Performance

Section 4.1

The challenge

- The challenge
- Empirical analysis
- Mathematical models
- Doubling method
- Familiar examples

The challenge (since the earliest days of computing machines)

"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time?"

— Charles Babbage

Q. How many times do you have to turn the crank?

Difference Engine #2
Designed by Charles Babbage, c. 1840
Built by London Science Museum, 1991

The challenge (modern version)

Q. Will I be able to use my program to solve a large practical problem?

Q. If not, how might I understand its performance characteristics so as to improve it?

Key insight (Knuth 1970s). Use the scientific method to understand performance.
Three reasons to study program performance

1. To predict program behavior
   - Will my program finish?
   - When will my program finish?

2. To compare algorithms and implementations.
   - Will this change make my program faster?
   - How can I make my program faster?

3. To develop a basis for understanding the problem and for designing new algorithms
   - Enables new technology.
   - Enables new research.

An algorithm is a method for solving a problem that is suitable for implementation as a computer program.

An algorithm design success story

N-body simulation
- Coal: Simulate gravitational interactions among N bodies.
- Brute-force algorithm uses $N^2$ steps per time unit.
- Issue (1970s): Too slow to address scientific problems of interest.
- Success story: Barnes-Hut algorithm uses $N \log N$ steps and enables new research.

Quick aside: binary logarithms

Def. The binary logarithm of a number $N$ (written $\lfloor \log_2 N \rfloor$) is the number $x$ satisfying $2^x = N$.

Q. How many recursive calls for `convert(N)`?

A. Largest integer less than or equal to $\log_2 N$ (written $\lfloor \log_2 N \rfloor$).

Fact. The number of bits in the binary representation of $N$ is $1 + \lfloor \log_2 N \rfloor$.

Fact. Binary logarithms arise in the study of algorithms based on recursively solving problems half the size (divide-and-conquer algorithms), like convert, FFT and Barnes-Hut.
An algorithmic challenge: 3-sum problem

Three-sum. Given $N$ integers, enumerate the triples that sum to 0.

For starters, just count them (might choose to process them all).

Applications in computational geometry
- Find collinear points.
- Does one polygon fit inside another?
- Robot motion planning.
- (a surprisingly long list)

Q. Can we solve this problem for $N = 1$ million?

Three-sum implementation

“Brute force” starting point
- Process all possible triples.
- Increment counter when sum is 0.

public static int count(int[] a)
{
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}

Q. How much time will this program take for $N = 1$ million?

Keep $1 < j < k$ to avoid processing each triple 6 times

 Columbus 2012
A first step in analyzing running time

Find representative inputs
- Option 1: Collect actual potential input data.
- Option 2: Write a program to generate representative inputs.

Input generator for ThreeSum

```java
public class Generator {
    // Generate N integers in [M, M]
    public static void main(String[] args) {
        int M = Integer.parseInt(args[0]);
        int N = Integer.parseInt(args[1]);
        for (int i = 0; i < N; i++) {
            StdOut.println(StdRandom.uniform(-M, M));
        }
    }
}
```

Empirical analysis

Run experiments
- Start with a moderate size.
- Measure and record running time.
- Double size.
- Repeat.
- Tabulate and plot results.

Tabulate and plot results

```
<table>
<thead>
<tr>
<th>N</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
</tr>
<tr>
<td>4000</td>
<td>11</td>
</tr>
<tr>
<td>8000</td>
<td>248</td>
</tr>
</tbody>
</table>
```

Aside: experimentation in CS

is _virtually free_, particularly by comparison with other sciences.

Bottom line. _No excuse_ for not running experiments to understand costs.

Data analysis

Curve fitting
- Plot on log-log scale.
- If points are on a straight line (often the case), a power law holds—a curve of the form \(aN^b\) fits.
- The exponent \(b\) is the slope of the line.
- Solve for \(a\) with the data.

Log-log plot

```
log \(T_N\) vs. log \(N\)
```

Do the math
```
\[ T_N = aN^b \]
```

- \(a\) substitute values from experiment
- \(b\) solve for \(a\)
- \(T_N\) substitute
- equation for straight line of slope 3 raise 2 to a power of both sides
- a curve that fits the data
Prediction and verification

**Hypothesis.** Running time of ThreeSum is $4.84 \times 10^{-10} \times N^3$.

**Prediction.** Running time for $N = 16,000$ will be 1982 seconds.

Q. How much time will this program take for $N = 1$ million?

A. 484 million seconds (more than 15 years).

Another hypothesis

**Hypothesis.** Running times on different computers differ by only a constant factor.
Mathematical models for running time

Q. Can we write down an accurate formula for the running time of a computer program?

A. (Prevailing wisdom, 1960s) No, too complicated.


• Determine the set of operations.
• Find the cost of each operation (depends on computer and system software).
• Find the frequency of execution of each operation (depends on algorithm and inputs).
• Total running time: sum of cost × frequency for all operations.

Warmup: 1-sum

public static int count(int[] a) {
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        if (a[i] == 0)
            cnt++;
    return cnt;
}

Q. Formula for total running time?

A. \( cN + 26.5 \) nanoseconds, where \( c \) is between 2 and 2.5, depending on input.

Warmup: 2-sum

public static int count(int[] a) {
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = 1; j < N; j++)
            if (a[i] + a[j] == 0)
                cnt++;
    return cnt;
}

Q. Formula for total running time?

A. \( cN^2 + cN + c \) nanoseconds, where... [complicated definitions].

Simplifying the calculations

Tilde notation

• Use only the fastest-growing term.
• Ignore the slower-growing terms.

Rationale

• When \( N \) is large, ignored terms are negligible.
• When \( N \) is small, everything is negligible.

Def. \( f(N) \sim g(N) \) means \( f(N)/g(N) \to 1 \) as \( N \to \infty \)

Example. \( 6N^2 + 20N + 5 \sim 6N^2 \)

Q. Formula for 2-sum running time when count is not large (typical case)?

A. \( \sim 6N^2 \) nanoseconds.
Mathematical model for 3-sum

```java
public static int count(int[] a) {
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}
```

Q. Formula for total running time when return value is not large (typical case)?

A. \( N^2/2 \) nanoseconds. ✓ matches \( 4.84 \times 10^{-10} \times N^2 \) empirical hypothesis

### Context

**Scientific method**
- **Observe** some feature of the natural world.
- **Hypothesize** a model consistent with observations.
- **Predict** events using the hypothesis.
- **Verify** the predictions by making further observations.
- **Validate** by refining until hypothesis and observations agree.

**Empirical analysis of programs**
- "Feature of natural world" is time taken by a program on a computer.
- Fit a curve to experimental data to get a formula for running time as a function of \( N \).
- Useful for predicting, but not explaining.

**Mathematical analysis of algorithms**
- Analyze algorithm to develop a formula for running time as a function of \( N \).
- Useful for predicting and explaining.
- Might involve advanced mathematics.
- Applies to any computer.

**Good news.** Mathematical models are easier to formulate in CS than in other sciences.
**Key questions and answers**

**Q. Is the running time of my program — a \( N^p \) seconds?**

**A.** Yes, there’s good chance of that. Might also have a \((\log N)^c\) factor.

**Q. How do you know?**

**A.** Computer scientists have applied such models for decades to many, many specific algorithms and applications.

**A.** Programs are built from simple constructs (examples to follow).

**A.** Real-world data is also often simply structured.

**A.** Deep connections exist between such models and a wide variety of discrete structures (including some programs).

---

**Doubling method**

**Hypothesis.** The running time of my program is \( T_N \sim N^p \).

**Consequence.** As \( N \) increases, \( T_N/T_{N/2} \) approaches \( 2^p \).

**Proof:**

\[
\frac{a(2N)^b}{N^b} = 2^b
\]

**Doubling method**

- Start with a moderate size.
- Measure and record running time.
- Double size.
- Repeat while you can afford it.
- Verify that ratios of running times approach \( 2^p \).
- Predict by extrapolation: multiply by \( 2^p \) to estimate \( T_{2N} \) and repeat.

**Bottom line.** It is often easy to meet the challenge of predicting performance.

**Order of growth**

**Def.** If a function \( f(N) \sim aN^p \) we say that \( g(N) \) is the order of growth of the function.

**Hypothesis.** Order of growth is a property of the algorithm, not the computer or the system.

**Evidence.** Known to be true for many, many programs with simple and similar structure.

**Experimental validation**

When we execute a program on a computer that is \( X \) times faster, we expect the program to be \( X \) times faster.

**Explanation with mathematical model**

Machine- and system-dependent features of the model are all constants.

**Order of growth**

**Hypothesis.** The order of growth of the running time of my program is \( N^p \) \((\log N)^c\).

**Evidence.** Known to be true for many, many programs with simple and similar structure.

**Linear (N)**

```java
public static void f(int N) { 
    for (int i = 0; i < N; i++)
        ... f(N/2)...
    }
}
```

**Quadratic (N^2)**

```java
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        ...
```

**Cubic (N^3)**

```java
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            ...
```

**Logarithmic (log N)**

```java
for (int i = 0; i < N; i++)
    ...
```

**Linearithmic (N log N)**

```java
public static void f(int N) { 
    if (N == 0) return;
    ... f(N/2)...
    }
}
```

**Exponential (2^N)**

```java
public static void f(int N) { 
    if (N == 0) return;
    ... f(N-1)...
    ...
}
```

Stay tuned for examples.
Order of growth classifications

<table>
<thead>
<tr>
<th>Order of growth</th>
<th>Slope of line in log-log plot (b)</th>
<th>Factor for doubling method (2^b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>logN</td>
<td>1</td>
</tr>
<tr>
<td>Linear</td>
<td>N</td>
<td>1, 2</td>
</tr>
<tr>
<td>Quadratic</td>
<td>N^2</td>
<td>2, 4</td>
</tr>
<tr>
<td>Cubic</td>
<td>N^3</td>
<td>3, 8</td>
</tr>
</tbody>
</table>

An important implication

**Moore’s Law.** Computer power increases by a roughly a factor of 2 every 2 years.

Q. My *problem size* also doubles every 2 years. How much do I need to spend to get my job done?

A. You can’t afford to use a quadratic algorithm (or worse) to address increasing problem sizes.

Do the math

\[ T_N = aN^3 \quad \text{running time today} \]

\[ T_{2N} = \left(\frac{a}{2}\right)(2N)^3 \quad \text{running time in 2 years} \]

- \[ T_N = 4aN^3 \]
- \[ T_{2N} = 4T_N \]

Meeting the challenge

My program is taking too long to finish. I’m going out to get a pizza.

Mine, too. I’m going to run a doubling experiment.

Hmmm. Still didn’t finish.

The experiment showed my program to have a higher order of growth than I expected. I found and fixed the bug. Time for some pizza!

Doubling experiments provide good insight on program performance

- Best practice to plan realistic experiments for debugging, anyway.
- Having some idea about performance is better than having no idea.
- Performance matters in many, many situations.

Caveats

It is sometimes not so easy to meet the challenge of predicting performance.

- There are many other apps running on my computer!
- We need more terms in the math model: \( N \log N + 100N \)
- Your input model is too simple: My real input data is completely different.
- What happens when the leading term oscillates?
- Your machine model is too simple: My computer has parallel processors and a cache.
- Where’s the log factor?

Good news. Doubling method is robust in the face of many of these challenges.
8. Performance analysis

- The challenge
- Empirical analysis
- Mathematical models
- Doubling hypothesis
- Familiar examples

Example: Gambler’s ruin simulation

Q. How long to compute chance of doubling 1 million dollars?

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T_0 )</th>
<th>( Tu/Tw )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>17</td>
<td>4.25</td>
</tr>
<tr>
<td>4000</td>
<td>56</td>
<td>3.29</td>
</tr>
<tr>
<td>8000</td>
<td>286</td>
<td>5.10</td>
</tr>
<tr>
<td>16000</td>
<td>1172</td>
<td>4.09</td>
</tr>
<tr>
<td>32000</td>
<td>1172 x 4 = 4688</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102400</td>
<td>1172 x 4 = 480512</td>
<td>4</td>
</tr>
</tbody>
</table>

math model says order of growth should be \( N^2 \)

A. 4.8 million seconds (about 2 months).

Pop quiz on performance

Q. Let \( T_0 \) be the running time of program Mystery and consider these experiments:

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T_0 ) (in seconds)</th>
<th>( Tu/Tw )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>4000</td>
<td>80</td>
<td>4</td>
</tr>
<tr>
<td>8000</td>
<td>320</td>
<td>4</td>
</tr>
</tbody>
</table>

Q. Predict the running time for \( N = 64,000 \).

Q. Estimate the order of growth.
Another example: Coupon collector

Q. How long to simulate collecting 1 million coupons?

$$\begin{array}{c|c|c}
 N & T_n & T_n/T_n \\
\hline
 125000 & 7 & \ \\
 250000 & 14 & 2 \\
 500000 & 31 & 2.21 \\
 1000000 & 31 \times 2 = 63 & 2 \\
\end{array}$$

The math model says order of growth should be \(N/\log N\).

A. About 1 minute, might run out of memory trying for 1 billion.

Analyzing typical memory requirements

A bit is 0 or 1 and the basic unit of memory.

1. megabyte (MB) is about 1 million bytes.
2. gigabyte (GB) is about 1 billion bytes.

<table>
<thead>
<tr>
<th>Primitive-type values</th>
<th>System-supported data structures (typical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>bytes</td>
</tr>
<tr>
<td>boolean</td>
<td>1 byte</td>
</tr>
<tr>
<td>char</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

Example. 2000-by-2000 double array uses ~32MB.

Summary

Use computational experiments, mathematical analysis, and the scientific method to learn whether your program might be useful to solve a large problem.

Q. What if it’s not fast enough?

A.

- does it scale?
- buy a new computer and solve bigger problems
- learn a better algorithm

Maybe
- found one?
- yes
- no

Plenty of new algorithms awaiting discovery. Example: Solve 3-sum efficiently

Case in point

Not so long ago, 2 CS grad students had a program to index and rank the web (to enable search).

- buy a new computer and solve bigger problems
- learn a better algorithm

Paging System

Lesson. Performance matters!
Section 4.1: Performance

http://introcs.cs.princeton.edu

Image source