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.

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

Google
Yahoo!
bing
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

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.
Why study algorithms?

To become a proficient programmer.

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

— Linus Torvalds (creator of Linux)

“Algorithms + Data Structures = Programs.”

— Niklaus Wirth
Why study algorithms?

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

“Computer models mirroring real life have become crucial for most advances made in chemistry today…. Today the computer is just as important a tool for chemists as the test tube.”

— Royal Swedish Academy of Sciences

(Nobel Prize in Chemistry 2013)

Martin Karplus, Michael Levitt, and Arieh Warshel
Why study algorithms?

To solve problems that could not otherwise be addressed.

http://www.youtube.com/watch?v=ua7YIN4eL_w
Why study algorithms?

Everybody else is doing it.
Why study algorithms?

For fun and profit.
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.

Why study anything else?
Lectures

Traditional lectures. Introduce new material.

Electronic devices. Permitted, but only to enhance lecture.

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Where</th>
<th>Who</th>
<th>Office Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>L01</td>
<td>TTh 11–12:20</td>
<td>Friend 101</td>
<td>Kevin Wayne</td>
<td>see web</td>
</tr>
</tbody>
</table>
Lectures

**Traditional lectures.** Introduce new material.

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

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Where</th>
<th>Who</th>
<th>Office Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>L01</td>
<td>TTh 11–12:20</td>
<td>Friend 101</td>
<td>Kevin Wayne</td>
<td>see web</td>
</tr>
<tr>
<td>L02</td>
<td>TTh 11–12:20</td>
<td>Shererd 001</td>
<td>Andy Guna</td>
<td>see web</td>
</tr>
</tbody>
</table>
## Precepts

Discussion, problem-solving, background for assignments.

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

† lead preceptor
Coursework and grading

Programming assignments. 45%
- Due on Wednesday at 11pm via electronic submission.
- Collaboration/lateness policies: see web.

Exercises. 10%
- Due on Sundays at 11pm in Blackboard.
- Collaboration/lateness policies: see web.

Exams. 15% + 30%
- Midterm (in class on Tuesday, October 21).
- Final (to be scheduled by Registrar).

Staff discretion. [adjust borderline cases]
- Report errata.
- Contribute to Piazza discussion forum.
- Attend and participate in precept/lecture.

Available in hardcover and Kindle.

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

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

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

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

http://algs4.cs.princeton.edu
Resources (people)

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

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

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

http://piazza.com/princeton/fall2014/cos226
http://www.princeton.edu/~cos226
http://labta.cs.princeton.edu
What's ahead?

Today. Attend traditional lecture (everyone).

Tomorrow. Attend precept (everyone).

FOR $i = 1$ to $N$

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

protip: start early
Q+A

Not registered? Go to any precept tomorrow.
Change precept? Use SCORE.

Haven't taken COS 126? See COS placement officer.
Placed out of COS 126? Review Sections 1.1–1.2 of Algorithms 4/e.
1.5 Union-Find

- dynamic connectivity
- quick find
- quick union
- improvements
- applications
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 elements, support two operation:

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

connect 4 and 3
connect 3 and 8
connect 6 and 5
connect 9 and 4
connect 2 and 1

Are 0 and 7 connected? ×
Are 8 and 9 connected? ✓
connect 5 and 0
connect 7 and 2
connect 6 and 1
connect 1 and 0
Are 0 and 7 connected? ✓
Q. Is there a path connecting elements $p$ and $q$?

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

A. Yes.
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.

---

Modeling the elements

Applications involve manipulating elements of all types.

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

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

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

---

Can use symbol table to translate from site names to integers: stay tuned (Chapter 3)
Modeling the connections

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

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

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

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

3 connected components
Implementing the operations

**Find.** In which component is element $p$?

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

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

\[
\begin{array}{c}
\{0\} \{1\} \{2\} \{3\} \\
0 \quad 1 \\
4 \quad 5 \\
6 \quad 7 \\
\{4\} \{5\} \{6\} \{7\} \\
\end{array}
\Rightarrow
\begin{array}{c}
\{0\} \{1\} \{2\} \{3\} \\
0 \quad 1 \quad 2 \\
4 \quad 5 \quad 6 \quad 7 \\
\{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \\
\end{array}
\]

union(2, 5)

\[
\begin{array}{c}
\{0\} \{1\} \{2\} \{3\} \\
0 \quad 1 \\
4 \quad 5 \\
6 \quad 7 \\
\{4\} \{5\} \{6\} \{7\} \\
\end{array}
\Rightarrow
\begin{array}{c}
\{0\} \{1\} \{2\} \{3\} \\
0 \quad 1 \quad 2 \\
4 \quad 5 \quad 6 \quad 7 \\
\{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \\
\end{array}
\]

3 connected components

2 connected components
**Union-find data type (API)**

**Goal.** Design efficient data structure for union-find.
- Number of elements $N$ can be huge.
- Number of operations $M$ can be huge.
- Union and find operations can be intermixed.

```java
public class UF {
  UF(int N) {
    // initialize union-find data structure with N singleton elements (0 to N – 1)
  }
  void union(int p, int q) {
    // add connection between p and q
  }
  int find(int p) {
    // component identifier for p (0 to N – 1)
  }
  boolean connected(int p, int q) {
    // are p and q in the same component?
    return find(p) == find(q);
  }
}
```

1-line implementation of connected()

- $N$: Number of elements
- $M$: Number of operations
Dynamic-connectivity client

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

```java
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
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 \( p \).

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected
Quick-find  [eager approach]

Data structure.

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

Find. What is the id of p?

Connected. Do p and q have the same id?

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>id[]</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>id[]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

after union of 6 and 1

problem: many values can change
Quick-find demo
Quick-find demo

id[]  0  1  2  3  4  5  6  7  8  9
       1  1  1  8  8  1  1  1  8  8
Quick-find: Java implementation

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

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

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

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

- set id of each element to itself (N array accesses)
- return the id of p (1 array access)
- change all entries with id[p] to id[q] (at most 2N + 2 array accesses)
Quick-find is too slow

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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>initialize</th>
<th>union</th>
<th>find</th>
<th>connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 $N$ elements.
Quadratic algorithms do not scale

Rough standard (for now).

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

Ex. Huge problem for quick-find.

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

Quadratic algorithms don't scale with technology.

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

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\text{id[]} & 0 & 1 & 9 & 4 & 9 & 6 & 6 & 7 & 8 & 9 \\
\end{array}
\]

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

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]...]]].

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.
Quick-union demo
Quick-union demo
public class QuickUnionUF {
    private int[] id;

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

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

    public void union(int p, int q) {
        int i = find(p);
        int j = find(q);
        id[i] = j;
    }
}
Quick-union is also too slow

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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>initialize</th>
<th>union</th>
<th>find</th>
<th>connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>quick-union</td>
<td>N</td>
<td>N †</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

† includes cost of finding two roots

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

**Quick-union defect.**
- Trees can get tall.
- Find/connected too expensive (could be $N$ array accesses).
1.5 UNION-FIND

- dynamic connectivity
- quick find
- quick union
- improvements
- applications
Improvement 1: weighting

Weighted quick-union.

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

Example diagrams:

- Quick-union: Always chooses the better alternative (may put larger tree lower)
- Weighted quick-union: Reasonable alternative: union by height/rank always chooses the better alternative

Diagrams showing the process of unioning trees with different sizes and heights.
Weighted quick-union demo

0 1 2 3 4 5 6 7 8 9
Weighted quick-union demo

![Weighted quick-union demo](image_url)
Weighted quick-union quiz

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

```
0 0 0 0 0 0 7 8 8 8
```

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

A. id[0]
B. id[2]
C. id[6]
D. id[8]
Quick-union and weighted quick-union example

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

average distance to root: 5.11

average distance to root: 1.52

Quick-union and weighted quick-union (100 sites, 88 union() operations)
Weighted quick-union: Java implementation

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

**Find/connected.** Identical to quick-union.

**Union.** Modify quick-union to:

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

```java
int i = find(p);
int j = find(q);
if (i == j) return;
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else { id[j] = i; sz[i] += sz[j]; }
```
Weighted quick-union analysis

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

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

in computer science, $\lg$ means base-2 logarithm
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 \( \lg N \).

Pf. What causes the depth of element \( x \) to increase?
Increases by 1 when tree \( T_1 \) containing \( x \) is merged into another tree \( T_2 \).
- The size of the tree containing \( x \) at least doubles since \( |T_2| \geq |T_1| \).
- Size of tree containing \( x \) can double at most \( \lg N \) times. Why?
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 $\lg N$. 

<table>
<thead>
<tr>
<th>algorithm</th>
<th>initialize</th>
<th>union</th>
<th>find</th>
<th>connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>$N$</td>
<td>$N$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>quick-union</td>
<td>$N$</td>
<td>$N$ $\dagger$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>weighted QU</td>
<td>$N$</td>
<td>$\lg N$ $\dagger$</td>
<td>$\lg N$</td>
<td>$\lg N$</td>
</tr>
</tbody>
</table>

$\dagger$ includes cost of finding two roots
Summary

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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>worst-case time</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>M N</td>
</tr>
<tr>
<td>quick-union</td>
<td>M N</td>
</tr>
<tr>
<td>weighted QU</td>
<td>(N + M \log N)</td>
</tr>
<tr>
<td>QU + path compression</td>
<td>(N + M \log N)</td>
</tr>
<tr>
<td>weighted QU + path compression</td>
<td>(N + M \lg^* N)</td>
</tr>
</tbody>
</table>

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.
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.
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.
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.

<table>
<thead>
<tr>
<th>model</th>
<th>system</th>
<th>vacant site</th>
<th>occupied site</th>
<th>percolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>electricity</td>
<td>material</td>
<td>conductor</td>
<td>insulated</td>
<td>conducts</td>
</tr>
<tr>
<td>fluid flow</td>
<td>material</td>
<td>empty</td>
<td>blocked</td>
<td>porous</td>
</tr>
<tr>
<td>social interaction</td>
<td>population</td>
<td>person</td>
<td>empty</td>
<td>communicates</td>
</tr>
</tbody>
</table>
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

Legend:
- 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 $p^*$. 

![Monte Carlo simulation diagram](image)

$N = 20$

135 open sites
How to check whether an $N$-by-$N$ system percolates?

Model as a dynamic connectivity 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 and name them 0 to $N^2 - 1$. 

![Diagram of a 5x5 grid with open and blocked sites](image)
Q. How to check whether an $N$-by-$N$ system percolates?

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

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

---

**brute-force algorithm:** $N^2$ calls to `connected()`
Clever trick. Introduce 2 virtual sites (and connections to top and bottom).
- Percolates iff virtual top site is connected to virtual bottom site.

Dynamic connectivity solution to estimate percolation threshold
Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

\[ N = 5 \]

open this site
Dynamic connectivity solution to estimate percolation threshold

**Q.** How to model opening a new site?

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

---

**Diagram:**

- Open site
- Blocked site

**Figure:**

- Grid representation
- Union operation visualization

- $N = 5$
Q. What is percolation threshold $p^*$?
A. About 0.592746 for large square lattices.

Fast algorithm enables accurate answer to scientific question.
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.