4.1 Performance
“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

Q. 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 we design must be reproducible.
- Hypothesis 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.

Andrew Appel
PU ’81
Three-Sum Problem

Three-sum problem. Given $N$ integers, find triples that sum to 0.

Context. Deeply related to problems in computational geometry.

Q. How would you write a program to solve the problem?

% 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
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

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
Q. How to time a program?
A. A stopwatch.
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;
    }
}
```

create a new stopwatch and start it running
return the elapsed time since creation, in seconds
**Stopwatch**

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

```java
public static void main(String[] args) {
    int[] a = StdArrayIO.readInt1D();
    Stopwatch timer = new Stopwatch();
    StdOut.println(count(a));
    StdOut.println(timer.elapsedTime());
}
```

**public class Stopwatch**

- Stopwatch()
  - create a new stopwatch and start it running
- double elapsedTime()
  - return the elapsed time since creation, in seconds
Empirical Analysis

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

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

Initial hypothesis. Running time obeys power law \( f(N) = a N^b \).

Data analysis. Plot running time vs. input size \( N \) on a log-log scale.

Consequence. Power law yields straight line \( \text{slope} = b \).

Refined hypothesis. Running time grows as cube of 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 $\uparrow$</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 $c = 8$

Hypothesis. Running time is about $a N^b$ with $b = \lg c$.  

Doubling Hypothesis
Doubling Challenge 1

Let $F(N)$ be running time of $\text{Mystery}$ as a function of input $N$.

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

Scenario 1. $F(2N) / F(N)$ converges to about 4.

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

Let $F(N)$ be running time of Mystery as a function of input $N$.

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

Scenario 2. $F(2N) / F(N)$ converges to about 2.

Q. What is order of growth of the running time?
Prediction and Validation

**Hypothesis.** Running time is about $aN^3$ 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>

Refined hypothesis. Running time is about $2.5 \times 10^{-10} \times N^3$ seconds.

**Prediction.** 1,100 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>

\[17.17 = a \times 4096^3 \Rightarrow a = 2.5 \times 10^{-10}\]

validates hypothesis!
Mathematical Analysis
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>

between $N$ (no zeros) and $2N$ (all zeros)
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>

\[
0 + 1 + 2 + \ldots + (N-1) = 1/2 \cdot N(N-1)
\]

becoming very tedious to count
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. $6 N^3 + 17 N^2 + 56 \sim 6 N^3$
Ex 2. $6 N^3 + 100 N^{4/3} + 56 \sim 6 N^3$
Ex 3. $6 N^3 + 17 N^2 \log N \sim 6 N^3$

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

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

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

```
{  
    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."
Constants in Power Law

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

**Exponent** \( a \) depends on: algorithm.

**Constant** \( c \) 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.

- \( \text{while (N > 1) \{ \text{N = N / 2; \ldots} \}} \) \( \lg N \)

- \( \text{lg N = \log_2 N} \)

- \( \text{for (int i = 0; i < N; i++) \ldots} \) \( N \)

- \( \text{for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) \ldots} \) \( N^2 \)

- \( \text{public static void f(int N) \{ \text{if (N == 0) return; f(N-1); f(N-1); \ldots} \}} \) \( 2^N \)

- \( \text{public static void g(int N) \{ \text{if (N == 0) return; g(N/2); g(N/2); for (int i = 0; i < N; i++) \ldots} \}} \) \( N \lg N \)
Order of Growth Classifications

<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>

Orders of growth (log-log plot)
## Order of Growth: Consequences

<table>
<thead>
<tr>
<th>order of growth</th>
<th>predicted running time if problem size is increased by a factor of 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>a few minutes</td>
</tr>
<tr>
<td>linearthmic</td>
<td>a few minutes</td>
</tr>
<tr>
<td>quadratic</td>
<td>several hours</td>
</tr>
<tr>
<td>cubic</td>
<td>a few weeks</td>
</tr>
<tr>
<td>exponential</td>
<td>forever</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>order of growth</th>
<th>predicted factor of problem size increase if computer speed is increased by a factor of 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>10</td>
</tr>
<tr>
<td>linearthmic</td>
<td>10</td>
</tr>
<tr>
<td>quadratic</td>
<td>3-4</td>
</tr>
<tr>
<td>cubic</td>
<td>2-3</td>
</tr>
<tr>
<td>exponential</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

Pascal's identity. \[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

contains first element
excludes first element
Binomial Coefficients: Sierpinski Triangle

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

Sierpinski triangle. Color black the odd integers in Pascal's triangle.
Binomial Coefficients: Poker Odds

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

Probability of "quads" in Texas hold 'em:

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

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

\[
\binom{4}{1} \times \binom{13}{6} \times \binom{3}{1} \times \binom{13}{4} \times \binom{2}{1} \times \binom{13}{2} \times \binom{1}{1} \times \binom{13}{1} \times \binom{52}{13} = \frac{29,858,811,840}{635,013,559,600} \quad (about \ 21 : 1)
\]
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));
    }
}
Timing experiments for binomial coefficients via direct recursive solution.

<table>
<thead>
<tr>
<th>((2n, n))</th>
<th>time (\dagger)</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 \(n\)?

increase \(n\) by 1, running time increases by about 4x
Doubling Challenge 3

Let $F(N)$ be running time to compute $\text{binomial}(2N, N)$.

```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);
}
```

**Observation.** $F(N+1) / F(N)$ converges to about 4.

**Q.** What is order of growth of the running time?

**A.** Exponential: $4^N$. will not finish unless $N$ is small
Why So Slow?

Function call tree.

(6, 4)

(5, 4)

(4, 4)   (4, 3)   (4, 3)

(3, 3)   (3, 2)   (3, 3)   (3, 2)

(2, 2)   (2, 1)   (2, 2)   (2, 1)

(1, 1)   (1, 0)   (1, 1)   (1, 0)

recomputed twice
**Dynamic Programming**

**Key idea.** Save solutions to subproblems to avoid recomputation.

\[
\begin{array}{cccccc}
 & & & & k & \\
 & & & & 0 & 1 & 2 & 3 & 4 \\
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
2 & 1 & 2 & 1 & 0 & 0 \\
3 & 1 & 3 & 3 & 1 & 0 \\
4 & 1 & 4 & 6 & 4 & 1 \\
5 & 1 & 5 & 10 & 10 & 5 \\
6 & 1 & 6 & 15 & 20 & 15 \\
\end{array}
\]

**binomial(n, k)**

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

\[
20 = 10 + 10
\]

**Tradeoff.** Trade (a little) memory for (a huge amount of) time.
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 for binomial coefficients via dynamic programming.

<table>
<thead>
<tr>
<th>(2n, n)</th>
<th>time</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>

Q. Is running time linear, quadratic, cubic, exponential in n?
Let $F(N)$ be running time to compute $\text{binomial}(2N, \ N)$ using DP.

\begin{verbatim}
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];
\end{verbatim}

Q. What is order of growth of the running time?

A. Quadratic: $a N^2$.  
   \hspace{1cm} \text{effectively instantaneous for small } N

Remark. There is a profound difference between $4^N$ and $N^2$. 

\hspace{1cm} \text{cannot solve a large problem} \hspace{1cm} \text{can solve a large problem}
Digression: Stirling's Approximation

Alternative: \[
\binom{n}{k} = \frac{n!}{n! (n-k)!}
\]

Caveat. 52! overflows a long, even though final result doesn't.

Stirling's approximation:

\[
\ln n! \approx n \ln n - n + \frac{\ln(2\pi n)}{2} + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5}
\]

Application. Probability of exact k heads in n flips with a biased coin.

\[
\binom{n}{k} p^k (1-p)^{n-k}
\]

(easy to compute approximate value with Stirling's formula)
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>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>byte</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>

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

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

*typical computer '08 has about 1GB memory*
Q. 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.
Q. How can I evaluate the performance of my program?
A. Computational experiments, mathematical analysis.

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>