4.1 Performance

Running Time

“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

The Challenge

Will my program be able to solve a large practical problem?

Key insight (Knuth 1970s):

Use the scientific method to understand performance.

Scientific Method

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

Principles.
- Experiments must be reproducible;
- Hypotheses must be falsifiable.
Reasons to Analyze Algorithms

Predict performance
• Will my program finish?
• When will my program finish?

Compare algorithms
• Will this change make my program faster?
• How can I make my program faster?

Basis for inventing new ways to solve problems
• Enables new technology
• Enables new research

Algorithmic Successes

Discrete Fourier transform.
• Break down waveform of N samples into periodic components.
• Applications: DVD, JPEG, MRI, astrophysics, ...
• Brute force: N^2 steps.
• FFT algorithm: N log N steps, enables new technology.

Algorithmic Successes

N-body Simulation.
• Simulate gravitational interactions among N bodies.
• Brute force: N^2 steps.
• Barnes-Hut: N log N steps, enables new research.

Example: Three-Sum Problem

Three-sum problem. Given N integers, find triples that sum to 0.
Context. Deeply related to problems in computational geometry.

% more 8ints.txt
30 -30 -20 -10 40 0 10 5
% java ThreeSum < 8ints.txt
4
30 -30 0
30 -20 -10
-30 -10 40
-10 0 10

Q. How would you write a program to solve the problem?
public class ThreeSum {
    // Return number of distinct triples (i, j, k)
    // such that a[i] + a[j] + a[k] == 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;
    }
    public static void main(String[] args) {
        int[] a = StdArrayIO.readInt1D();
        StdOut.println(count(a));
    }
}

Empirical Analysis

Empirical analysis. Run the program for various input sizes.

<table>
<thead>
<tr>
<th>$N$</th>
<th>time †</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.03</td>
</tr>
<tr>
<td>1024</td>
<td>0.26</td>
</tr>
<tr>
<td>2048</td>
<td>2.16</td>
</tr>
<tr>
<td>4096</td>
<td>17.18</td>
</tr>
<tr>
<td>8192</td>
<td>136.76</td>
</tr>
</tbody>
</table>

† Running Linux on Sun-Fire-X4100 with 16GB RAM

Stopwatch

Q. How to time a program?
A. A stopwatch.
Q. **How to time a program?**
A. A *Stopwatch* object.

```java
public class Stopwatch {
    private final long start;
    public Stopwatch() {
        start = System.currentTimeMillis();
    }
    public double elapsedTime() {
        return (System.currentTimeMillis() - start) / 1000.0;
    }
}
```

**Stopwatch**

Q. **How to time a program?**
A. A *Stopwatch* object.

```java
public class Stopwatch {
    private final long start;
    public Stopwatch() {
        start = System.currentTimeMillis();
    }
    public double elapsedTime() {
        return (System.currentTimeMillis() - start) / 1000.0;
    }
}
```

**Stopwatch**

**Empirical Analysis**

**Data analysis.** Plot running time vs. input size $N$.

**Empirical Analysis**

**Data analysis.** Plot running time vs. input size $N$ on a log-log scale.

**Q.** How does running time grow as a function of input size $N$?

**Q.** How does running time grow as a function of input size $N$?

**Initial hypothesis:** Running time satisfies a power law $f(N) = a N^b$.

**On log-log scale,**

$$\log(F(N)) = b \log N + a$$

**Estimate slope by fitting line through data points.**

**Refined hypothesis.** Running time grows as the **cube** of the input size: $a N^3$. 
Doubling Hypothesis

Doubling hypothesis. Quick way to estimate $b$ in a power law hypothesis.

Run program, doubling the size of the input.

<table>
<thead>
<tr>
<th>$N$</th>
<th>time $t$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.033</td>
<td>-</td>
</tr>
<tr>
<td>1024</td>
<td>0.26</td>
<td>7.88</td>
</tr>
<tr>
<td>2048</td>
<td>2.16</td>
<td>8.43</td>
</tr>
<tr>
<td>4096</td>
<td>17.18</td>
<td>7.96</td>
</tr>
<tr>
<td>8192</td>
<td>136.76</td>
<td>7.96</td>
</tr>
</tbody>
</table>

Seems to converge to a constant $c_0$?

Hypthesize that running time is about $a \cdot N^b$ with $b = \lg c_0$

Doubling Challenge 1

Let $F(N)$ be the running time of program Mystery for input $N$.

```
public static Mystery {
    ...
    int N = Integer.parseInt(args[0]);
    ...
}
```

Observation: $F(2N)/F(N)$ is about 4.

What is the order of growth of the running time?

A. Quadratic: $a \cdot N^2$  

$$\frac{a \cdot (2N)^2}{a \cdot (N)^2} = 4$$

Doubling Challenge 2

Let $F(N)$ be the running time of program Mystery for input $N$.

```
public static Mystery {
    ...
    int N = Integer.parseInt(args[0]);
    ...
}
```

Observation: $F(2N)/F(N)$ is about 2.

What is the order of growth of the running time?
Doubling Challenge 2

Let $F(N)$ be the running time of program Mystery for input $N$.

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

Observation: $F(2N)/F(N)$ is about 2.

What is the order of growth of the running time?

A. Linear: $aN$  
\[
\frac{a(2N)}{a(N)} = 2
\]

Could be a $N \log N$  
\[
\frac{a(2N) \log(2N)}{a(N) \log N} = 2 \cdot \frac{2 \log 2}{\log N}
\]

Prediction and Validation

**Hypothesis.** Running time is about $aN^3$ seconds for input of size $N$.

Q. How to estimate $a$?

A. Run the program!

<table>
<thead>
<tr>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>17.18</td>
</tr>
<tr>
<td>4096</td>
<td>17.15</td>
</tr>
<tr>
<td>4096</td>
<td>17.17</td>
</tr>
</tbody>
</table>

\[17.17 = a \frac{4096}{4096} = 1.25 \times 10^{-10}\]

Refined hypothesis. Running time is about $1.25 \times 10^{-10} N^3$ seconds.

**Prediction.** 1100 seconds for $N = 16,384$.

**Observation.**

<table>
<thead>
<tr>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16384</td>
<td>1118.86</td>
</tr>
</tbody>
</table>

\[a = \frac{17.17}{4096} = 1.25 \times 10^{-10}\]

validates hypothesis!

Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution time.

```java
int count = 0;
for (int i = 0; i < N; i++)
    if (a[i] == 0) count++;
```

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>2</td>
</tr>
<tr>
<td>variable assignment</td>
<td>2</td>
</tr>
<tr>
<td>less than comparison</td>
<td>$N+1$</td>
</tr>
<tr>
<td>equal to comparison</td>
<td>$N$</td>
</tr>
<tr>
<td>array access</td>
<td>$N$</td>
</tr>
<tr>
<td>increment</td>
<td>$\leq 2N$</td>
</tr>
</tbody>
</table>
Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution time.

```java
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        if (a[i] + a[j] == 0) count++;
```

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>N + 2</td>
</tr>
<tr>
<td>variable assignment</td>
<td>N + 2</td>
</tr>
<tr>
<td>less than comparison</td>
<td>1/2 (N+1)(N+2))</td>
</tr>
<tr>
<td>equal to comparison</td>
<td>1/2 (N(N-1))</td>
</tr>
<tr>
<td>array access</td>
<td>(N(N-1))</td>
</tr>
<tr>
<td>increment</td>
<td>(\leq N^2)</td>
</tr>
</tbody>
</table>

Those are becoming very tedious to count.

Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution time.

```java
{ 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;
}
```

Inner loop. Focus on instructions in "inner loop."

Tilde Notation

Tilde notation.
- Estimate running time as a function of input size \(N\).
- Ignore lower order terms.
  - when \(N\) is large, terms are negligible
  - when \(N\) is small, we don’t care

Ex 1. \(6N^3 + 17N^2 + 56 \sim 6N^3\)
Ex 2. \(6N^3 + 100N^{4/3} + 56 \sim 6N^3\)
Ex 3. \(6N^3 + 17N^2 \log N \sim 6N^3\)

Technical definition. \(f(N) \sim g(N)\) means \(\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1\)

discard lower-order terms (e.g., \(N = 1000\): 6 trillion vs. 169 million)

Constants in Power Law

Power law. Running time of a typical program is \(\sim aN^b\).

Exponent \(b\) depends on: algorithm.

Constant \(a\) depends on:
- algorithm
- input data
- caching
- machine
- compiler
- garbage collection
- just-in-time compilation
- CPU use by other applications

Our approach. Use doubling hypothesis (or mathematical analysis) to estimate exponent \(b\), run experiments to estimate \(a\).
Analysis: Empirical vs. Mathematical

Empirical analysis.
• Measure running times, plot, and fit curve.
• Easy to perform experiments.
• Model useful for predicting, but not for explaining.

Mathematical analysis.
• Analyze algorithm to estimate # ops as a function of input size.
• May require advanced mathematics.
• Model useful for predicting and explaining.

Critical difference. Mathematical analysis is independent of a particular machine or compiler; applies to machines not yet built.

Order of Growth Classifications

Observation. A small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

<table>
<thead>
<tr>
<th>order of growth</th>
<th>function</th>
<th>factor for doubling hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>logarithmic</td>
<td>\log N</td>
<td>1</td>
</tr>
<tr>
<td>linear</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>linearithmic</td>
<td>N \log N</td>
<td>2</td>
</tr>
<tr>
<td>quadratic</td>
<td>N^2</td>
<td>4</td>
</tr>
<tr>
<td>cubic</td>
<td>N^3</td>
<td>8</td>
</tr>
<tr>
<td>exponential</td>
<td>2^N</td>
<td>2^N</td>
</tr>
</tbody>
</table>

Effect of increasing problem size for a program that runs for a few seconds

Effect of increasing computer speed on problem size that can be solved in a fixed amount of time
**Dynamic Programming**

---

**Binomial Coefficients**

**Binomial coefficient.** \( \binom{n}{k} = \text{number of ways to choose } k \text{ of } n \text{ elements.} \)

**Pascal’s identity.**

\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

contains first element

excludes first element

---

**Sierpinski Triangle**

**Binomial Coefficient.** \( \binom{n}{k} = \text{number of ways to choose } k \text{ of } n \text{ elements.} \)

**Sierpinski triangle.** Color black the odd integers in Pascal’s triangle.

---

**Binomial Coefficients: Poker Odds**

**Binomial coefficient.** \( \binom{n}{k} = \text{number of ways to choose } k \text{ of } n \text{ elements.} \)

**Probability of “quads” in Texas hold ’em:**

\[
\frac{\binom{13}{4} \times \binom{48}{3}}{\binom{52}{7}} = \frac{224,848}{133,784,560} \approx \frac{594}{1} \text{ (about 594 : 1)}
\]

**Probability of 6-4-2-1 split in bridge:**

\[
\frac{4 \times 13 \times 3 \times 13 \times 2 \times 13 \times 1 \times 13}{52 \choose 13} = \frac{29,858,811,840}{635,013,559,000} \approx \frac{21}{1} \text{ (about 21 : 1)}
\]
**Binomial Coefficients: First Attempt**

```java
public class SlowBinomial {
    // Natural recursive implementation
    public static long binomial(long n, long k) {
        if (k == 0) return 1;
        if (n == 0) return 0;
        return binomial(n-1, k-1) + binomial(n-1, k);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int K = Integer.parseInt(args[1]);
        StdOut.println(binomial(N, K));
    }
}
```

**Performance Challenge 3**

**Is this an efficient way to compute binomial coefficients?**

```java
public static long binomial(long n, long k) {
    if (k == 0) return 1;
    if (n == 0) return 0;
    return binomial(n-1, k-1) + binomial(n-1, k);
}
```

A. NO, NO, NO: same essential recomputation problem as naive Fibonacci.

**Timing Experiments**

- Direct recursive solution.
- Increase n by 1, running time increases by about 4x
- Running Linux on Sun-Fire X4100 with 16GB RAM

<table>
<thead>
<tr>
<th>(2n, n)</th>
<th>time †</th>
</tr>
</thead>
<tbody>
<tr>
<td>(26, 13)</td>
<td>0.46</td>
</tr>
<tr>
<td>(28, 14)</td>
<td>1.27</td>
</tr>
<tr>
<td>(30, 15)</td>
<td>4.30</td>
</tr>
<tr>
<td>(32, 16)</td>
<td>15.69</td>
</tr>
<tr>
<td>(34, 17)</td>
<td>57.40</td>
</tr>
<tr>
<td>(36, 18)</td>
<td>230.42</td>
</tr>
</tbody>
</table>

Q. Is running time linear, quadratic, cubic, exponential in n?
Performance Challenge 4

Let $F(N)$ be the time to compute $\text{binomial}(2N, N)$ using the naive algorithm.

Observation: $F(N+1)/F(N)$ is about 4.

What is the order of growth of the running time?

A. EXPONENTIAL: $a^N$

Will not finish unless $N$ is small.

Key idea. Save solutions to subproblems to avoid recomputation.

Tradeoff. Trade (a little) memory for (a huge amount of) time.

Binomial Coefficients: Dynamic Programming

```
public class Binomial {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int K = Integer.parseInt(args[1]);
        long[][] bin = new long[N+1][K+1];

        // base cases
        for (int k = 1; k <= K; k++) bin[0][k] = 0;
        for (int n = 0; n <= N; n++) bin[n][0] = 1;

        // bottom-up dynamic programming
        for (int n = 1; n <= N; n++)
            for (int k = 1; k <= K; k++)
                bin[n][k] = bin[n-1][k-1] + bin[n-1][k];

        // print results
        StdOut.println(bin[N][K]);
    }
}
```
Timing Experiments

Timing experiments for binomial coefficients with dynamic programming.

<table>
<thead>
<tr>
<th>(2n, n)</th>
<th>time $^\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(26, 13)</td>
<td>instant</td>
</tr>
<tr>
<td>(28, 14)</td>
<td>instant</td>
</tr>
<tr>
<td>(30, 15)</td>
<td>instant</td>
</tr>
<tr>
<td>(32, 16)</td>
<td>instant</td>
</tr>
<tr>
<td>(34, 17)</td>
<td>instant</td>
</tr>
<tr>
<td>(36, 18)</td>
<td>instant</td>
</tr>
</tbody>
</table>

† Running Linux on Sun-Fire-X4100 with 16GB RAM

Performance Challenge 5

Let \(F(N)\) be the time to compute \(\text{binomial}(2N, N)\) using dynamic programming.

```plaintext
for (int n = 1; n <= 2*N; n++)
    for (int k = 1; k <= N; k++)
        bin[n][k] = bin[n-1][k-1] + bin[n-1][k];
```

What is the order of growth of the running time?

A. Quadratic: \(N^2\)

Effectively instantaneous for small \(N\).

Key point: There is PROFOUND DIFFERENCE between \(4^N\) and \(N^2\).
**Memory**

Typical Memory Requirements for Java Data Types

- **Bit.** 0 or 1.
- **Byte.** 8 bits.
- **Megabyte (MB).** $2^{10}$ bytes ~ 1 million bytes.
- **Gigabyte (GB).** $2^{20}$ bytes ~ 1 billion bytes.

<table>
<thead>
<tr>
<th>type</th>
<th>bytes</th>
<th>type</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
<td>int[]</td>
<td>$4N + 16$</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>double[]</td>
<td>$8N + 16$</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
<td>int[][]</td>
<td>$4N^2 + 20N + 16$</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>double[][]</td>
<td>$8N^2 + 20N + 16$</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>String</td>
<td>2N + 40</td>
</tr>
</tbody>
</table>

Q. What’s the biggest double array you can store on your computer?

Performance Challenge 6

How much memory does this program use (as a function of N)?

```java
public class RandomWalk {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int[][] count = new int[N][N];
        int x = N/2;
        int y = N/2;
        for (int i = 0; i < N; i++) {
            // no new variable declared in loop
            count[x][y]++;
        }
    }
}
```

A. $\sim 4N^2$ bytes.

Performance Challenge 6

How much memory does this program use (as a function of N)?

```java
public class RandomWalk {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int[][] count = new int[N][N];
        int x = N/2;
        int y = N/2;
        for (int i = 0; i < N; i++) {
            // no new variable declared in loop
            count[x][y]++;
        }
    }
}
```
Summary

Q. How can I evaluate the performance of my program?
A. Computational experiments, mathematical analysis, scientific method

Q. What if it’s not fast enough? Not enough memory?
• Understand why.
• Buy a faster computer.
• Learn a better algorithm (COS 226, COS 423).
• Discover a new algorithm.

<table>
<thead>
<tr>
<th>attribute</th>
<th>better machine</th>
<th>better algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>$$$ or more.</td>
<td>$ or less.</td>
</tr>
<tr>
<td>applicability</td>
<td>makes &quot;everything&quot; run faster</td>
<td>does not apply to some problems</td>
</tr>
<tr>
<td>improvement</td>
<td>quantitative improvements</td>
<td>dramatic qualitative improvements possible</td>
</tr>
</tbody>
</table>