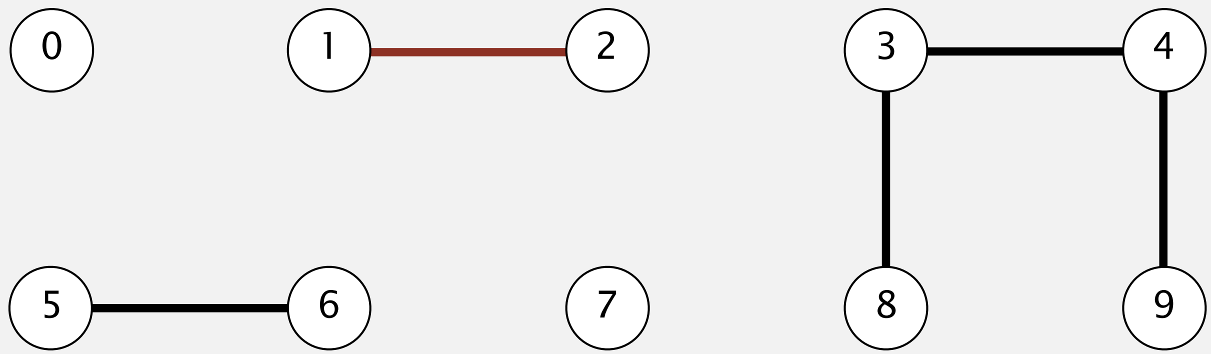
union(9, 4)



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

Quick-find demo

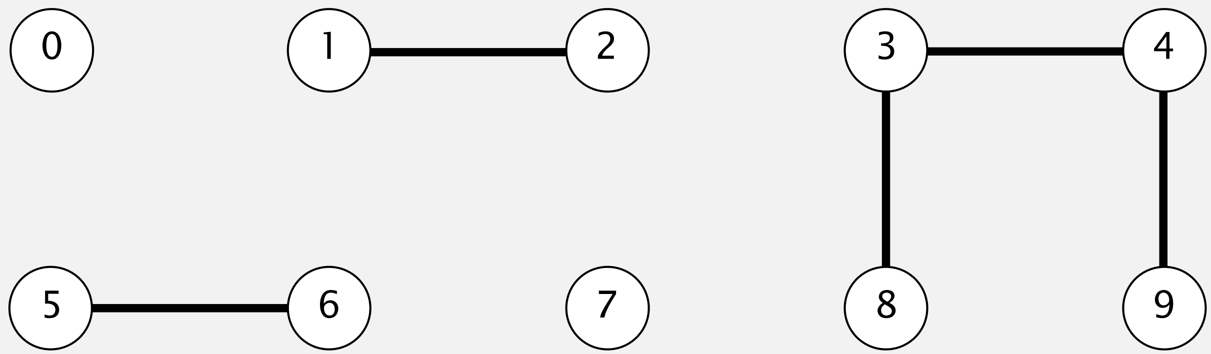
union(2, 1)



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

Quick-find demo

connected(8, 9)



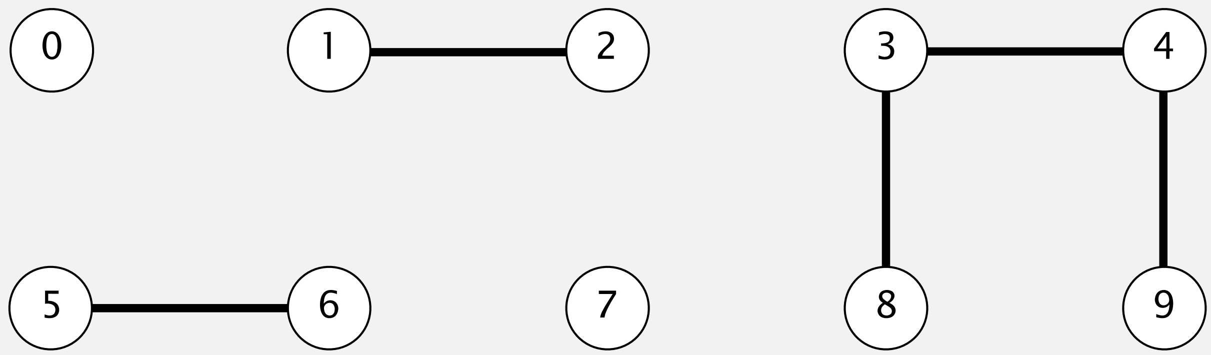
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	5	5	7	8	8

↑ ↑

true

Quick-find demo

connected(5, 0)



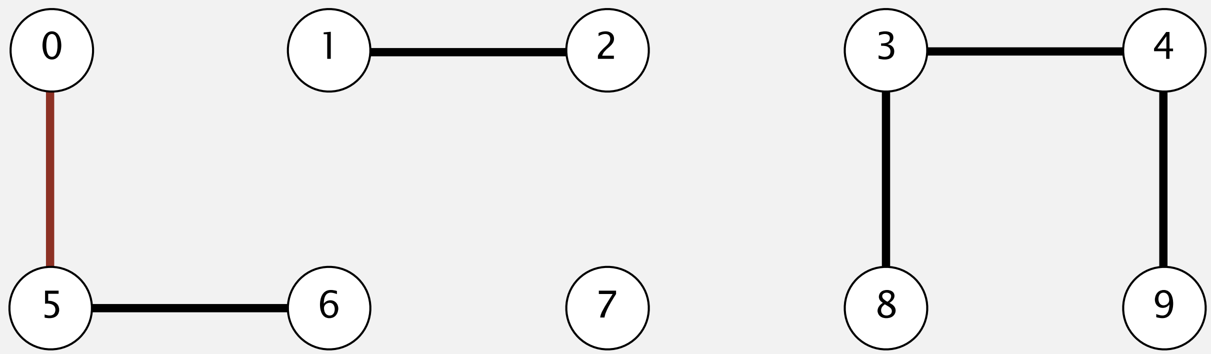
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	5	5	7	8	8

↑ (0) ↑ (5)

false

Quick-find demo

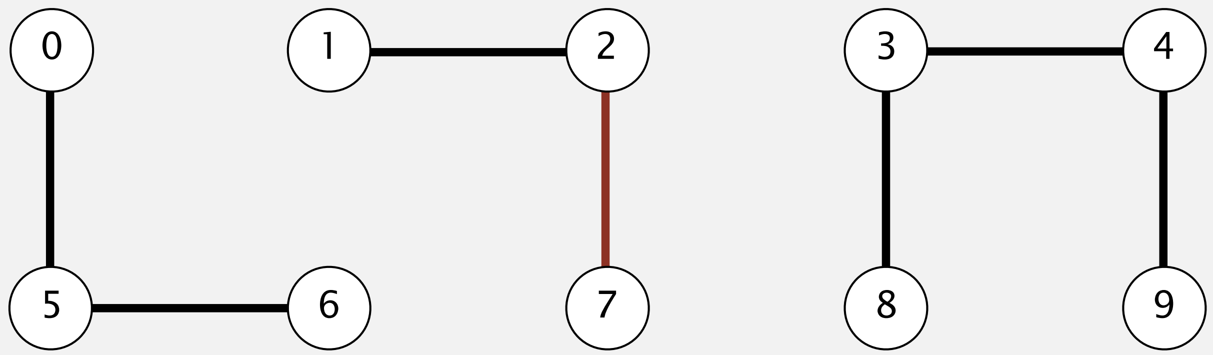
union(5, 0)



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

Quick-find demo

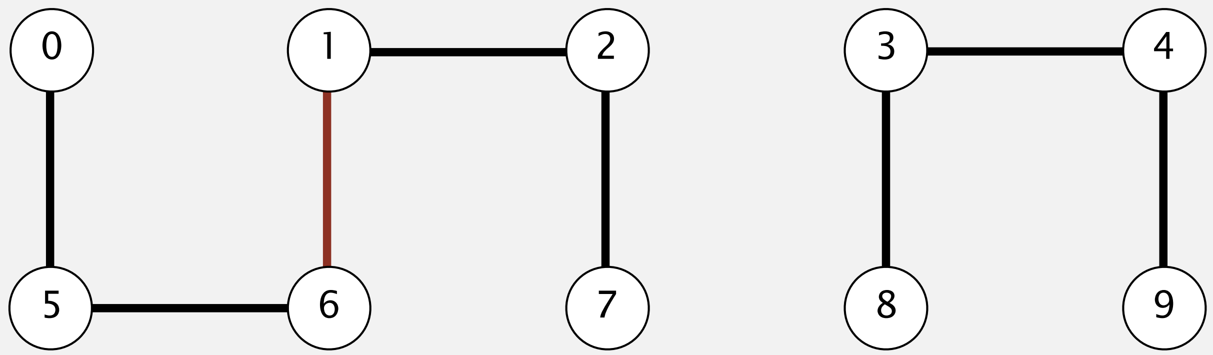
union(7, 2)



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

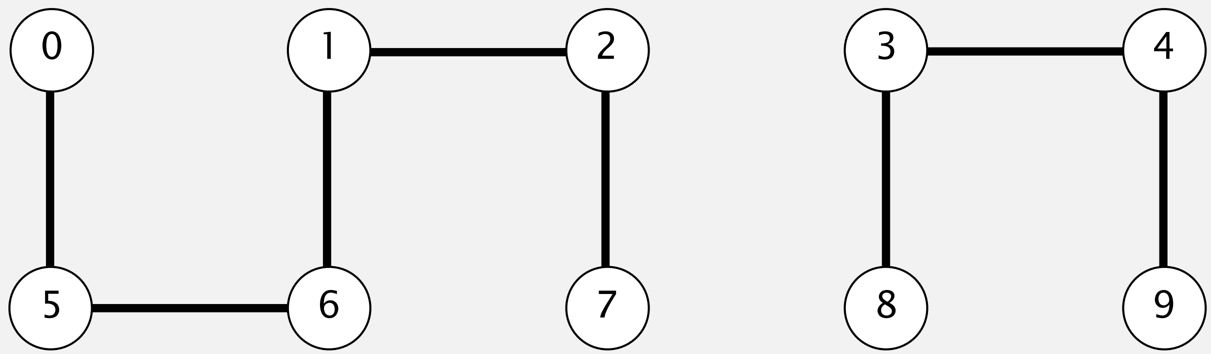
Quick-find demo

union(6, 1)



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

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]
(n+2 to 2n+2 array accesses)

Quick-find: Java implementation

```
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;
}
```

Q. What's wrong with this instead?

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

A. id[p] may change part-way through the loop!

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.

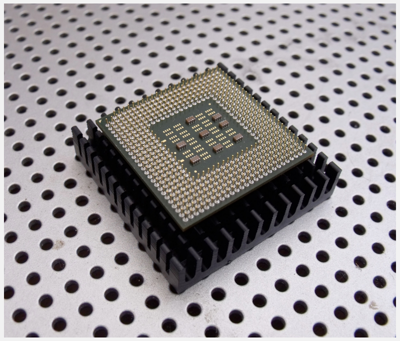
quadratic

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!

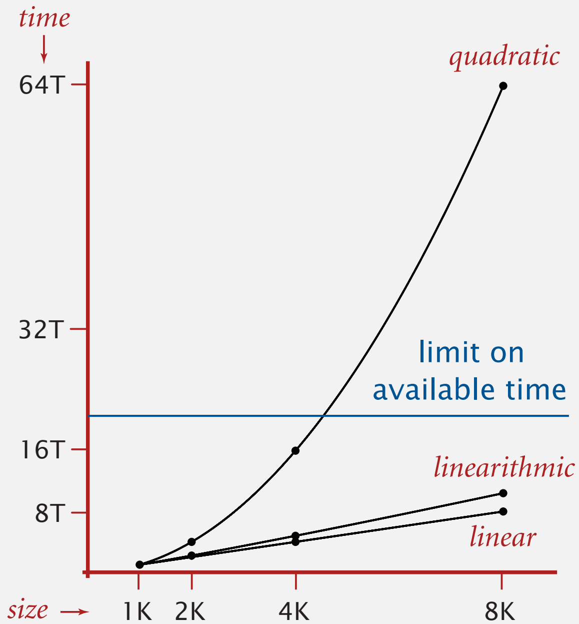


Ex. Huge problem for quick-find.

- 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

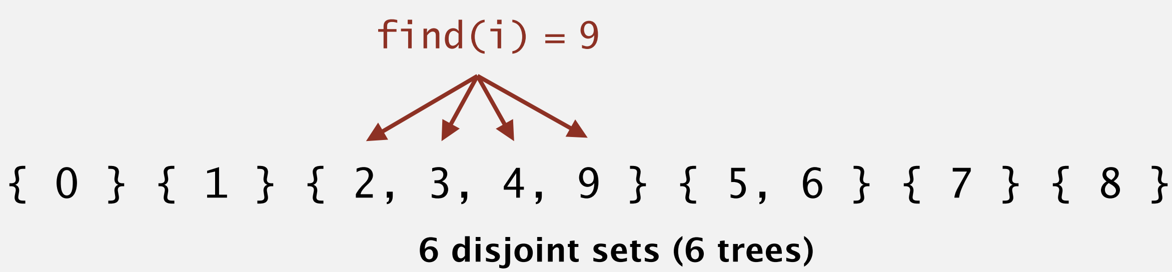
- *dynamic-connectivity problem*
- *quick find*
- *quick union*
- *improvements*
- *applications*

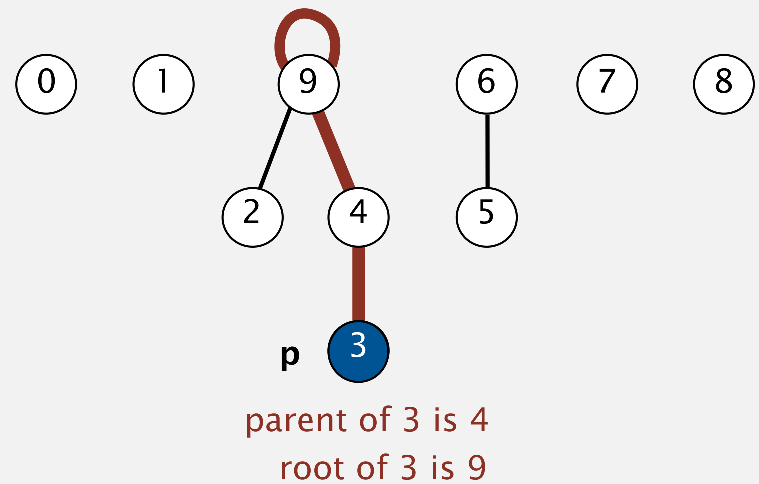
Quick-union [lazy approach]

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





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

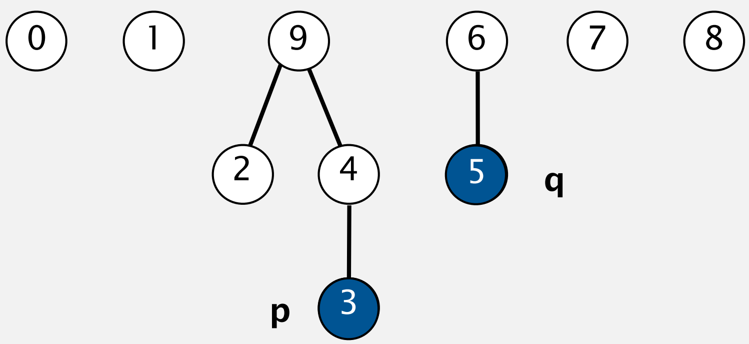
A. Return root of tree containing `p`.

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 `q`'s root.

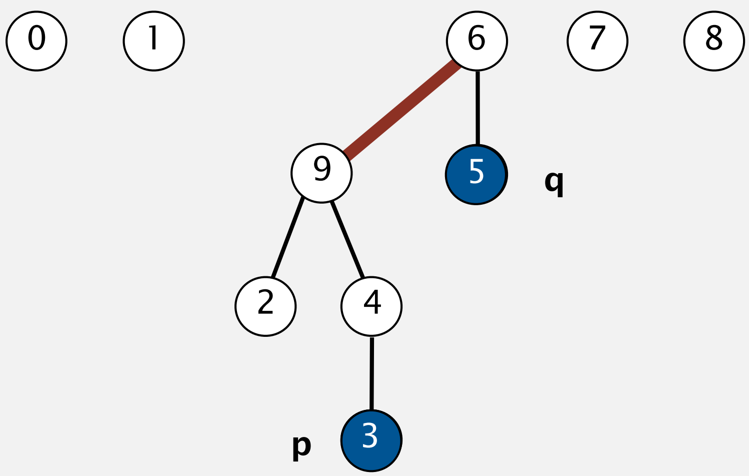
Quick-union [lazy approach]

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

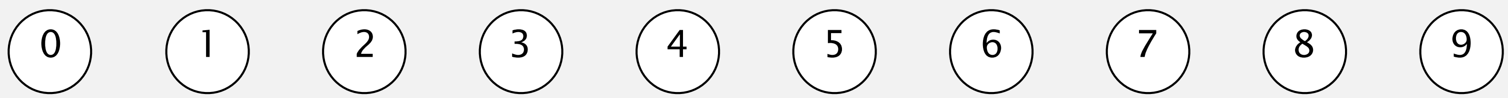
only one value changes



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

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

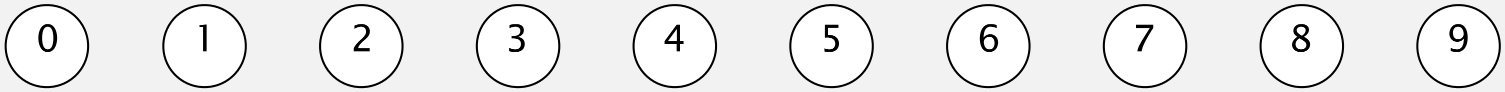
Quick-union demo



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

Quick-union demo

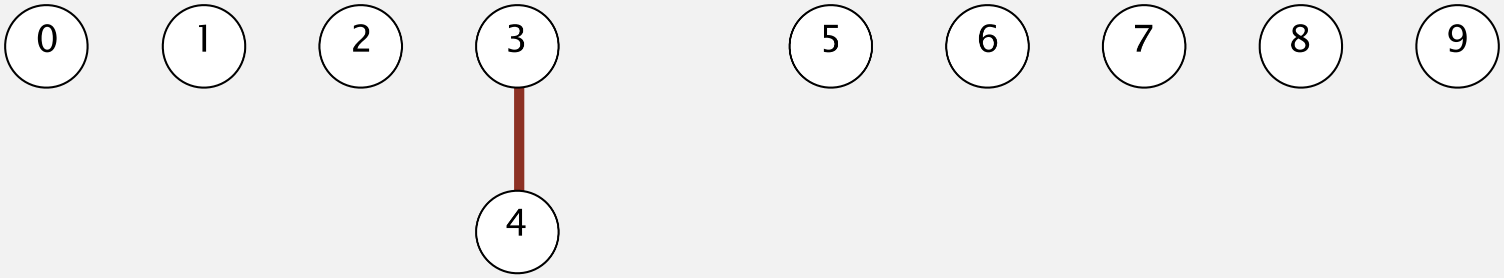
union(4, 3)



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

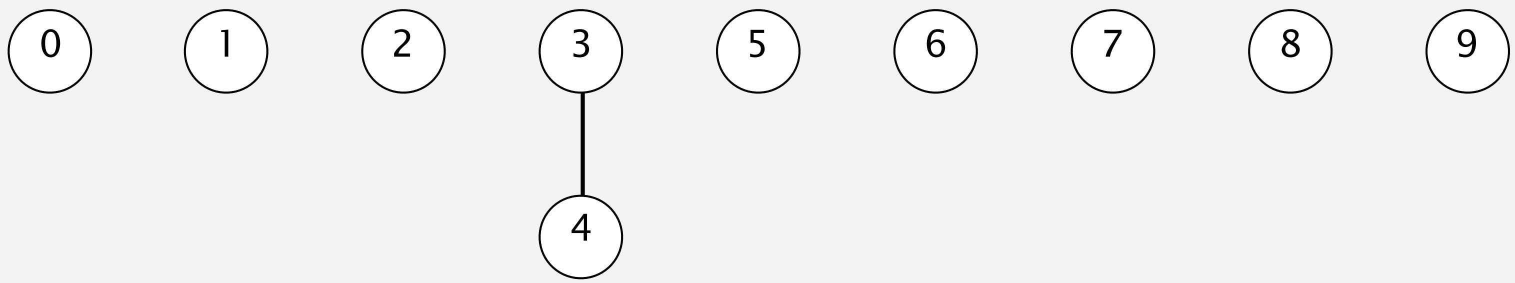
Quick-union demo

union(4, 3)



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

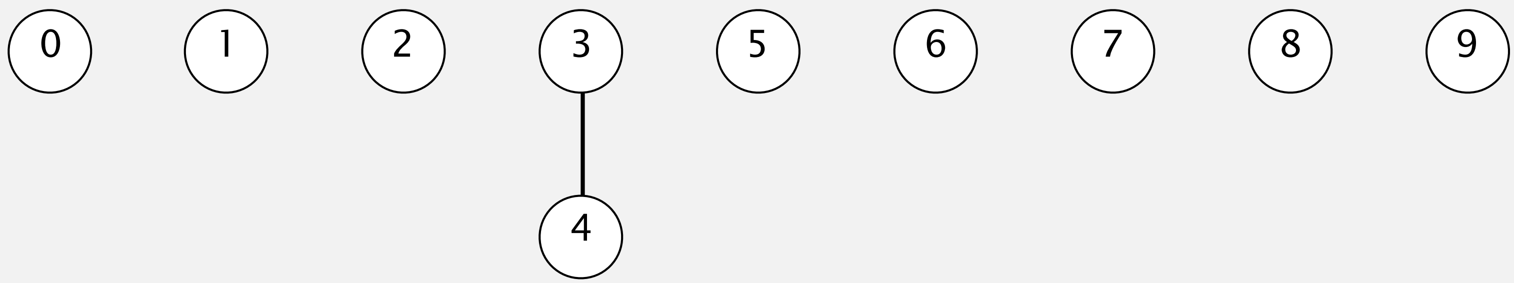
Quick-union demo



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

Quick-union demo

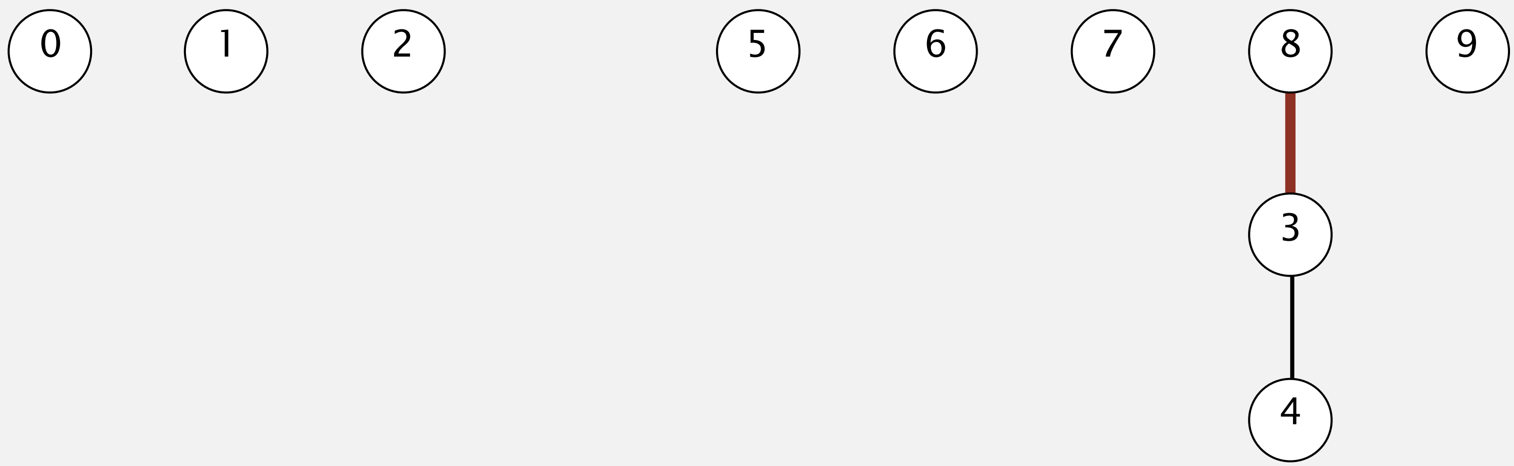
union(3, 8)



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

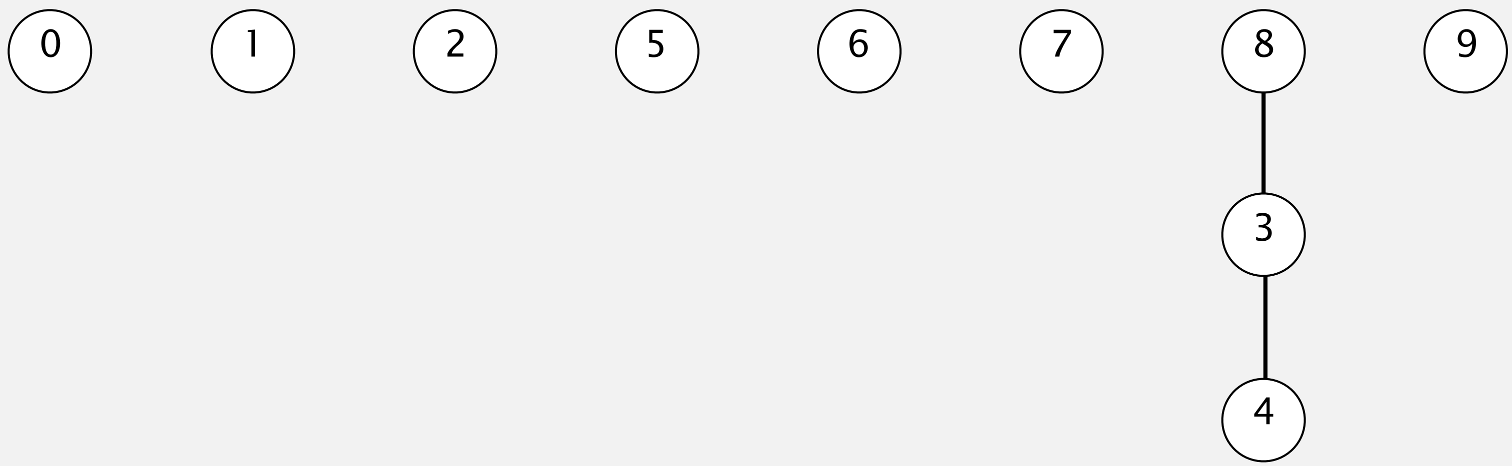
Quick-union demo

union(3, 8)



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

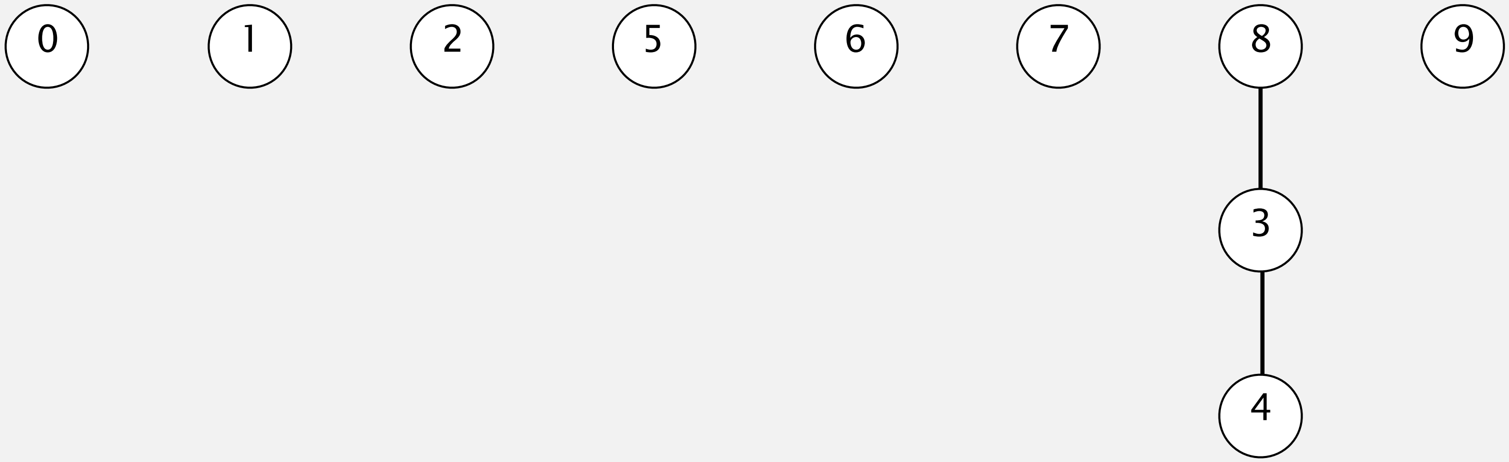
Quick-union demo



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

Quick-union demo

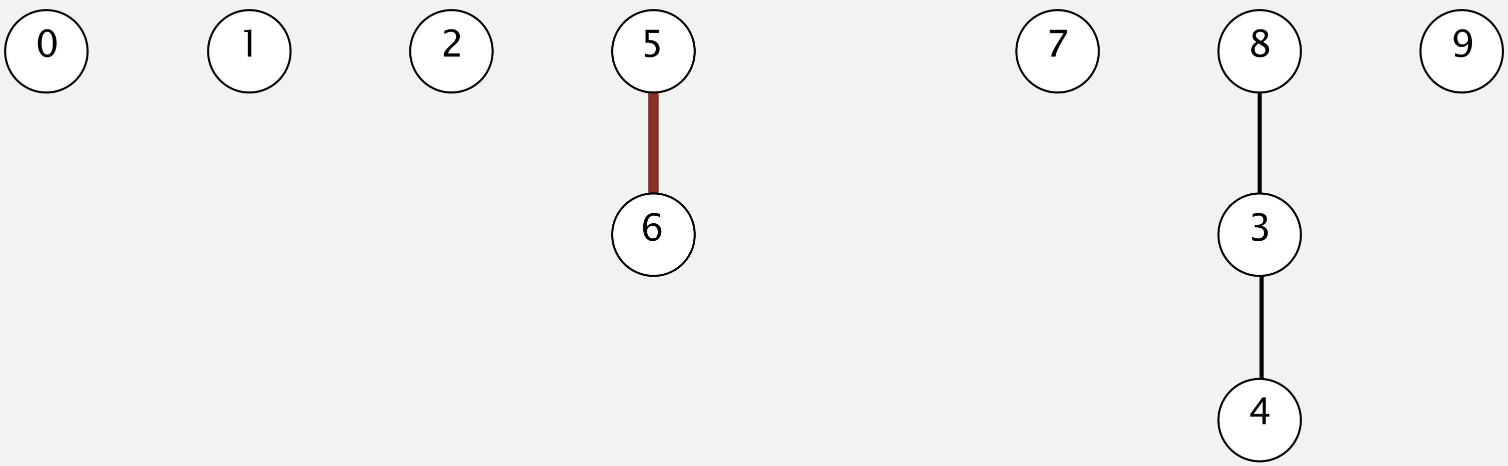
union(6, 5)



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

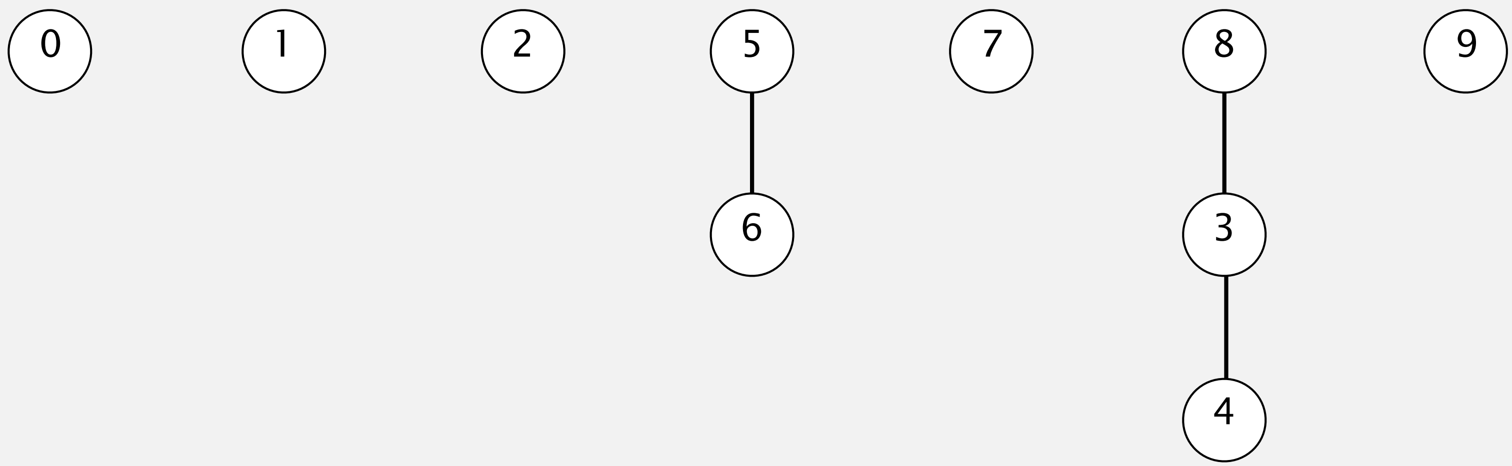
Quick-union demo

union(6, 5)



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

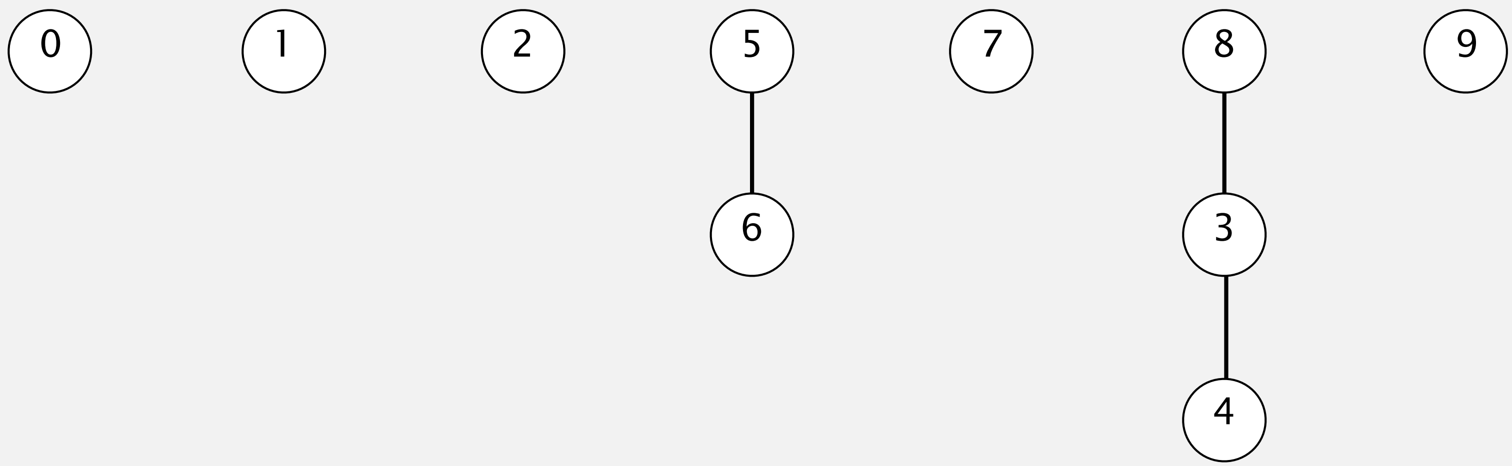
Quick-union demo



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

Quick-union demo

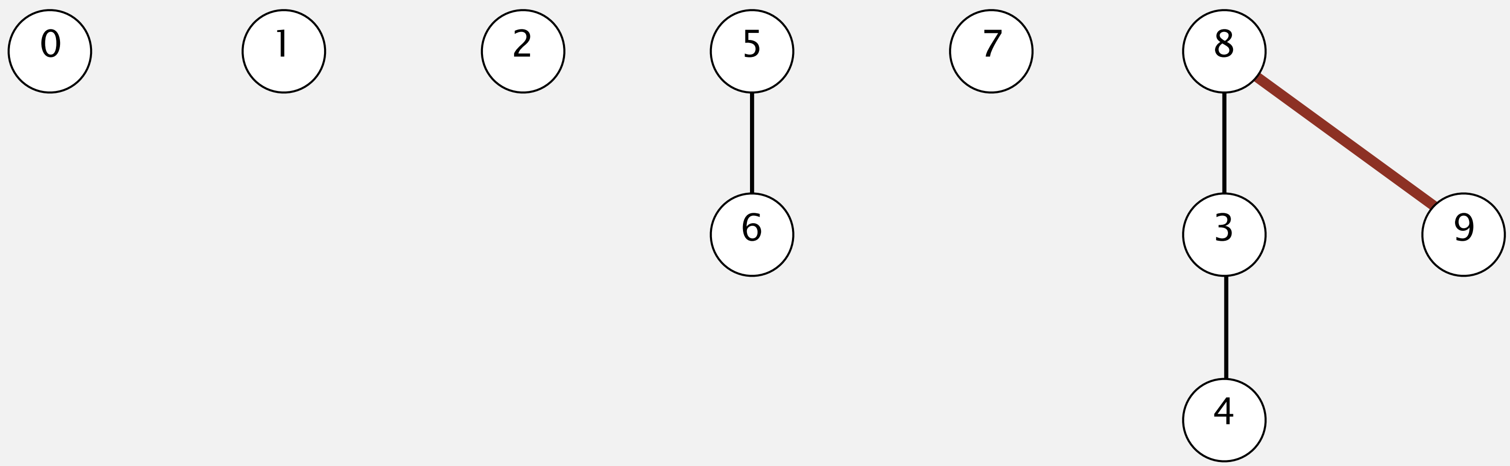
union(9, 4)



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

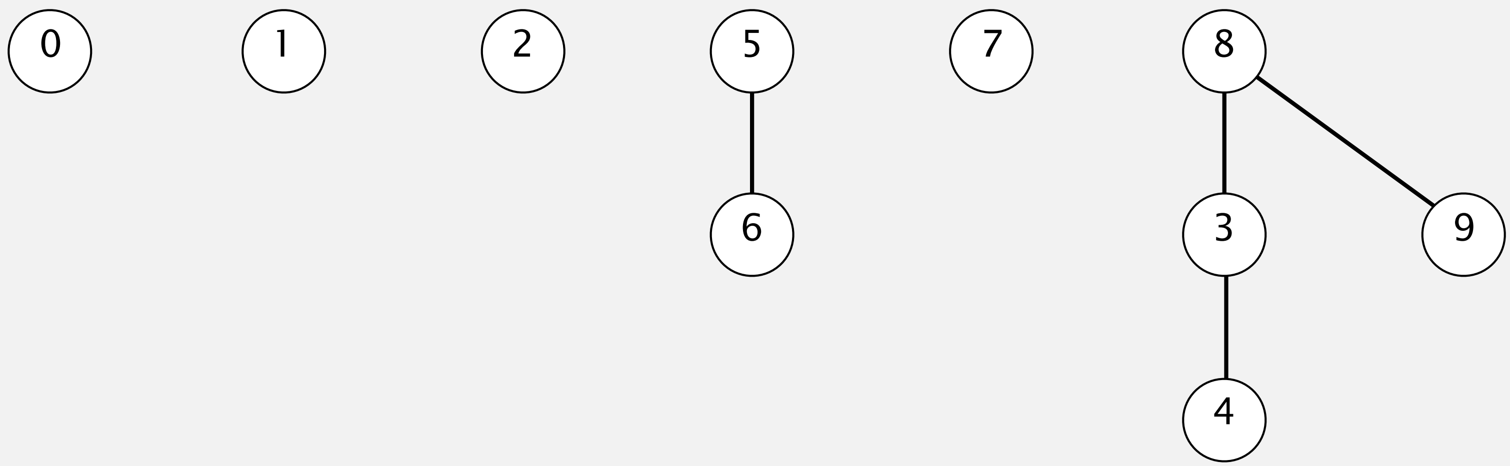
Quick-union demo

union(9, 4)



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

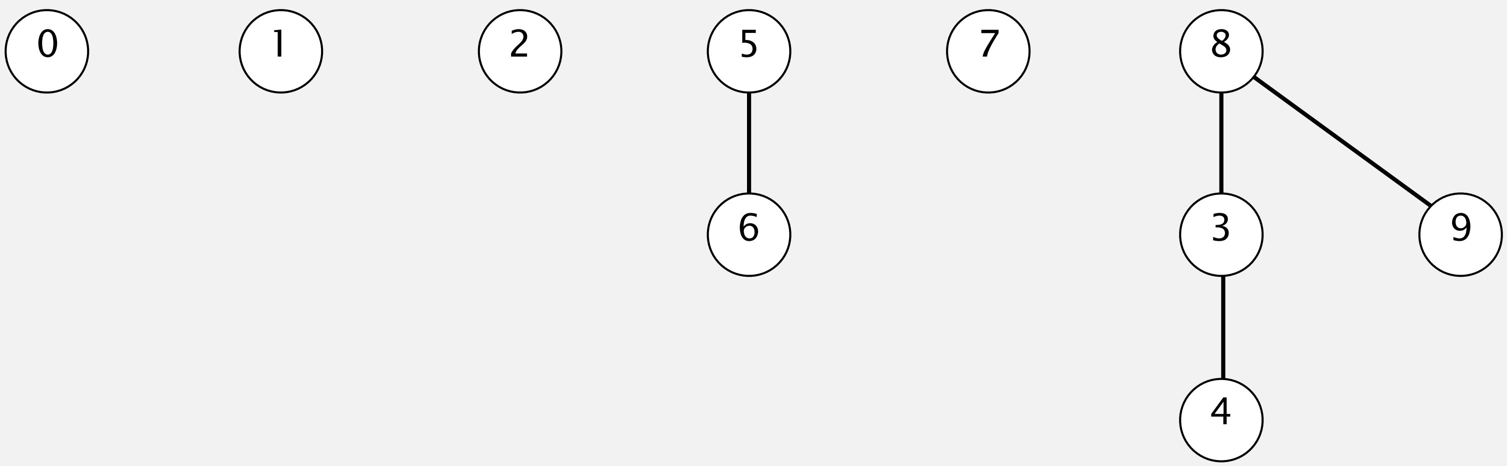
Quick-union demo



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

Quick-union demo

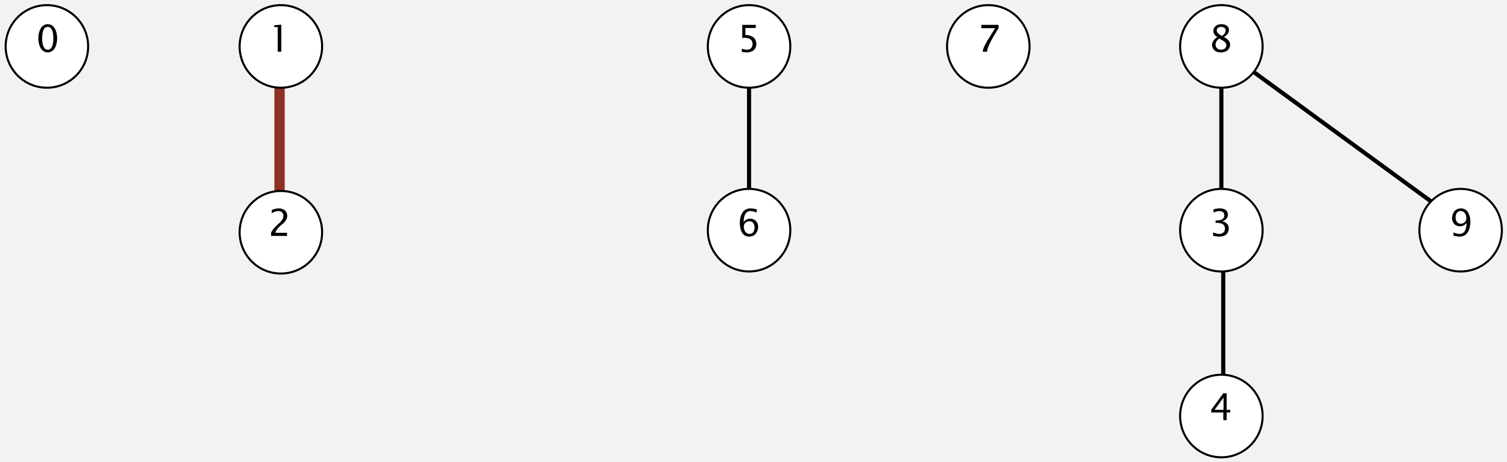
union(2, 1)



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

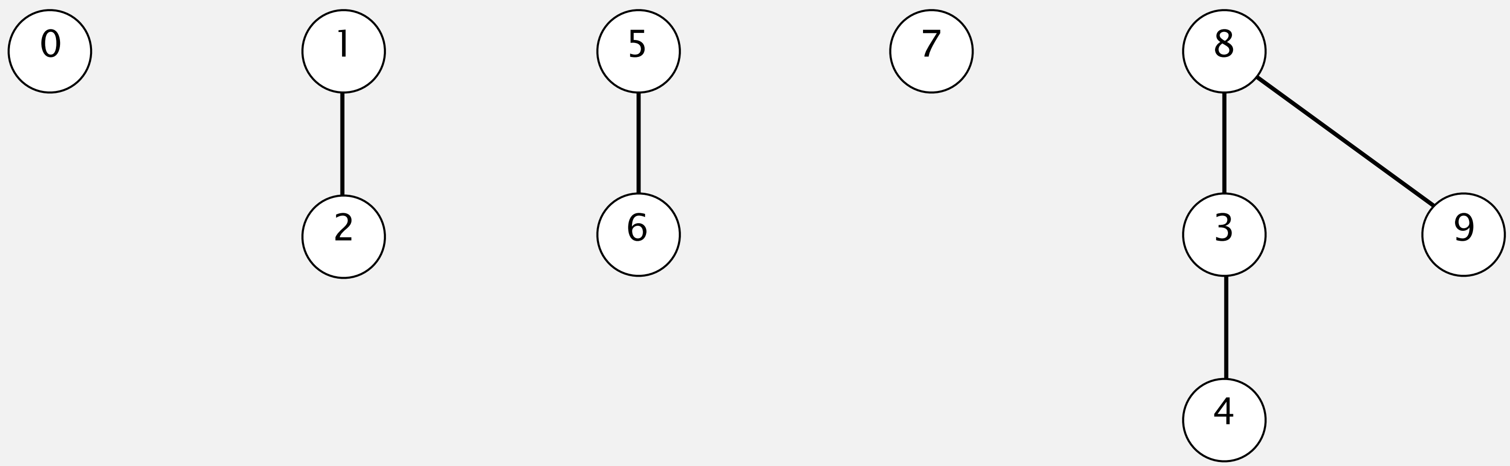
Quick-union demo

union(2, 1)



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

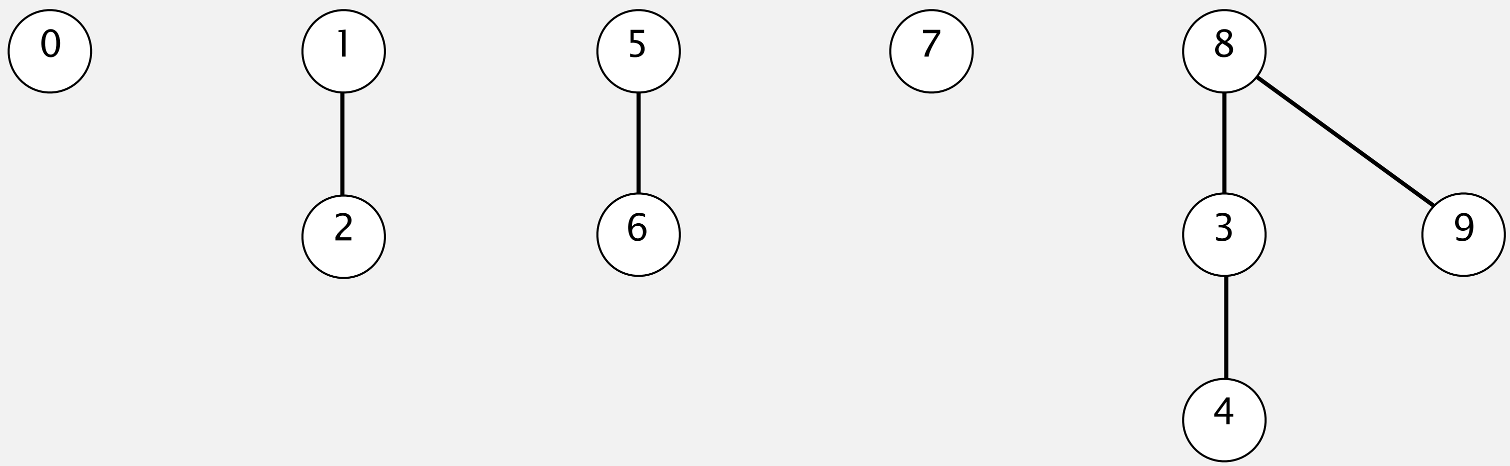
Quick-union demo



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

Quick-union demo

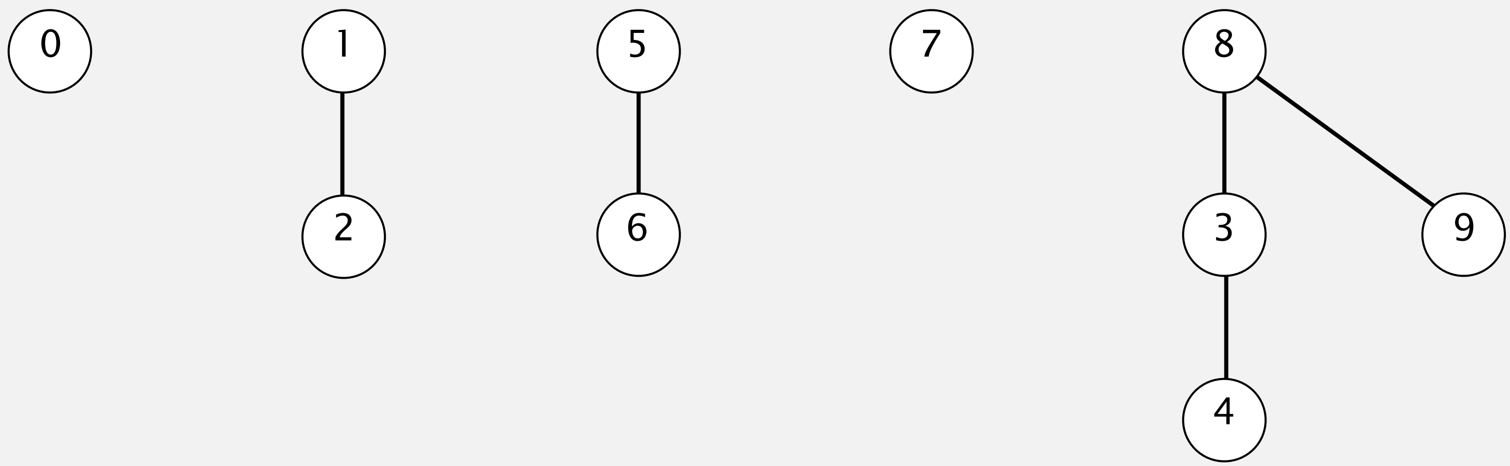
find(8) == find(9) ✔



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

Quick-union demo

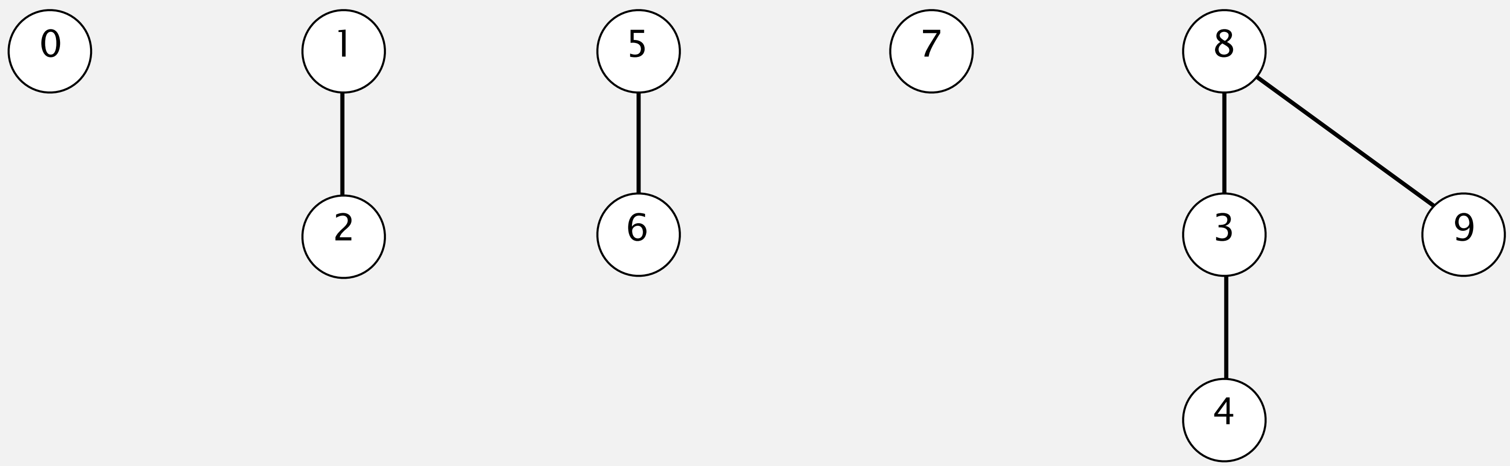
find(5) == find(4) ✗



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

Quick-union demo

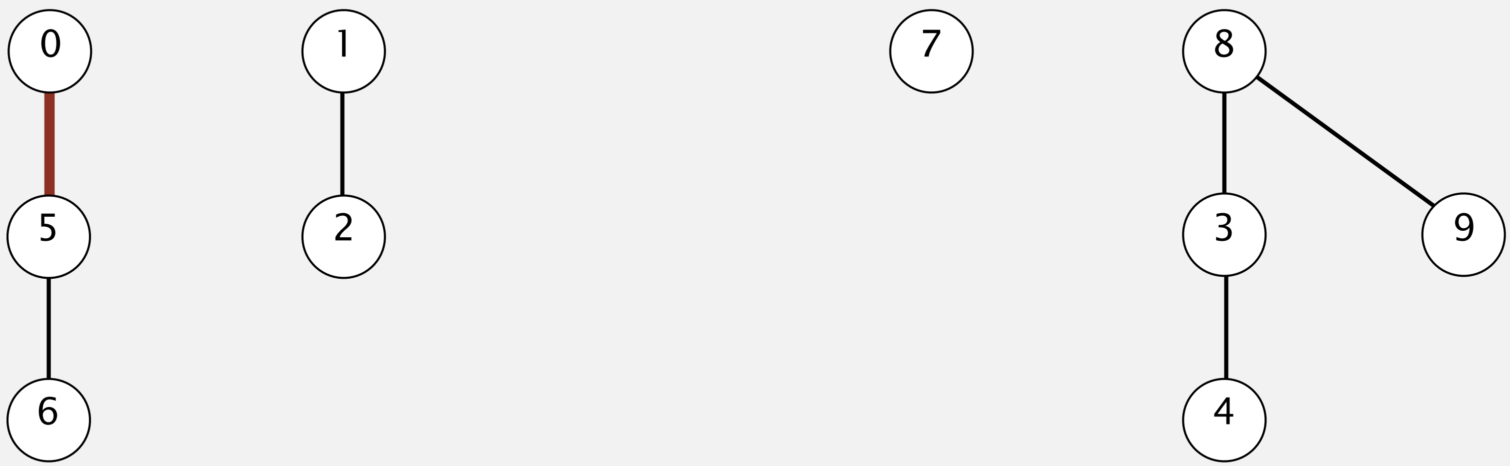
union(5, 0)



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

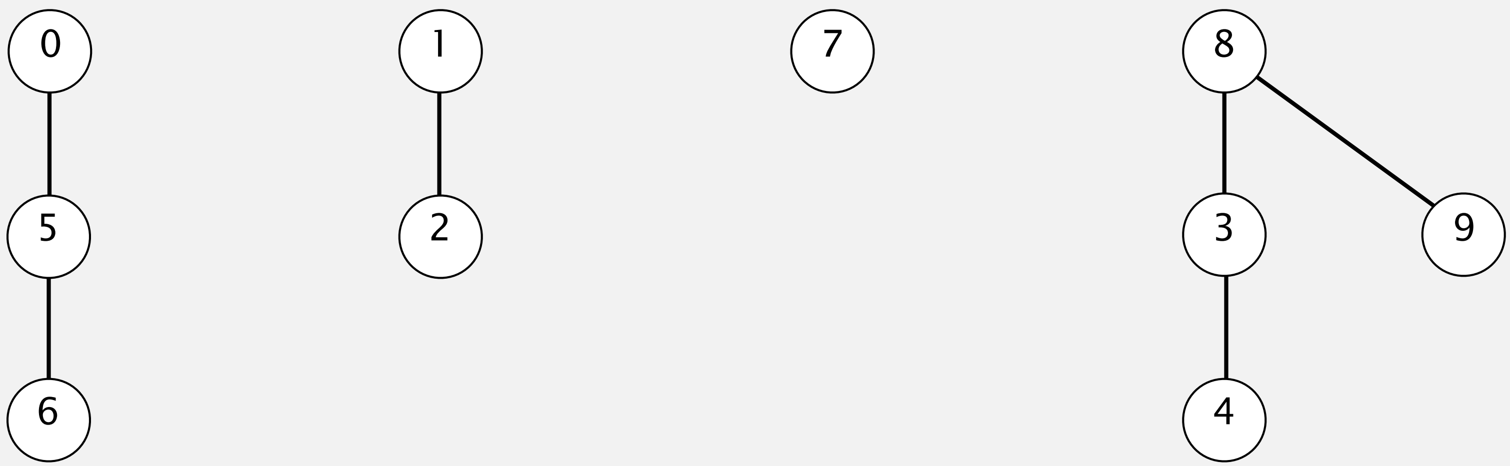
Quick-union demo

union(5, 0)



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

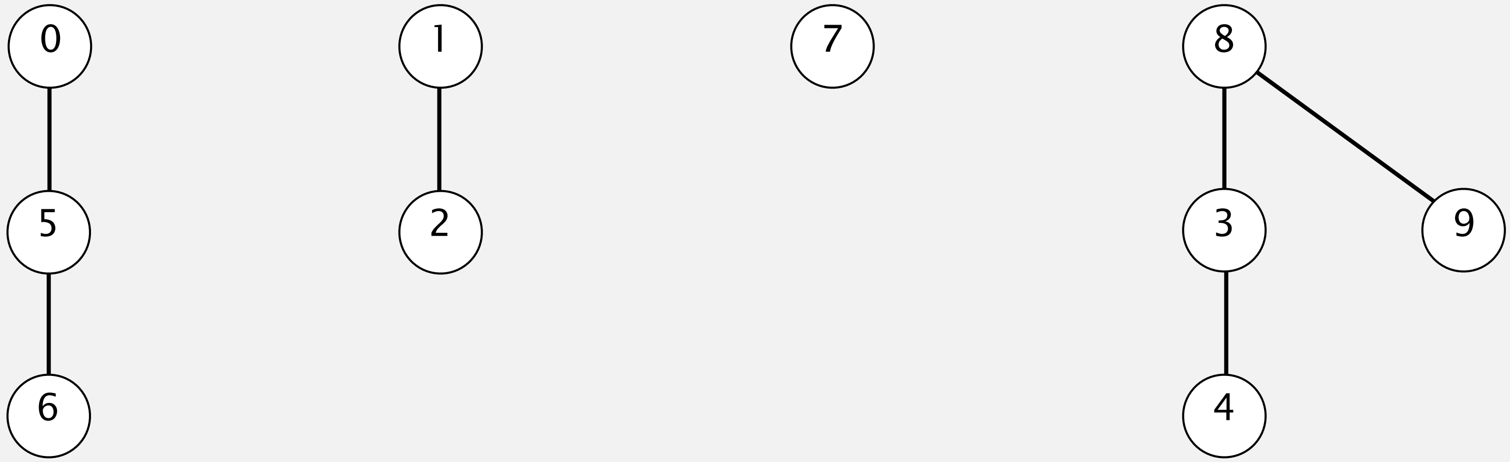
Quick-union demo



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

Quick-union demo

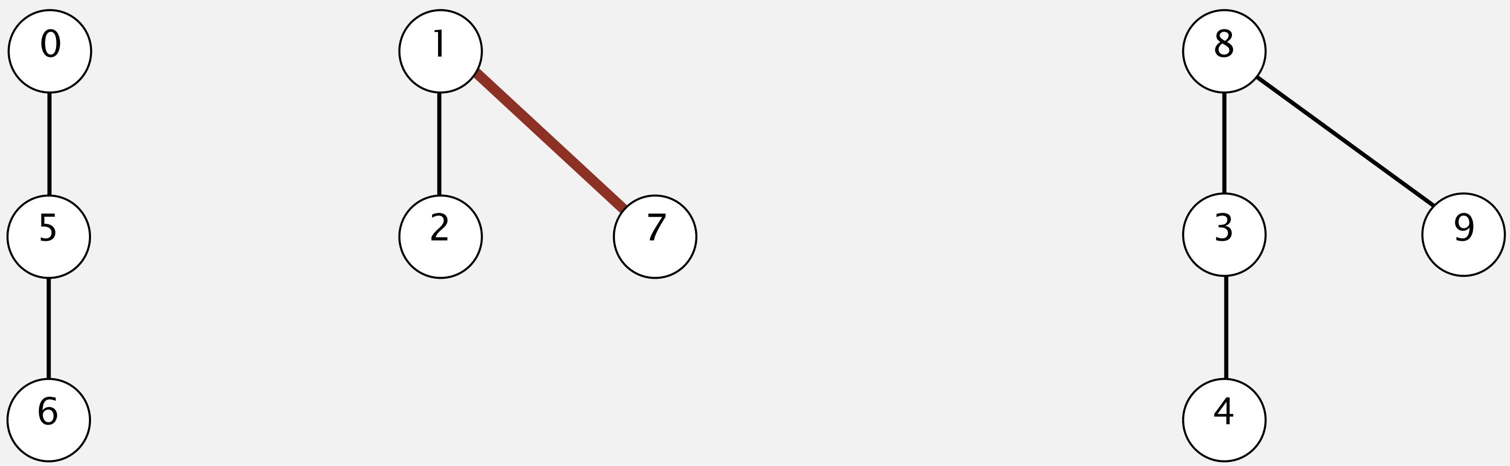
union(7, 2)



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

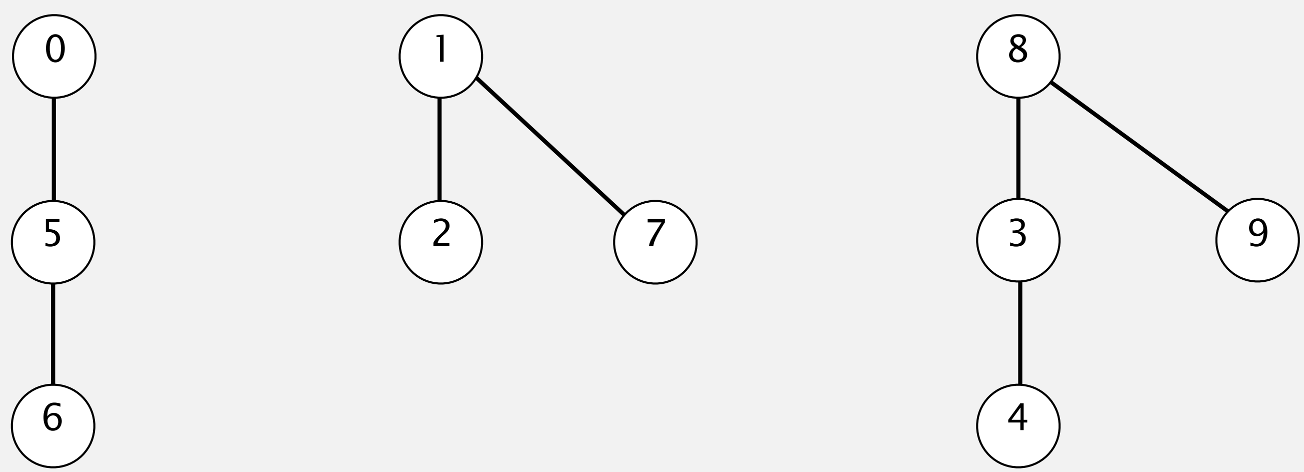
Quick-union demo

union(7, 2)



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

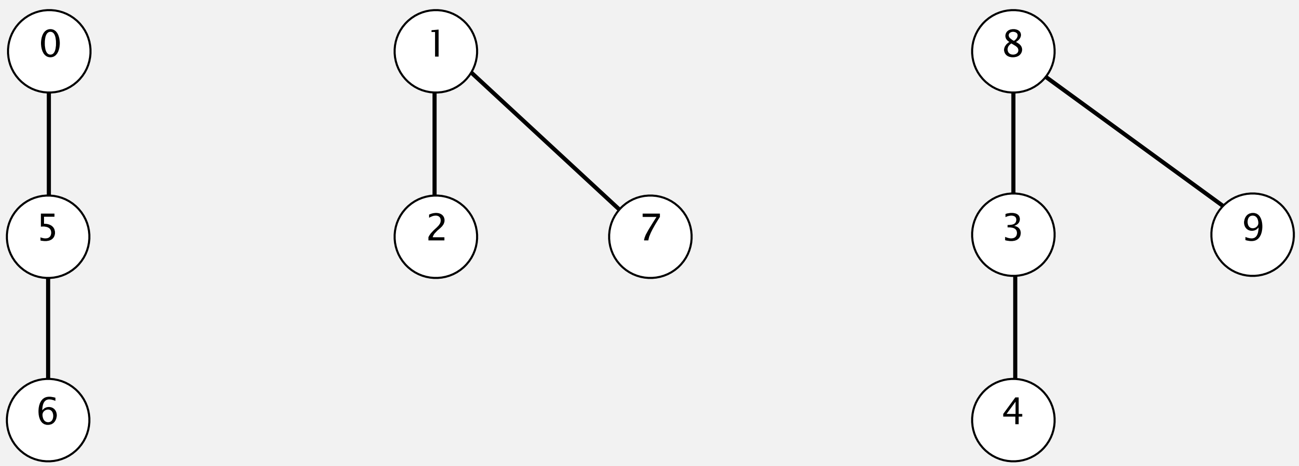
Quick-union demo



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

Quick-union demo

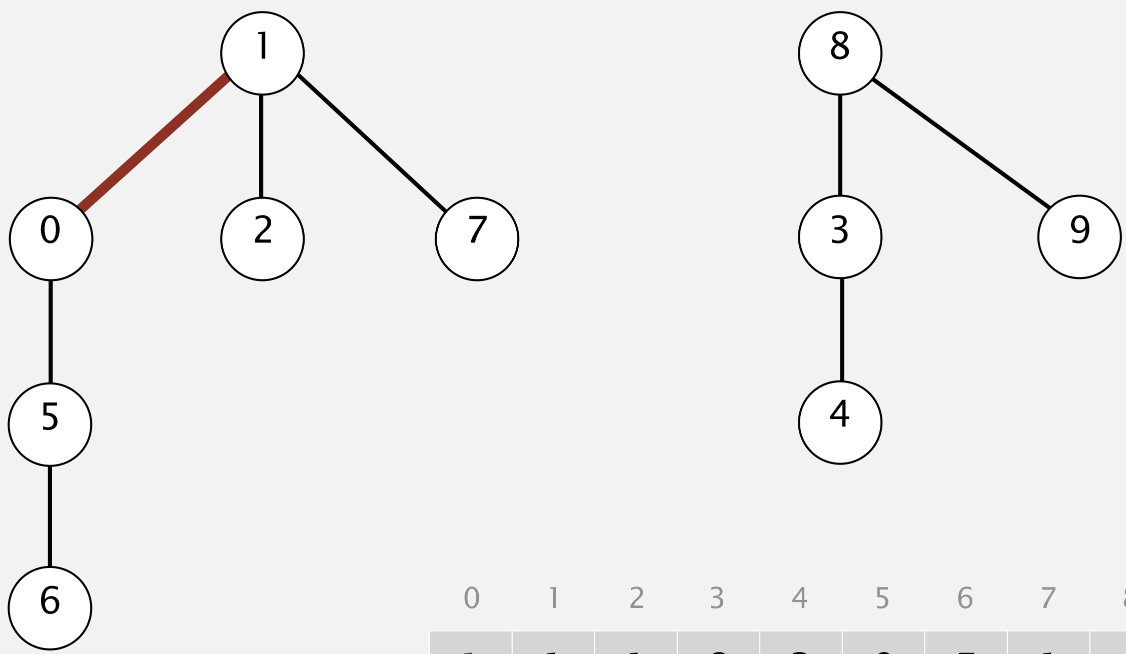
union(6, 1)



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

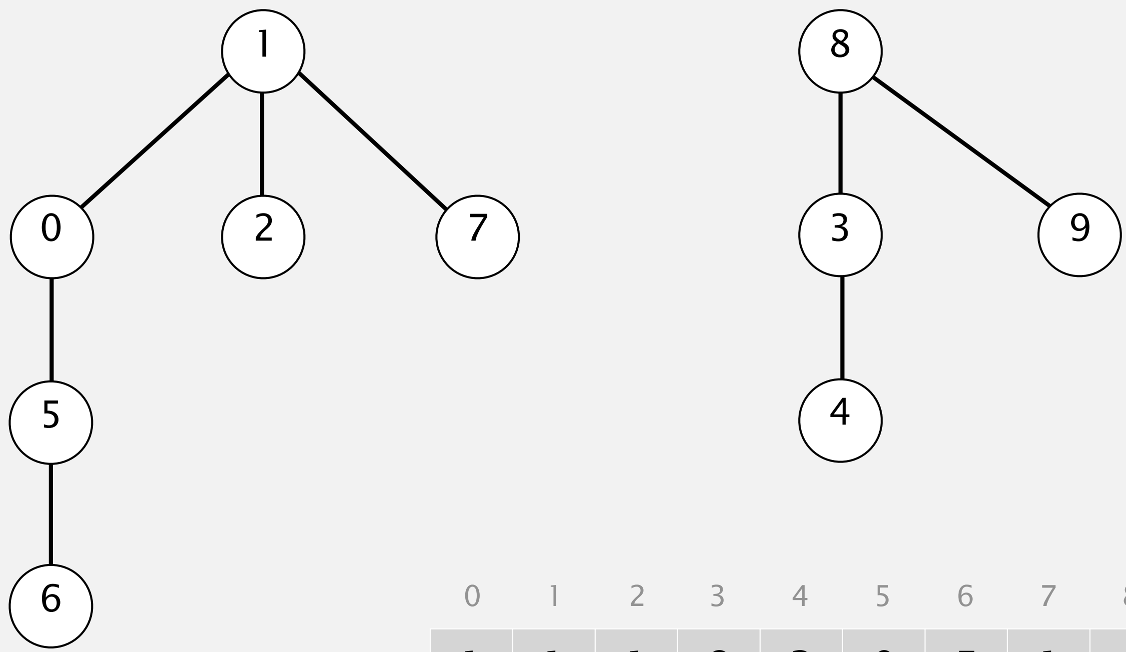
Quick-union demo

union(6, 1)



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

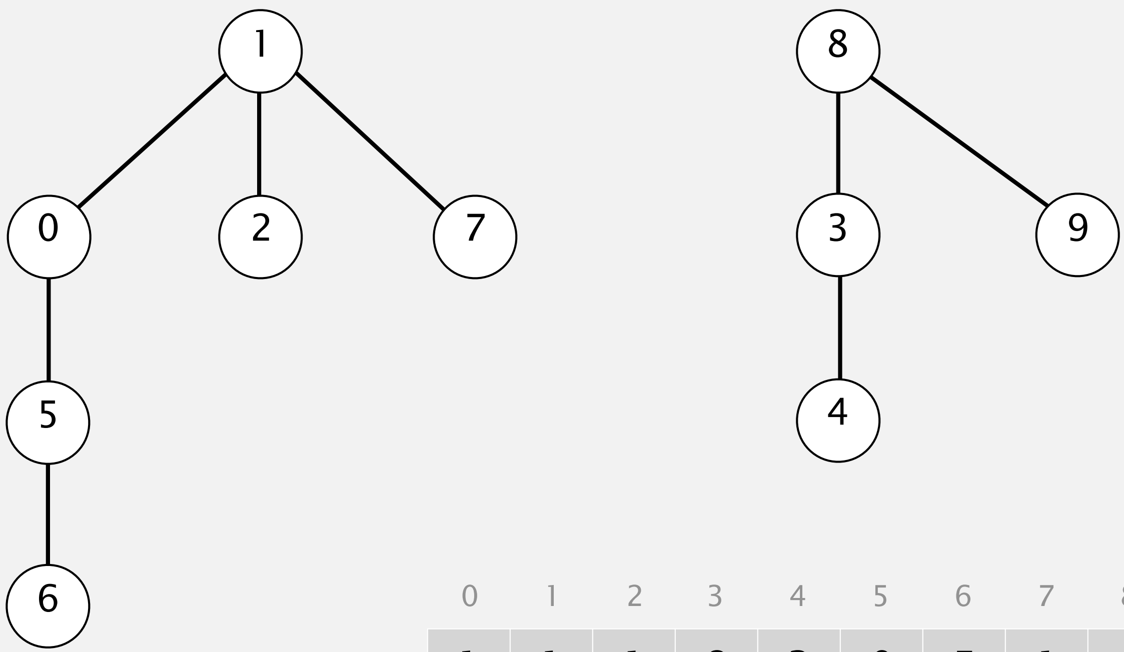
Quick-union demo



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

Quick-union demo

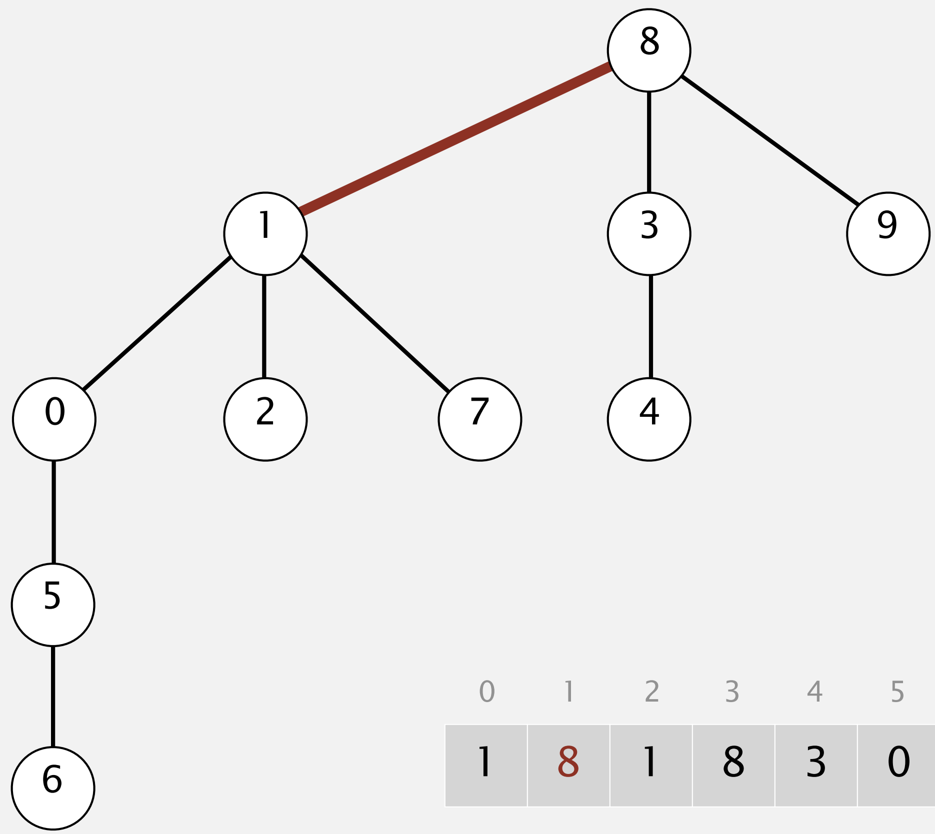
union(7, 3)



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

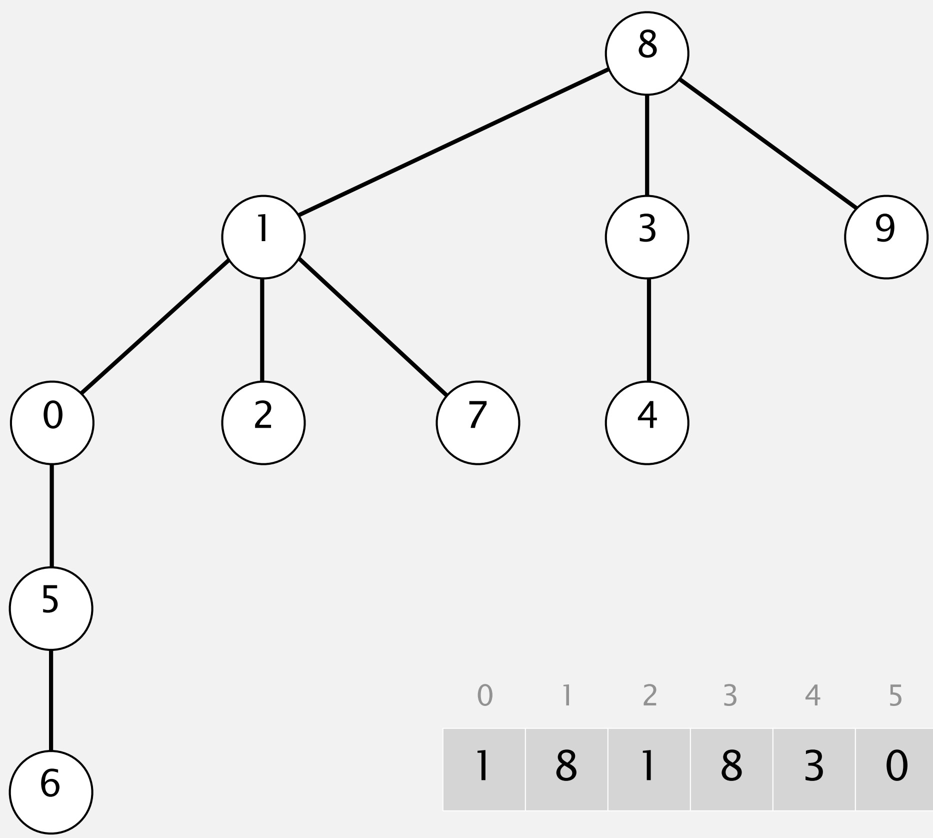
Quick-union demo

union(7, 3)



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

Quick-union demo



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

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

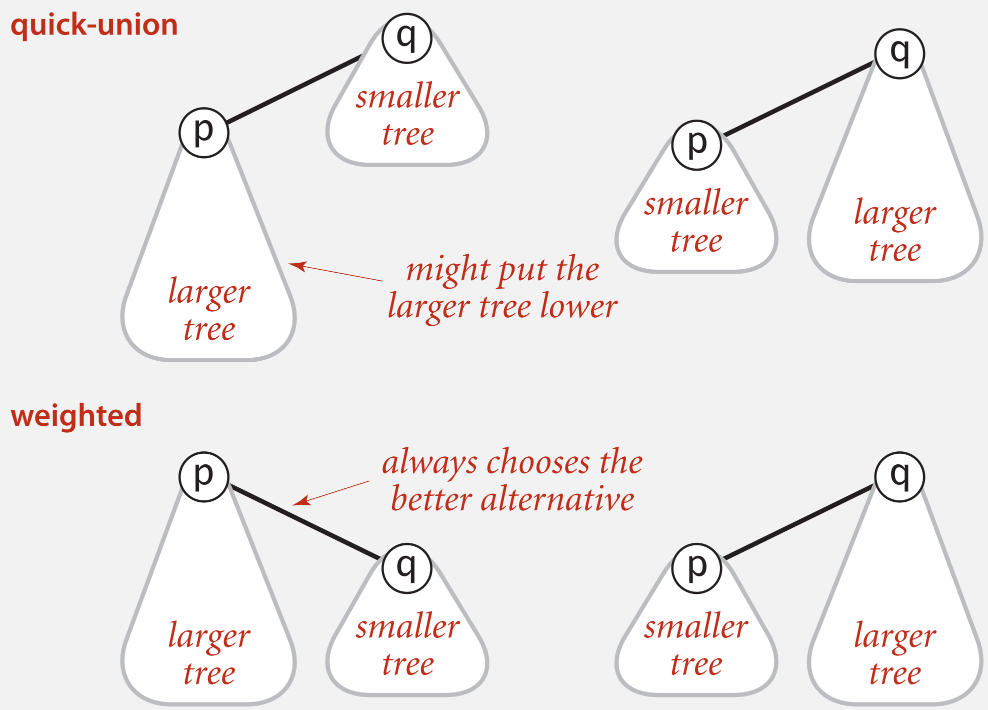
- *dynamic-connectivity problem*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Improvement 1: 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.

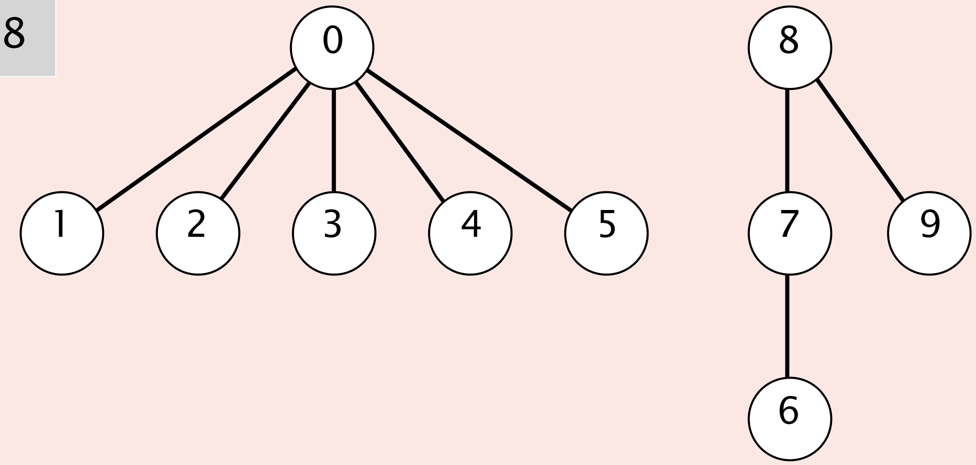
reasonable alternative: union by height/rank



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]`

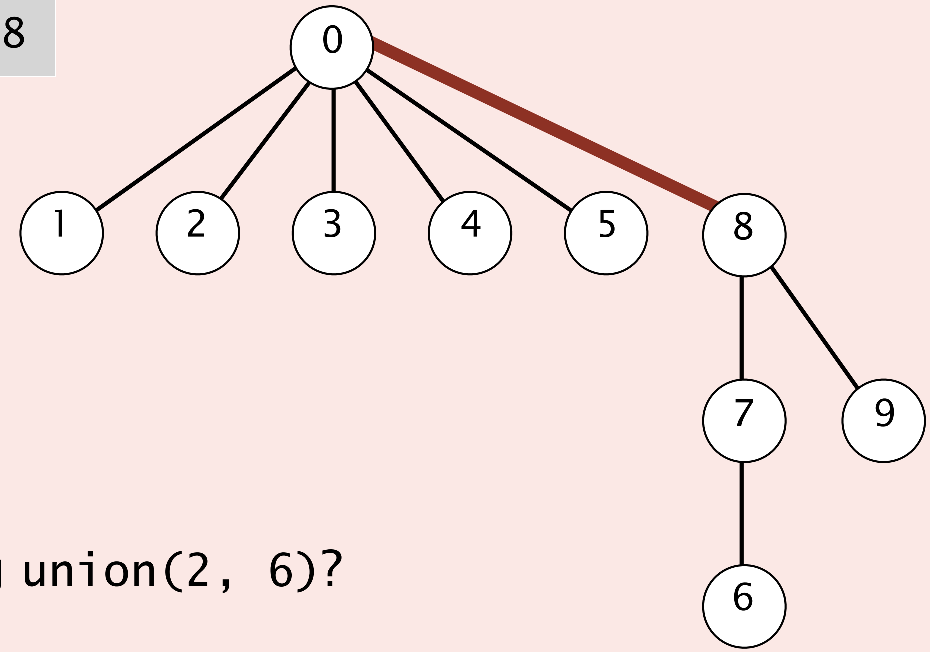
C. `parent[6]`

D. `parent[8]`

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]`

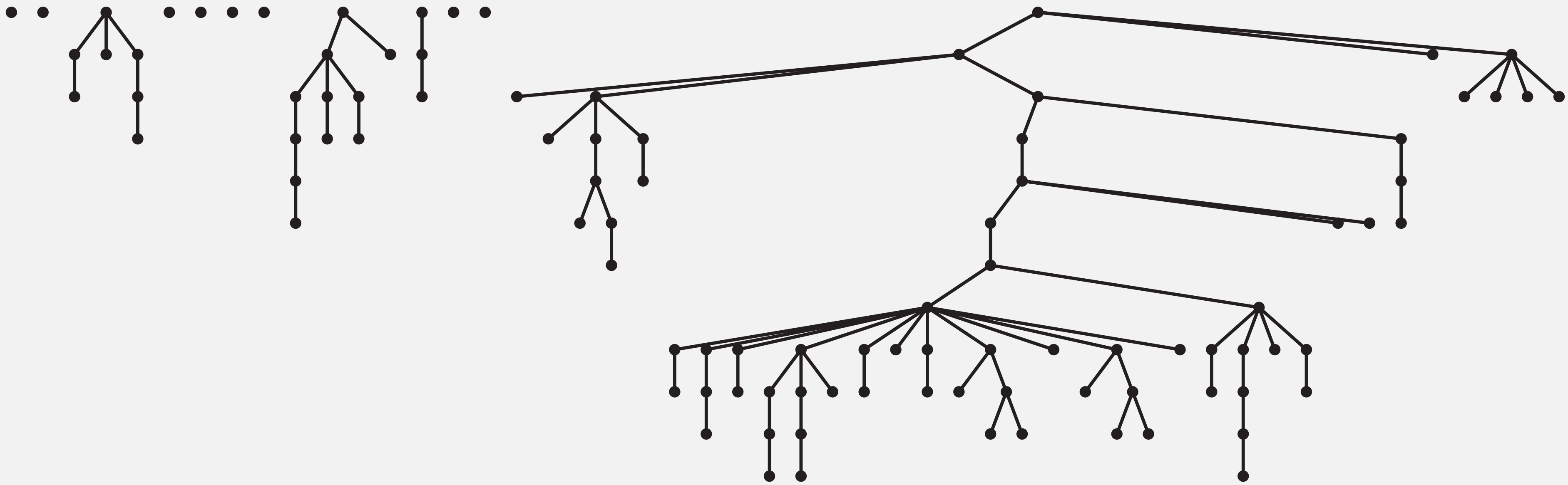
B. `parent[2]`

C. `parent[6]`

D. `parent[8]`

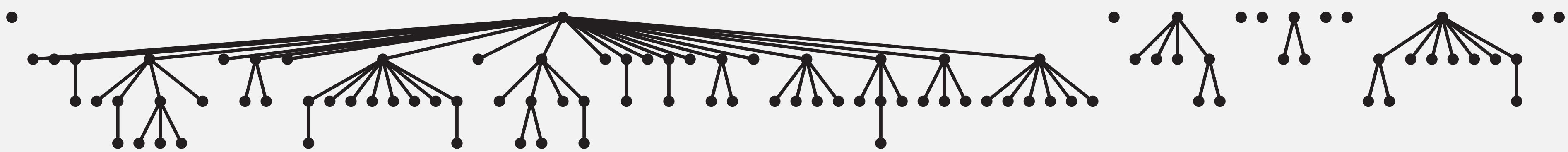
Quick-union vs. weighted quick-union

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.

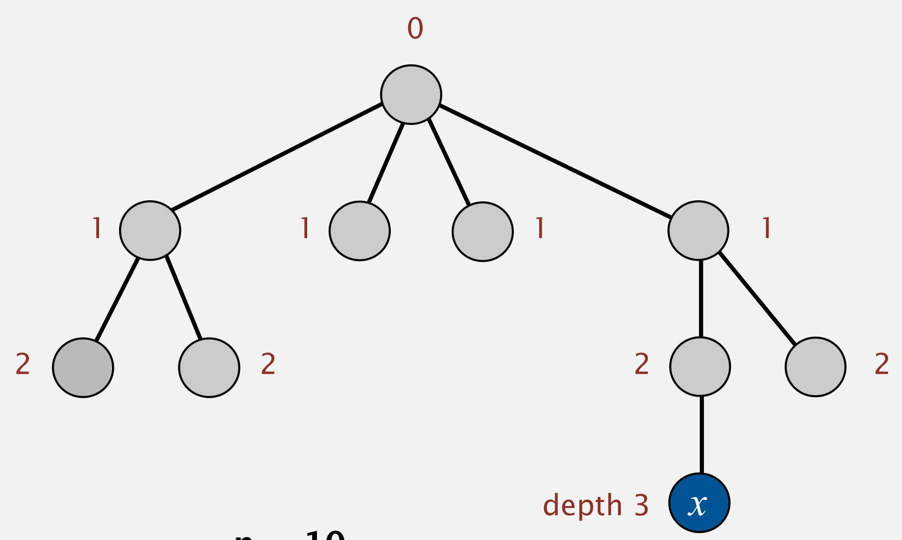
```
int i = find(p);
int j = find(q);
if (i == j) return;
if  (size[i] < size[j]) { parent[i] = j; size[j] += size[i]; }
else                     { parent[j] = i; size[i] += size[j]; }
```

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 $\lg n$. ← in computer science, lg means base-2 logarithm



n = 10
depth(x) = 3 ≤ lg n

Weighted quick-union analysis

Running time.

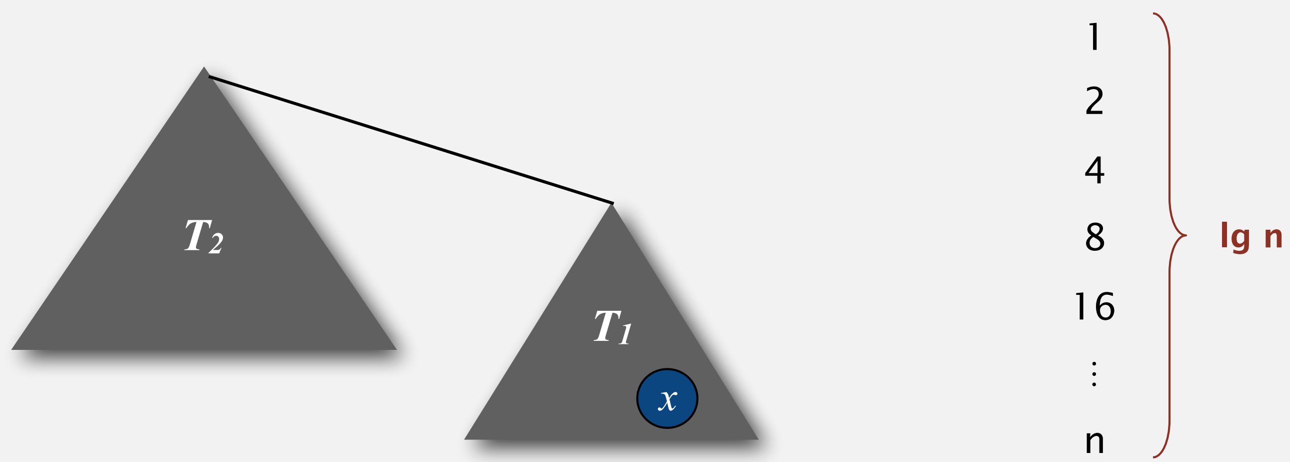
- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

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 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	$\log n$ †	$\log n$

← log mean logarithm, for some constant base

† 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	$n + m \log^* n$

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.

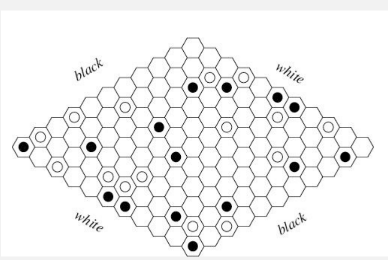
http://algs4.cs.princeton.edu

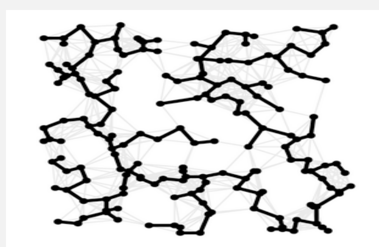
1.5 Union–Find

- *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.
- Hindley–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.



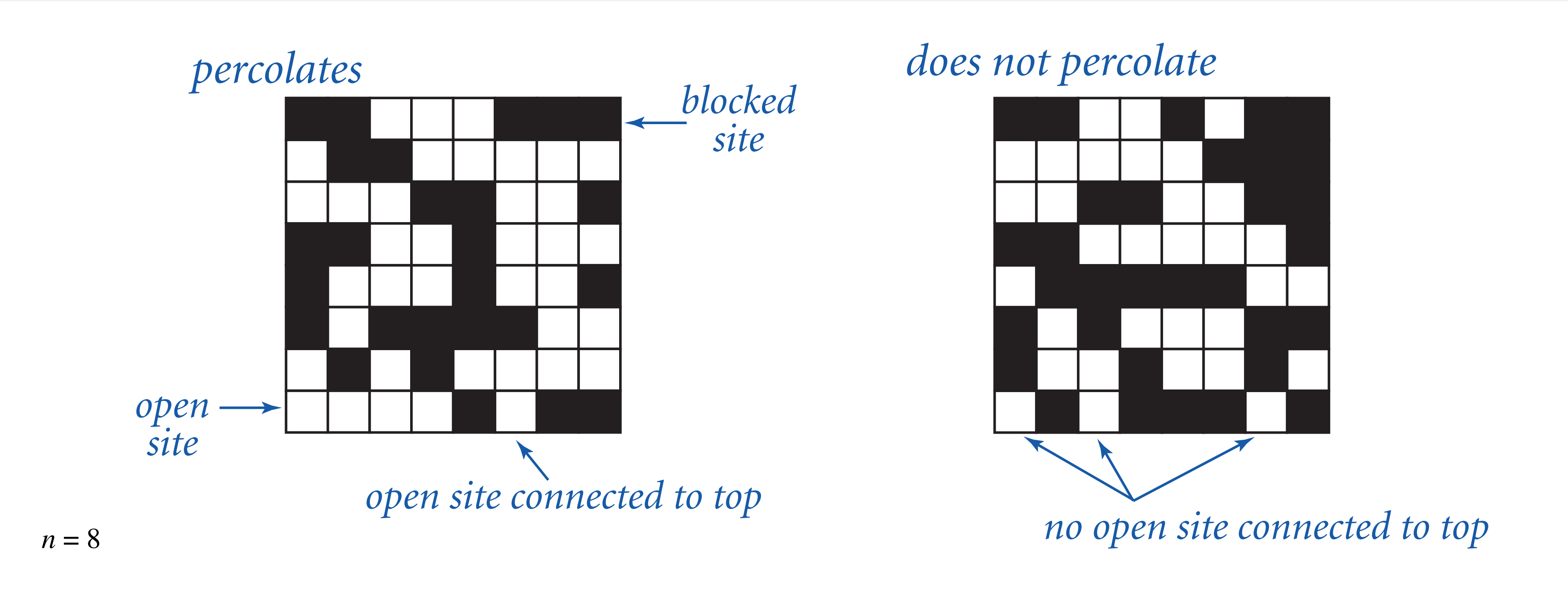


Percolation

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

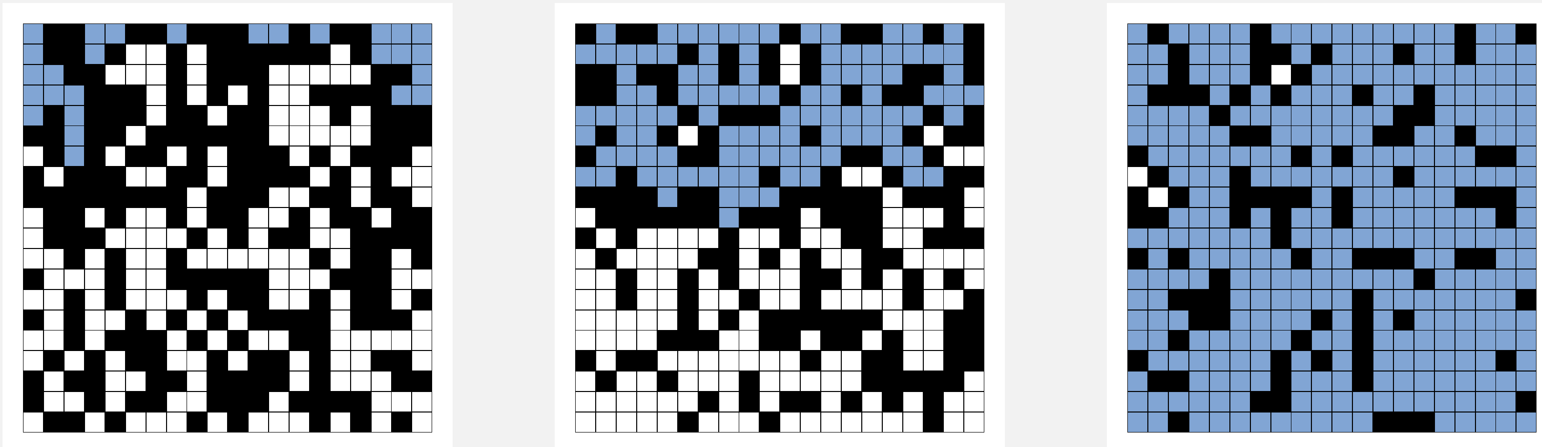
An abstract model for many physical systems:

- 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

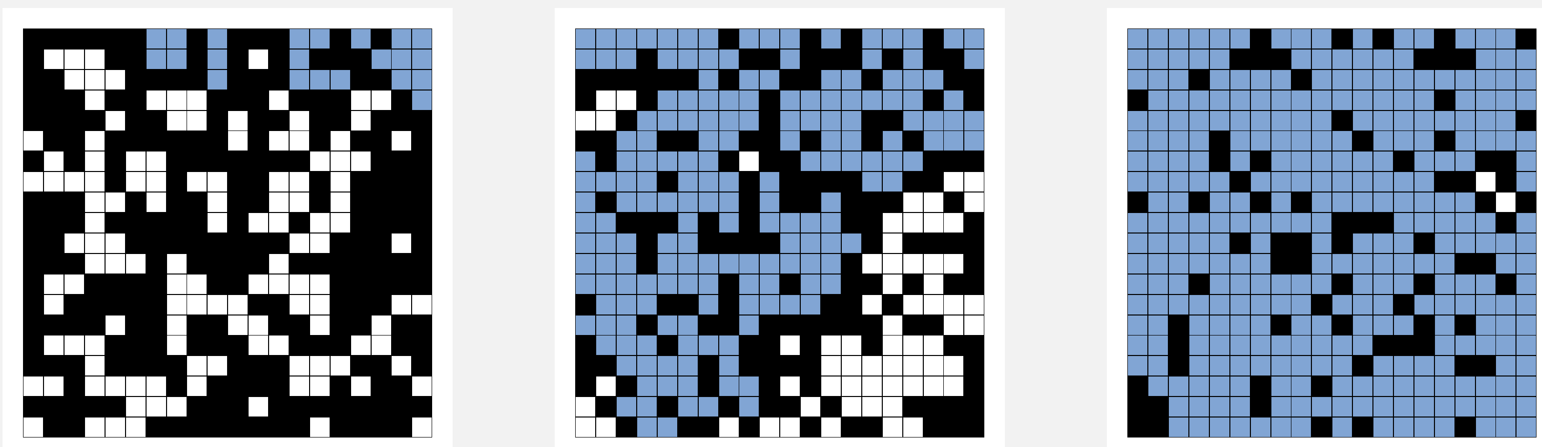
Depends on grid size n and site vacancy probability p.



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)

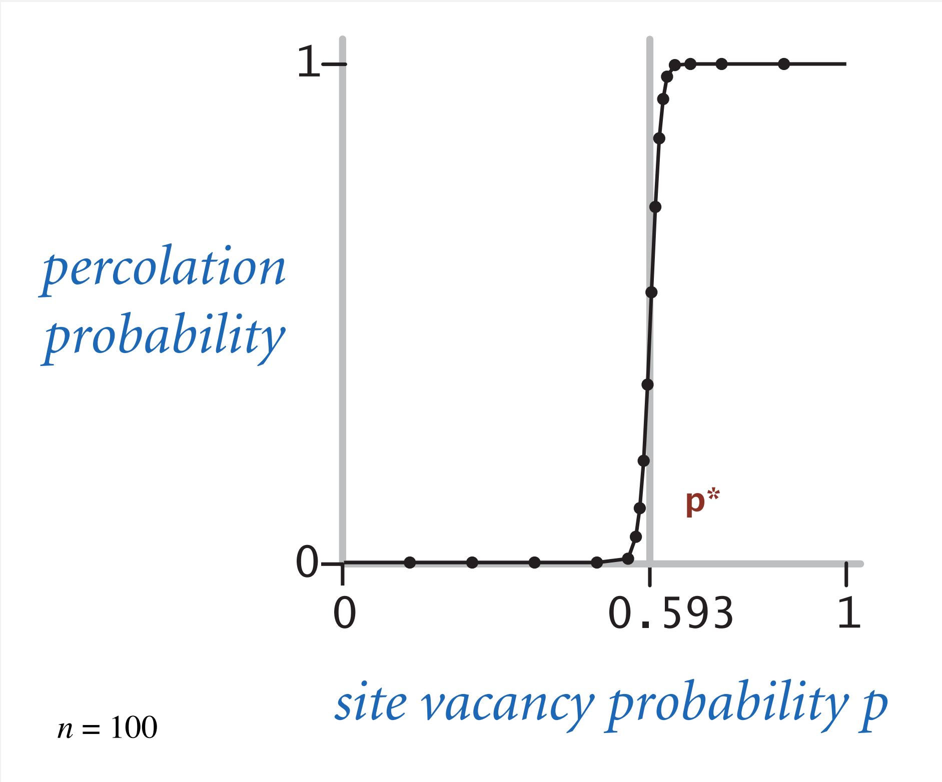
blocked site

Percolation phase transition

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

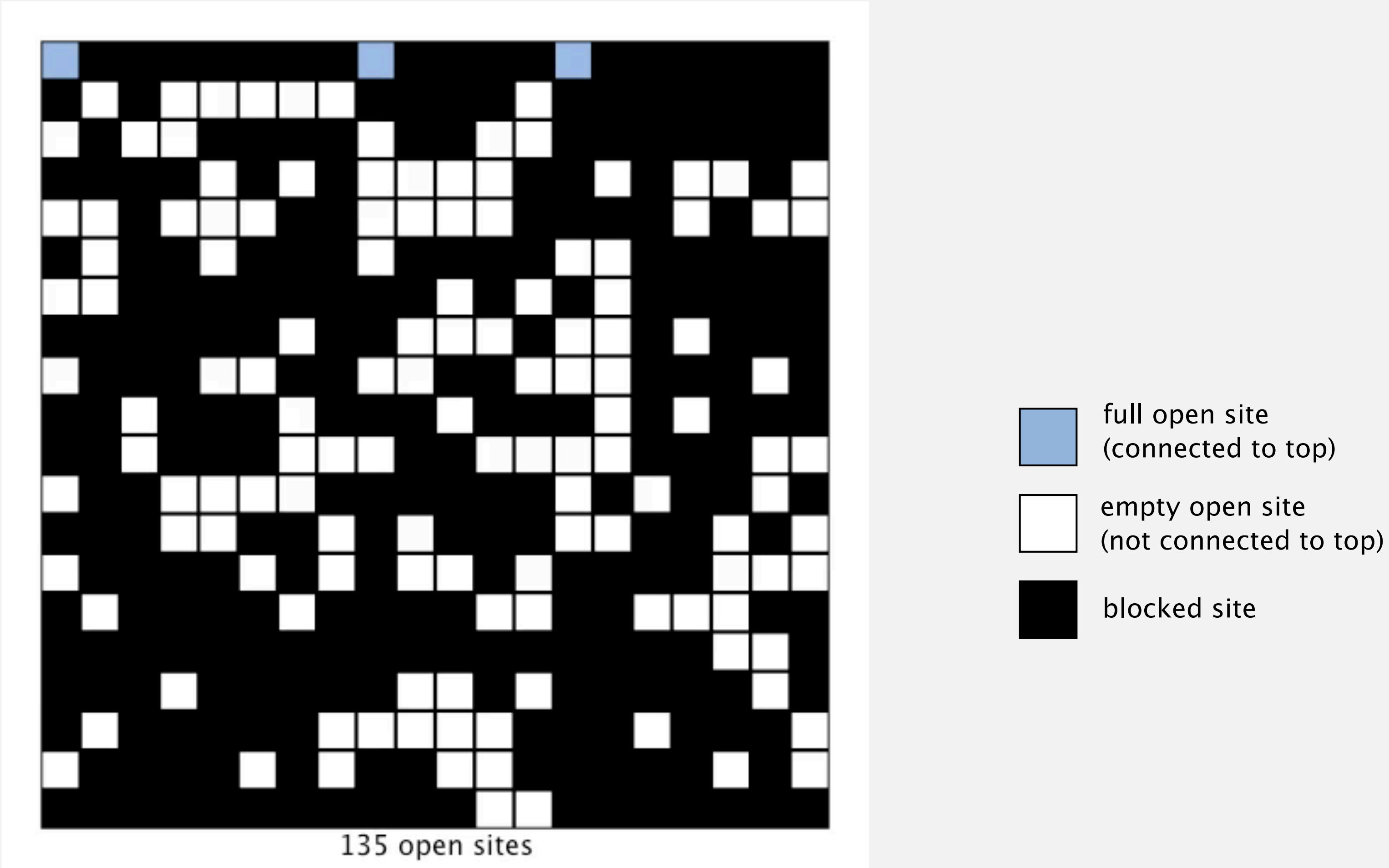
- $p > p^*$: almost certainly percolates.
- $p < p^*$: almost certainly does not percolate.

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



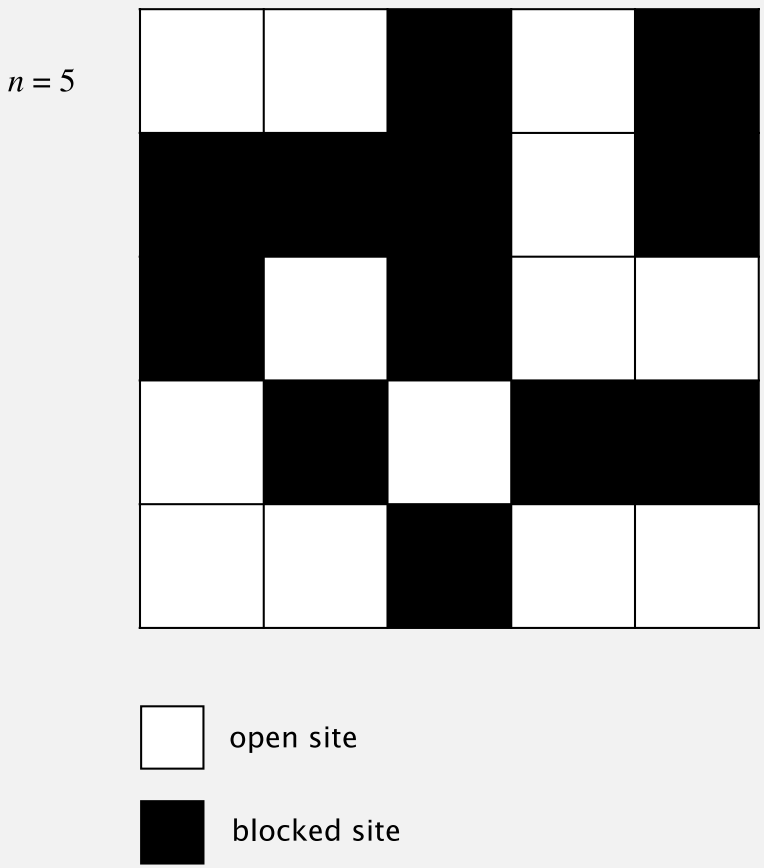
$$\hat{p} = \frac{204}{400} = 0.51$$

$n = 20$

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.

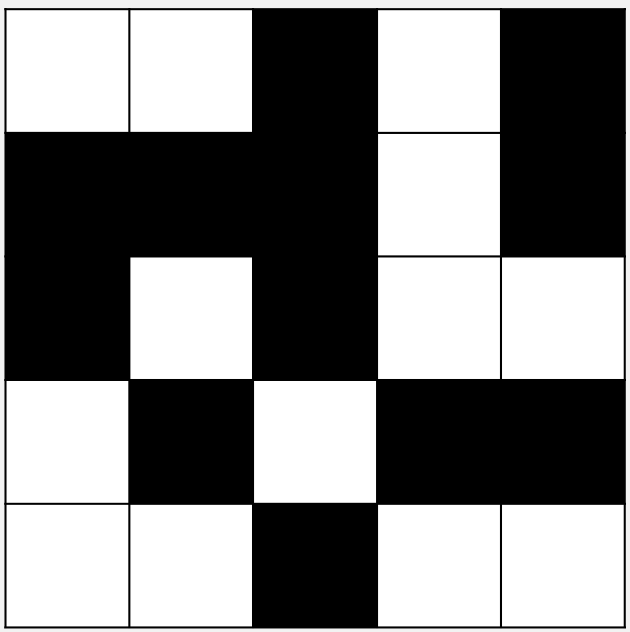


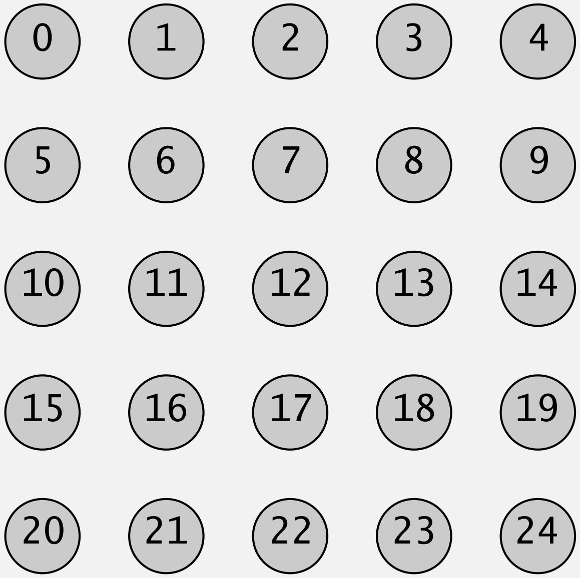
Dynamic-connectivity solution to estimate percolation threshold

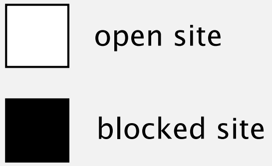
Q. How to check whether an n-by-n system percolates?

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

$n = 5$







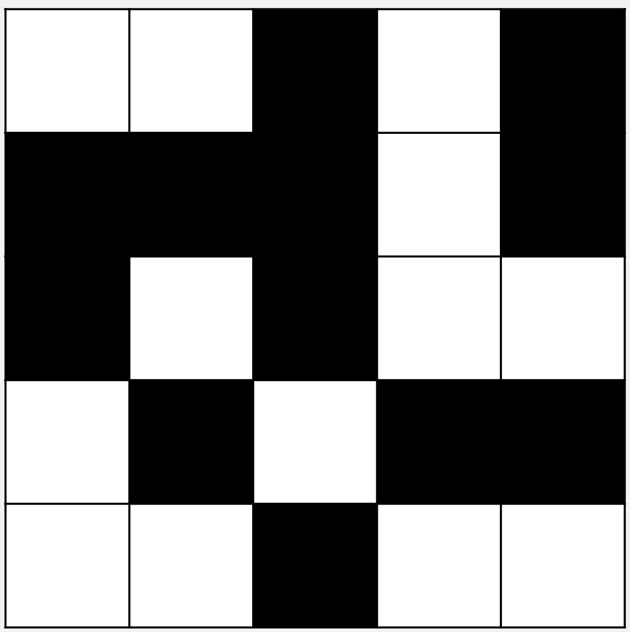
Dynamic-connectivity solution to estimate percolation threshold

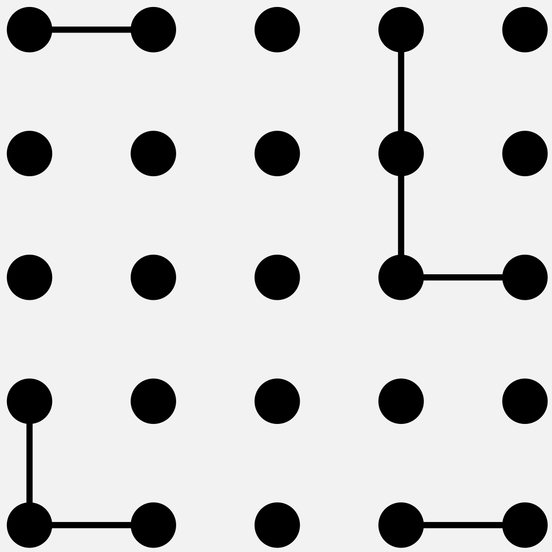
Q. How to check whether an n-by-n system percolates?

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

4 possible neighbors: left, right, top, bottom

$n = 5$





open site

blocked site

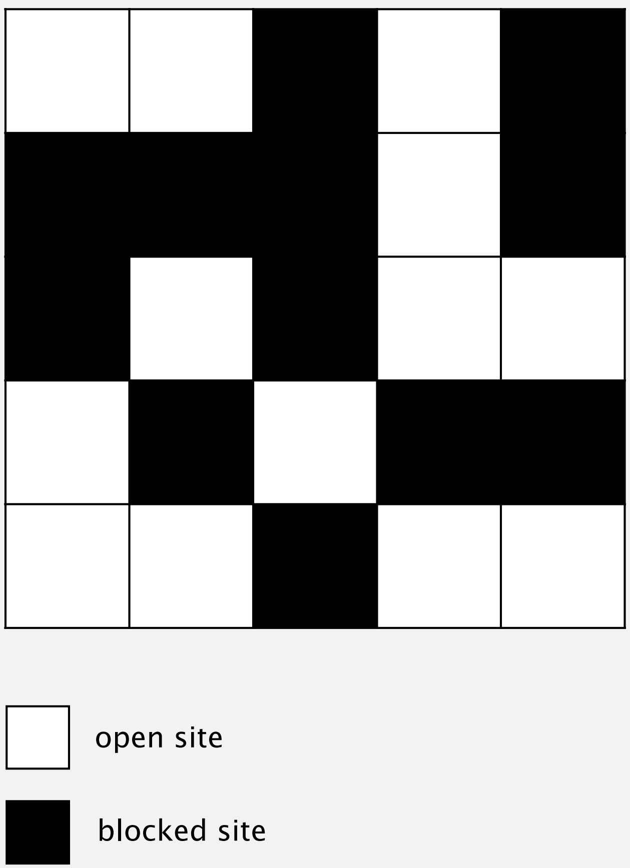
Dynamic-connectivity solution to estimate percolation threshold

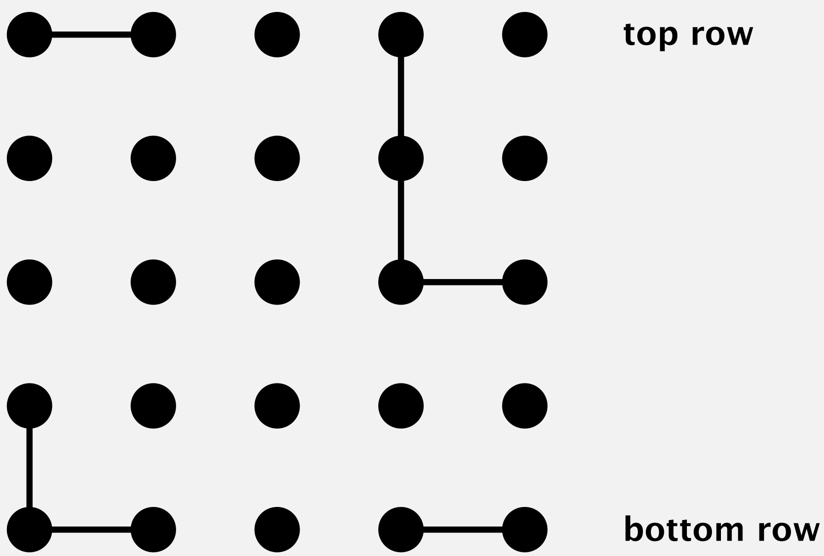
Q. How to check whether an n-by-n system percolates?

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

brute-force algorithm: 2n find queries

$n = 5$



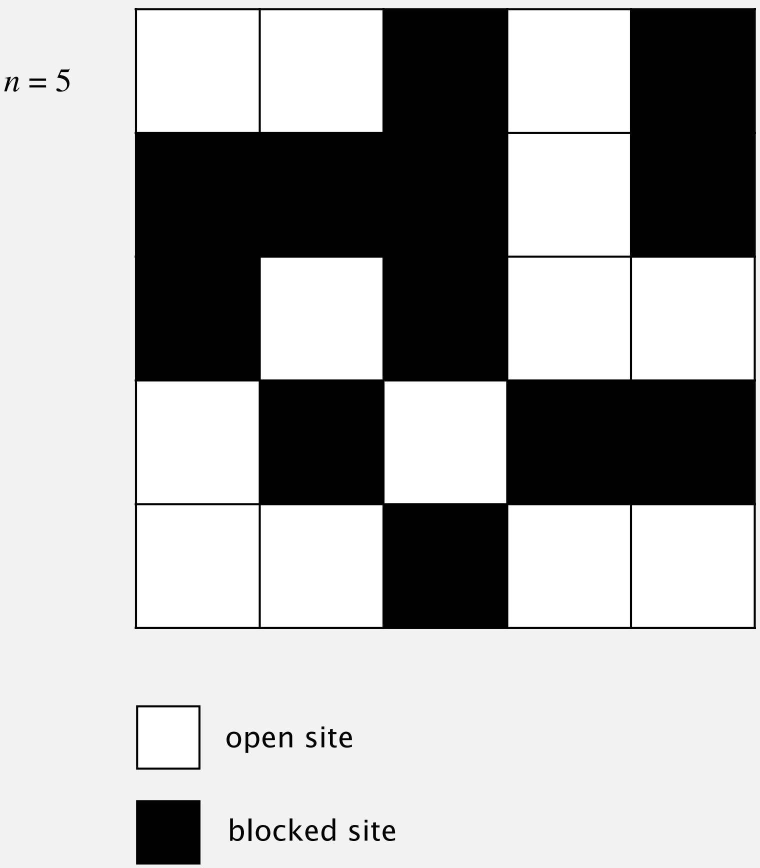


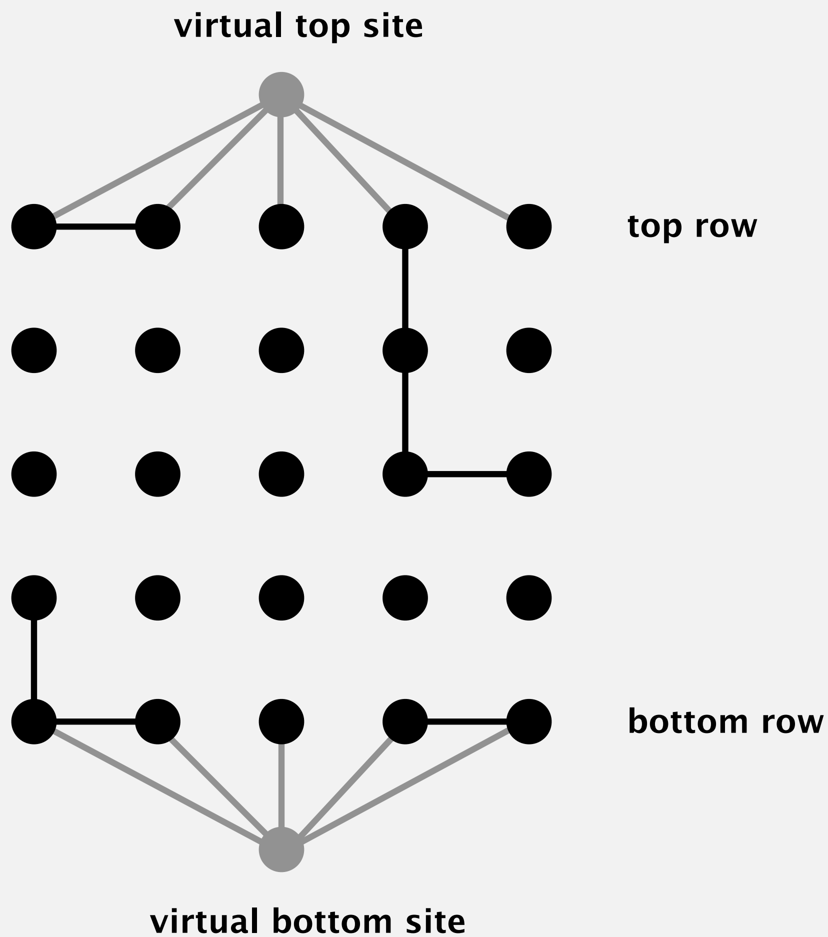
Dynamic-connectivity solution to estimate percolation threshold

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

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

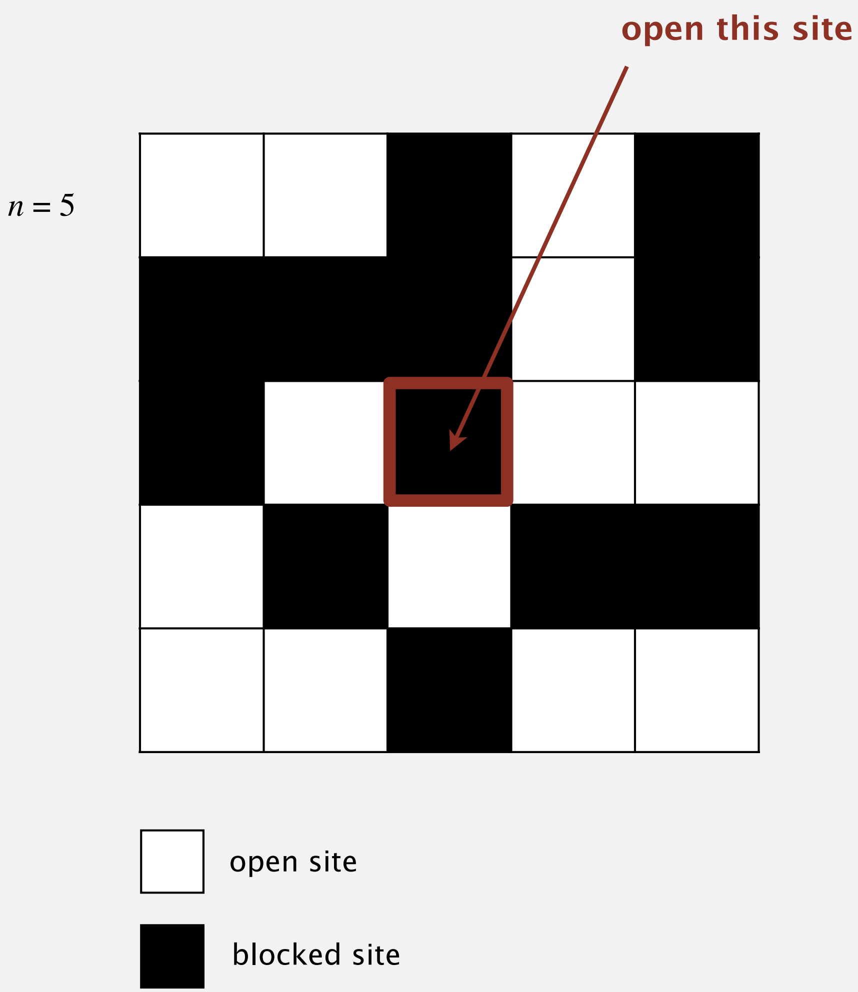
more efficient algorithm: only 1 connected query

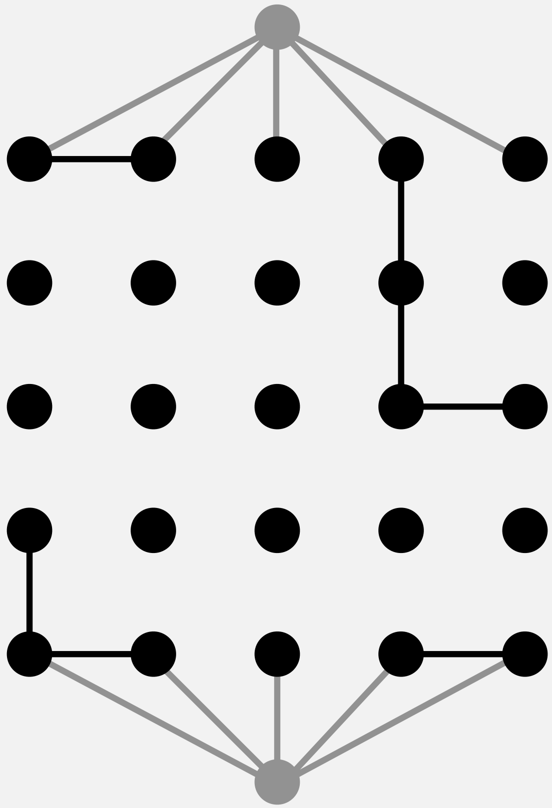




Dynamic-connectivity solution to estimate percolation threshold

Q. How to model opening a new site?



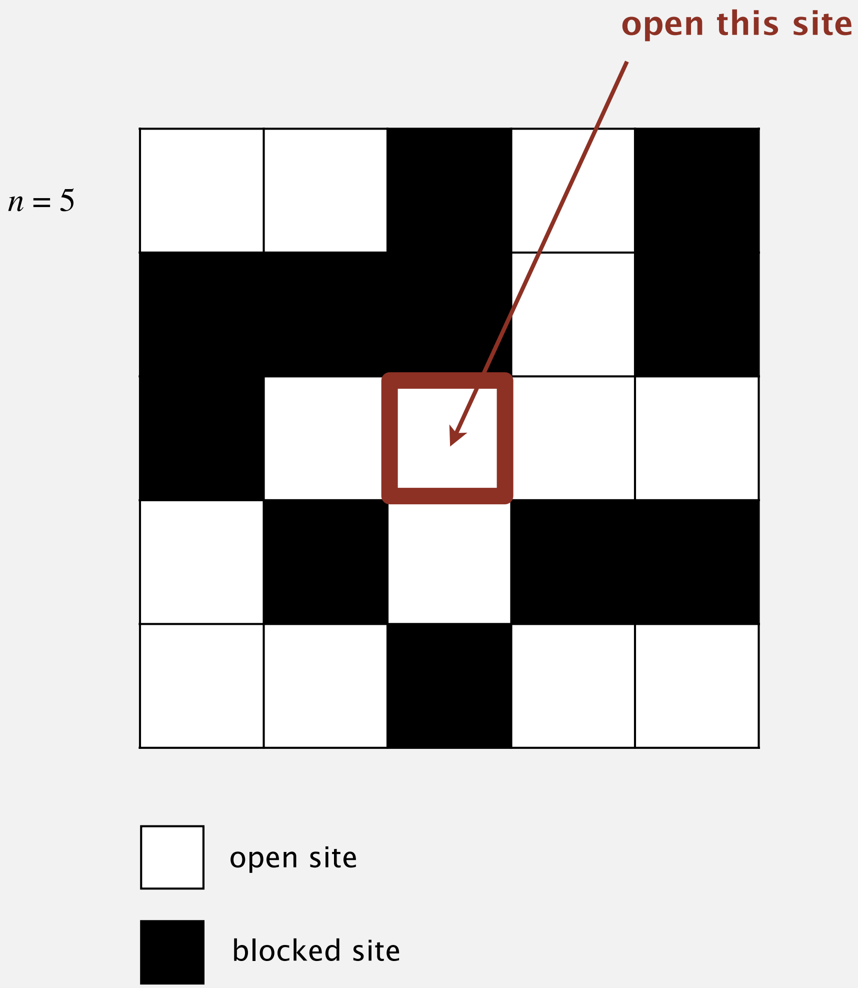


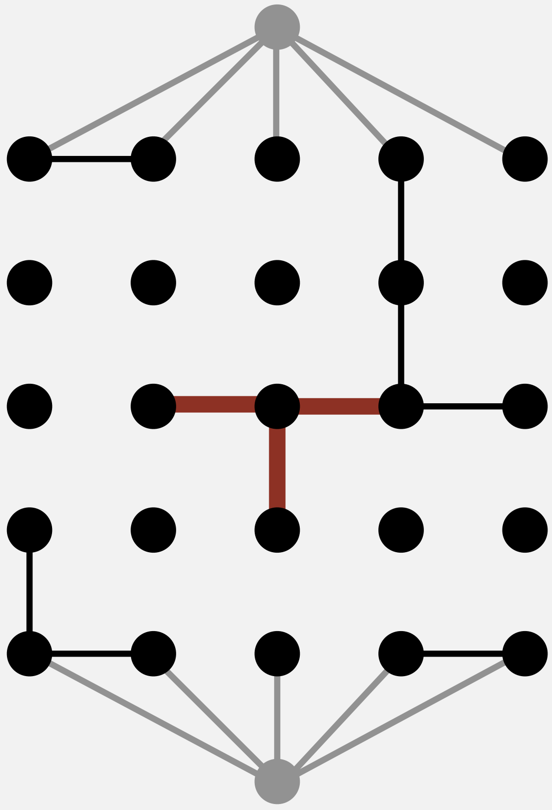
Dynamic-connectivity solution to estimate percolation threshold

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



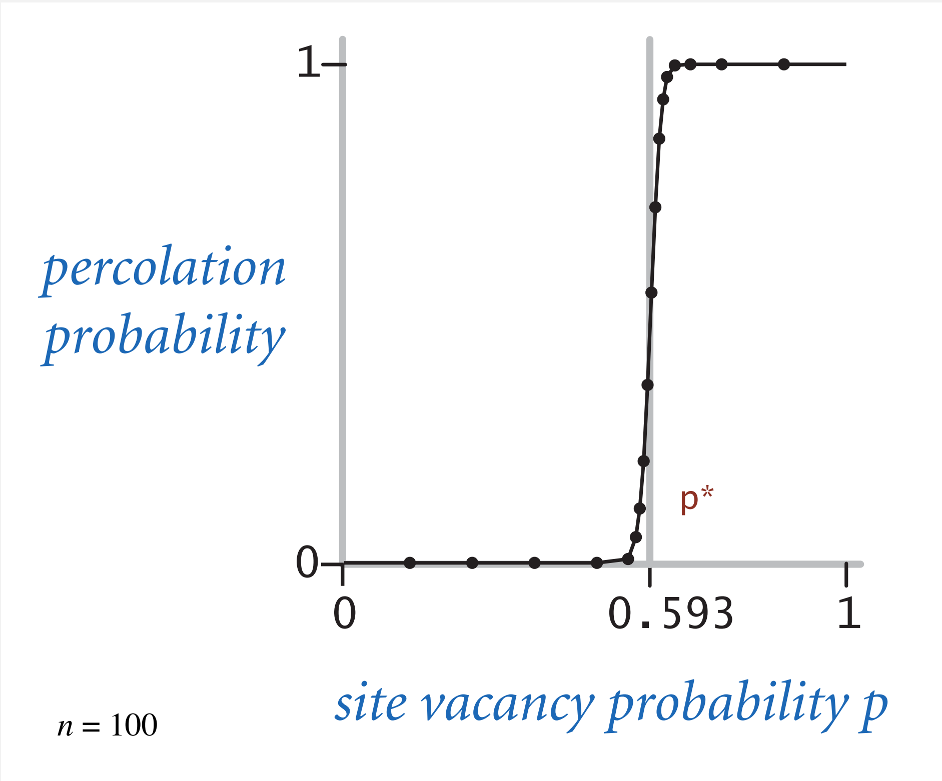


Percolation threshold

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.
- Correct? 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.