7. Performance

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. 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 $\lg N$) is the number $x$ satisfying $2^x = N$.

Q. How many recursive calls for convert(int)?

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

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

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.

Another algorithm design success story

Discrete Fourier transform
• Goal: Break down waveform of $N$ samples into periodic components.
• Applications: digital signal processing, spectroscopy, ... 
• Brute-force algorithm uses $N^2$ steps.
• Issue (1950s): Too slow to address commercial applications of interest.
• Success story: FFT algorithm uses $N \log N$ steps and enables new technology.
An algorithmic challenge: 3-sum problem

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

For simplicity, just count them.

public class ThreeSum
{
    public static int count(int[] a)
    { /* See next slide. */ }
    
    public static void main(String[] args)
    { int[] a = StdIn.readInts();
        StdOut.println(count(a));
    }
}

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

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

Three-sum implementation

"Brute force" algorithm
- 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 $i < j < k$ to avoid processing each triple 6 times

triples with $1 < j < k$
A first step in analyzing running time

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

Input generator for ThreeSum

```java
class Generator {
    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 input size \( N \).
- Measure and record running time.
- Double input size \( N \).
- Repeat.
- Tabulate and plot results.

Run experiments

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T )</th>
<th>( \log N )</th>
<th>( \log T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.5</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>3.3</td>
<td>4</td>
</tr>
<tr>
<td>4000</td>
<td>11</td>
<td>3.6</td>
<td>5</td>
</tr>
<tr>
<td>8000</td>
<td>24.8</td>
<td>3.9</td>
<td>8</td>
</tr>
</tbody>
</table>

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

- Straight line of slope 3
- \( \log T_N = \log a + 3 \log N \)
- \( T_N = aN^3 \)
- \( a = 4.84 \times 10^{-10} \times N^3 \)
- \( T_N = 4.84 \times 10^{-10} \times N^3 \)

Aside: experimentation in CS

is virtually free, particularly by comparison with other sciences.

Bottom line. No excuse for not running experiments to understand costs.
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.

---

7. Performance

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

```java
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

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

Q. Formula for total running time?

A. \( c_1N^2 + c_2N + c_3 \) 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 \( N \to \infty \)

Ex. \( 5/4 \ N^2 + 13/4 \ N + 53/2 \sim \frac{5}{4} \ N^2 \)

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

A. \( \sim \frac{5}{4} \ N^2 \) nanoseconds.

Note: This page is a part of a larger document, focusing on the mathematical models for running time and providing code snippets and formulas for different scenarios.
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. \( N^3/2 \) nanoseconds. \( \checkmark \) matches \( 4.84 \times 10^{-15} \times N^3 \) 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 $N^a$ seconds?
A. Yes, there's good chance of that. Might also have a $(\log N)$ 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).

Order of growth

Def. If a function $f(N) = o(g(N))$ 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.

Experimental validation

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

Explanations with mathemathical 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 $N^a$.

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

Linear ($N$)

Quadratic ($N^2$)

Logarithmic ($\log N$)

Linearithmic ($N \log N$)

Exponential ($2^N$)

Stay tuned for examples.

Doubling method

Hypothesis. The running time of my program is $T_N \sim a N^b$.

Consequence. As $N$ increases, $T_{2N}/T_N$ approaches $2^b$.

Proof:

No need to calculate a $(l)$.

3-sum example

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$</th>
<th>$T_{2N}/T_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4000</td>
<td>31</td>
<td>7.75</td>
</tr>
<tr>
<td>8000</td>
<td>248</td>
<td>8</td>
</tr>
<tr>
<td>16000</td>
<td>$248 \times 8 = 1984$</td>
<td>8</td>
</tr>
<tr>
<td>32000</td>
<td>$248 \times 8 = 15872$</td>
<td>8</td>
</tr>
<tr>
<td>64000</td>
<td>$248 \times 8 = 32000$</td>
<td>8</td>
</tr>
</tbody>
</table>

Bottom line. It is often easy to meet the challenge of predicting performance.
Order of growth classifications

<table>
<thead>
<tr>
<th>order of growth</th>
<th>function</th>
<th>slope of line in log-log plot</th>
<th>factor for doubling method</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>(\log N)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>logarithmic</td>
<td>(N \log N)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>linear</td>
<td>(N)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>quadratic</td>
<td>(N^2)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>cubic</td>
<td>(N^3)</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

**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.
- Hmmmm. 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.

**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**

- \(T_N = aN^3\) running time today
- \(T_{2N} = (a/2)(2N)^3\) running time in 2 years
  - \(= 4aN^3\)
  - \(= 4T_N\)

**An important implication**

**Caveats**

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

- 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.
- 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.
7. Performance

- 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?

```java
public class Gambler
{
    public static void main(String[] args)
    {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        double start = 0; System.currentTimeMillis() / 1000.0;
        int wins = 0;
        for (int i = 0; i < trials; i++)
        {
            int t = stake;
            while (t > 0 && t < goal)
            {
                if (Math.random() < 0.5) t++;
                else t--;
                if (t == goal) wins++;
            }
            double now = System.currentTimeMillis() / 1000.0;
            System.out.println(wins + " wins in " + trials + " trials.");
            System.out.println("%0.2f seconds\n", now - start);
        }
    }
}
```

A. 4.8 million seconds (about 2 months).

Pop quiz on performance

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

```java
public class Mystery
{
    public static void main(String[] args)
    {
        ...
        int N = Integer.parseInt(args[0]);
        ...
    }
}
```

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T_N ) (in seconds)</th>
<th>( T_N / T_1 )</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. Let \( T_N \) be the running time of program Mystery and consider these experiments:

```java
public class Mystery
{
    public static void main(String[] args)
    {
        ...
        int N = Integer.parseInt(args[0]);
        ...
    }
}
```

<table>
<thead>
<tr>
<th>( N )</th>
<th>( T_N ) (in seconds)</th>
<th>( T_N / T_1 )</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. Let \( T_N \) be the running time of program Mystery and consider these experiments:

```java
public class Mystery
{
    public static void main(String[] args)
    {
        ...
        int N = Integer.parseInt(args[0]);
        ...
    }
}
```

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

Q. Estimate the order of growth.
Pop quiz on performance

Q. Let $T_N$ be the running time of program Mystery and consider these experiments.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$ (in seconds)</th>
<th>$T_N/T_2$</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>
<tr>
<td>16000</td>
<td>320 x 4 = 1280</td>
<td>4</td>
</tr>
<tr>
<td>32000</td>
<td>1280 x 4 = 5120</td>
<td>4</td>
</tr>
<tr>
<td>64000</td>
<td>5120 x 4 = 20480</td>
<td>4</td>
</tr>
</tbody>
</table>

Q. Predict the running time for $N = 64,000$.

A. 20480 seconds.

Q. Estimate the order of growth.

A. $N^2$, since $\log 4 = 2$.

Another example: Coupon collector

Q. How long to simulate collecting 1 million coupons?

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$</th>
<th>$T_N/T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>125000</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>250000</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>500000</td>
<td>31</td>
<td>2.21</td>
</tr>
<tr>
<td>1000000</td>
<td>31 x 2 = 63</td>
<td>2</td>
</tr>
</tbody>
</table>

Another example: Coupon collector

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$</th>
<th>$T_N/T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>125000</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>250000</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>500000</td>
<td>31</td>
<td>2.21</td>
</tr>
<tr>
<td>1000000</td>
<td>31 x 2 = 63</td>
<td>2</td>
</tr>
</tbody>
</table>

Analyzing typical memory requirements

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

A **byte** is eight bits — the smallest addressable unit.

<table>
<thead>
<tr>
<th>Primitive-type values</th>
<th>System–supported data structures (typical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>type</td>
</tr>
<tr>
<td>boolean</td>
<td>int[N]</td>
</tr>
<tr>
<td>char</td>
<td>double[N]</td>
</tr>
<tr>
<td>int</td>
<td>int[N][N]</td>
</tr>
<tr>
<td>float</td>
<td>double[N][N]</td>
</tr>
<tr>
<td>long</td>
<td>String</td>
</tr>
<tr>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>double</td>
<td>8</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. 

- **Yes**: does it scale?
  - **Yes**: invent a better algorithm
  - **No**: learn a better algorithm

- **No**: buy a new computer and solve bigger problems

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).

- **Does it scale?**
  - Yes: buy a new computer and solve bigger problems.
  - No: learn a better algorithm.

- **Changed the world?**
  - Yes: found one.
  - No: invent a better algorithm.

Lesson. Performance matters!

---

7. Performance