1.4 Analysis of Algorithms

- introduction
- running time (experimental analysis)
- running time (mathematical models)
- memory usage
1.4 Analysis of Algorithms

- Introduction
- Running time (experimental analysis)
- Running time (mathematical models)
- Memory usage
Cast of characters

- **programmer** needs to develop a working solution
- **client** wants to solve problem efficiently
- **theoretician** seeks to understand
- **student (you)** might play all of these roles someday
“As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise—By what course of calculation can these results be arrived at by the machine in the shortest time?” — Charles Babbage (1864)
“As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise—By what course of calculation can these results be arrived at by the machine in the shortest time?” — Charles Babbage (1864)
An algorithmic success story

**N-body simulation.**
- Simulate gravitational interactions among $n$ bodies.
- Applications: cosmology, fluid dynamics, semiconductors, ...
- Brute force: $n^2$ steps.
- Barnes–Hut algorithm: $n \log n$ steps, enables new research.

Andrew Appel  
PU '81
The challenge

Q. Will my program be able to solve a large practical input?

Our approach. Combination of experiments and mathematical modeling.
Example: 3-SUM

**3-SUM.** Given $n$ distinct integers, how many triples sum to exactly zero?

```
~/Desktop/3sum> more 8ints.txt
8
30 -40 -20 -10 40 0 10 5
~/Desktop/3sum> java ThreeSum 8ints.txt
4
```

<table>
<thead>
<tr>
<th></th>
<th>a[i]</th>
<th>a[j]</th>
<th>a[k]</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>-40</td>
<td>10</td>
<td>0 ✓</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>-20</td>
<td>-10</td>
<td>0 ✓</td>
</tr>
<tr>
<td>3</td>
<td>-40</td>
<td>40</td>
<td>0</td>
<td>0 ✓</td>
</tr>
<tr>
<td>4</td>
<td>-10</td>
<td>0</td>
<td>10</td>
<td>0 ✓</td>
</tr>
</tbody>
</table>

**Context.** Related to problems in computational geometry.
3-SUM: brute-force algorithm

```java
public class ThreeSum {
    
    public static int count(int[] a) {
        int n = a.length;
        int count = 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) {
                        count++;
                    }
                }
            }
        }
        return count;
    }

    public static void main(String[] args) {
        In in = new In(args[0]);
        int[] a = in.readAllInts();
        StdOut.println(count(a));
    }
}
```
1.4 Analysis of Algorithms

- introduction
- running time (experimental analysis)
- running time (mathematical models)
- memory usage
Measuring the running time

Q. How to time a program?
A. Manual.
Measuring the running time

Q. How to time a program?
A. Automatic.

```java
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.Stopwatch;

public static void main(String[] args)
{
    In in = new In(args[0]);
    int[] a = in.readInts();
    Stopwatch stopwatch = new Stopwatch();
    StdOut.println(ThreeSum.count(a));
    double time = stopwatch.elapsedTime();
    StdOut.println("elapsed time = " + time);
}
```
Empirical analysis

Run the program for various input sizes and measure running time.
Empirical analysis

Run the program for various input sizes and measure running time.

<table>
<thead>
<tr>
<th>n</th>
<th>time (seconds) †</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td>1,000</td>
<td>0.1</td>
</tr>
<tr>
<td>2,000</td>
<td>0.8</td>
</tr>
<tr>
<td>4,000</td>
<td>6.4</td>
</tr>
<tr>
<td>8,000</td>
<td>51.1</td>
</tr>
<tr>
<td>16,000</td>
<td>?</td>
</tr>
</tbody>
</table>

† on a 2.8GHz Intel PU-226 with 64GB DDR E3 memory and 32MB L3 cache; running Oracle Java 1.7.0_45-b18 on Springdale Linux v. 6.5
Data analysis

**Standard plot.** Plot running time $T(n)$ vs. input size $n$.

![Graph showing running time vs. problem size]

**Hypothesis (power law).** $T(n) = a \cdot n^b$.

**Questions.** How to validate hypothesis? How to estimate $a$ and $b$?
Data analysis

**Log–log plot.** Plot running time $T(n)$ vs. input size $n$ using log–log scale.

Regression. Fit straight line through data points.

Hypothesis. The running time is about $1.006 \times 10^{-10} \times n^{2.999}$ seconds.
Prediction and validation

Hypothesis. The running time is about $1.006 \times 10^{-10} \times n^{2.999}$ seconds.

Predictions.
- 51.0 seconds for $n = 8,000$.
- 408.1 seconds for $n = 16,000$.

Observations.

<table>
<thead>
<tr>
<th>n</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,000</td>
<td>51.1</td>
</tr>
<tr>
<td>8,000</td>
<td>51.0</td>
</tr>
<tr>
<td>8,000</td>
<td>51.1</td>
</tr>
<tr>
<td>16,000</td>
<td>410.8</td>
</tr>
</tbody>
</table>

"order of growth" of running time is about $n^3$
[stay tuned]

validates hypothesis!
Doubling hypothesis. Quick way to estimate exponent \( b \) in a power-law relationship.

Run program, **doubling** the size of the input.

<table>
<thead>
<tr>
<th>( n )</th>
<th>time (seconds) ( \dagger )</th>
<th>ratio</th>
<th>( \log_2 ) ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>0</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>4.8</td>
<td>2.3</td>
</tr>
<tr>
<td>1,000</td>
<td>0.1</td>
<td>6.9</td>
<td>2.8</td>
</tr>
<tr>
<td>2,000</td>
<td>0.8</td>
<td>7.7</td>
<td>2.9</td>
</tr>
<tr>
<td>4,000</td>
<td>6.4</td>
<td>8</td>
<td>3.0</td>
</tr>
<tr>
<td>8,000</td>
<td>51.1</td>
<td>8</td>
<td>3.0</td>
</tr>
</tbody>
</table>

\[
\frac{T(n)}{T(n/2)} = \frac{an^b}{a(n/2)^b} = 2^b
\]

\[
\Rightarrow b = \log_2 \frac{T(n)}{T(n/2)}
\]

\( \log_2 (6.4/0.8) = 3.0 \)

seems to converge to a constant \( b \approx 3 \)

**Hypothesis.** Running time is about \( an^b \) with \( b = \log_2 \) ratio.

**Caveat.** Cannot identify logarithmic factors with doubling hypothesis.
Doubling hypothesis

**Doubling hypothesis.** Quick way to estimate exponent $b$ in a power-law relationship.

**Q.** How to estimate $a$ (assuming we know $b$)?

**A.** Run the program (for a sufficient large value of $n$) and solve for $a$.

<table>
<thead>
<tr>
<th>n</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,000</td>
<td>51.1</td>
</tr>
<tr>
<td>8,000</td>
<td>51.0</td>
</tr>
<tr>
<td>8,000</td>
<td>51.1</td>
</tr>
</tbody>
</table>

$51.1 = a \times 8000^3$

$\Rightarrow a = 0.998 \times 10^{-10}$

**Hypothesis.** Running time is about $0.998 \times 10^{-10} \times n^3$ seconds.
Estimate the running time to solve a problem of size $n = 96,000$.

A. 39 seconds
B. 52 seconds
C. 117 seconds
D. 350 seconds

<table>
<thead>
<tr>
<th>n</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.02</td>
</tr>
<tr>
<td>2,000</td>
<td>0.05</td>
</tr>
<tr>
<td>4,000</td>
<td>0.20</td>
</tr>
<tr>
<td>8,000</td>
<td>0.81</td>
</tr>
<tr>
<td>16,000</td>
<td>3.25</td>
</tr>
<tr>
<td>32,000</td>
<td>13.01</td>
</tr>
</tbody>
</table>
Experimental algorithmics

System independent effects.

- Algorithm.
- Input data.

\[
\begin{align*}
\text{determines exponent } b \\
in \text{power law } a n^b
\end{align*}
\]

System dependent effects.

- Hardware: CPU, memory, cache, ...
- Software: compiler, interpreter, garbage collector, ...
- System: operating system, network, other apps, ...

\[
\begin{align*}
\text{determines constant } a \\
in \text{power law } a n^b
\end{align*}
\]

Bad news. Sometimes difficult to get accurate measurements.
Context: the scientific method

Experimental algorithmics is an example of the scientific method.

Good news. Experiments are easier and cheaper than other sciences.
1.4 Analysis of Algorithms

- introduction
- running time (experimental analysis)
- running time (mathematical models)
- memory usage
Mathematical models for running time

**Total running time**: sum of cost × frequency for all operations.

- Need to analyze program to determine set of operations.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

**Warning**: No general-purpose method (e.g., halting problem).
Example: 1-SUM

Q. How many operations as a function of input size $n$?

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

- exactly $n$ array accesses

<table>
<thead>
<tr>
<th>operation</th>
<th>cost (ns) †</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>2/5</td>
<td>2</td>
</tr>
<tr>
<td>assignment statement</td>
<td>1/5</td>
<td>2</td>
</tr>
<tr>
<td>less than compare</td>
<td>1/5</td>
<td>$n + 1$</td>
</tr>
<tr>
<td>equal to compare</td>
<td>1/10</td>
<td>$n$</td>
</tr>
<tr>
<td>array access</td>
<td>1/10</td>
<td>$n$</td>
</tr>
<tr>
<td>increment</td>
<td>1/10</td>
<td>$n$ to $2n$</td>
</tr>
</tbody>
</table>

† representative estimates (with some poetic license)

In practice, depends on caching, bounds checking, ... (see COS 217)
Analysis of algorithms: quiz 2

How many array accesses as a function of $n$?

```c
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++;
```

A. $\frac{1}{2} n (n - 1)$
B. $n (n - 1)$
C. $2 n^2$
D. $2 n (n - 1)$
Example: 2-SUM

Q. How many operations as a function of input size $n$?

```c
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++;
```

$$0 + 1 + 2 + \ldots + (n-1) = \frac{1}{2} n(n-1)$$

$$= \binom{n}{2}$$

<table>
<thead>
<tr>
<th>operation</th>
<th>cost (ns)</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>2/5</td>
<td>$n + 2$</td>
</tr>
<tr>
<td>assignment statement</td>
<td>1/5</td>
<td>$n + 2$</td>
</tr>
<tr>
<td>less than compare</td>
<td>1/5</td>
<td>$\frac{1}{2} (n + 1) (n + 2)$</td>
</tr>
<tr>
<td>equal to compare</td>
<td>1/10</td>
<td>$\frac{1}{2} n (n - 1)$</td>
</tr>
<tr>
<td>array access</td>
<td>1/10</td>
<td>$n (n - 1)$</td>
</tr>
<tr>
<td>increment</td>
<td>1/10</td>
<td>$\frac{1}{2} n (n + 1)$ to $n^2$</td>
</tr>
</tbody>
</table>
Simplification 1: cost model

Cost model. Use some elementary operation as a proxy for running 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>cost (ns)</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>2/5</td>
<td>n + 2</td>
</tr>
<tr>
<td>assignment statement</td>
<td>1/5</td>
<td>n + 2</td>
</tr>
<tr>
<td>less than compare</td>
<td>1/5</td>
<td>(\frac{1}{2} (n + 1) (n + 2))</td>
</tr>
<tr>
<td>equal to compare</td>
<td>1/10</td>
<td>(\frac{1}{2} n (n - 1))</td>
</tr>
<tr>
<td>array access</td>
<td>1/10</td>
<td>(n (n - 1))</td>
</tr>
<tr>
<td>increment</td>
<td>1/10</td>
<td>(\frac{1}{2} n (n + 1)) to (n^2)</td>
</tr>
</tbody>
</table>

(cost model = array accesses)

(we're assuming compiler/JVM does not optimize any array accesses away!)
Simplification 2: asymptotic notations

**Tilde notation.** Discard lower-order terms.

**Big Theta notation.** Also discard leading coefficient.

<table>
<thead>
<tr>
<th>function</th>
<th>tilde</th>
<th>big Theta</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4n^5 + 20n + 16$</td>
<td>$\sim 4n^5$</td>
<td>$\Theta(n^5)$</td>
</tr>
<tr>
<td>$7n^2 + 100n^{4/3} + 56$</td>
<td>$\sim 7n^2$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>$\frac{1}{6}n^3 - \frac{1}{2}n^2 + \frac{1}{2}n$</td>
<td>$\sim \frac{1}{6}n^3$</td>
<td>$\Theta(n^3)$</td>
</tr>
</tbody>
</table>

**Rationale.**

- When $n$ is large, lower-order terms are negligible.
- When $n$ is small, we don’t care.
## Common order-of-growth classifications

<table>
<thead>
<tr>
<th>Order of growth</th>
<th>Name</th>
<th>Typical code framework</th>
<th>Description</th>
<th>Example</th>
<th>(T(2n) / T(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Theta(1))</td>
<td>constant</td>
<td>(a = b + c;)</td>
<td>statement</td>
<td>add two numbers</td>
<td>1</td>
</tr>
</tbody>
</table>
| \(\Theta(\log n)\) | logarithmic | while \((n > 1)\)  
{ \(n = n/2;\) ... } | divide in half    | binary search                       | \(~ 1\)          |
| \(\Theta(n)\)   | linear     | for (int \(i = 0; i < n; i++\))  
{ ... } | single loop       | find the maximum                    | 2                |
| \(\Theta(n \log n)\) | linearithmic | see mergesort lecture                                       | divide and conquer | mergesort                           | \(~ 2\)          |
| \(\Theta(n^2)\)  | quadratic  | for (int \(i = 0; i < n; i++\))  
for (int \(j = 0; j < n; j++\))  
{ ... } | double loop       | check all pairs                    | 4                |
| \(\Theta(n^3)\)  | cubic      | for (int \(i = 0; i < n; i++\))  
for (int \(j = 0; j < n; j++\))  
for (int \(k = 0; k < n; k++\))  
{ ... } | triple loop       | check all triples                  | 8                |
| \(\Theta(2^n)\)  | exponential| see combinatorial search lecture                             | exhaustive search | check all subsets                    | \(2^n\)          |
Example: 2-SUM

Q. Approximately how many array accesses as a function of input size $n$?

```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++;
```

"inner loop"

$$0 + 1 + 2 + \ldots + (n-1) = \frac{1}{2} n(n-1)$$

A. $\sim n^2$ array accesses.
Example: 3-SUM

Q. **Approximately how many array accesses** as a function of input size \( n \)?

```
int count = 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)
                count++;
```

A. \( \sim \frac{1}{2} n^3 \) array accesses.

\[
{n \choose 3} = \frac{n(n-1)(n-2)}{3!} \approx \frac{1}{6} n^3
\]

**Bottom line.** Use cost model and asymptotic notation to simplify analysis.
Estimating a discrete sum

Q. How to estimate a discrete sum?
A1. Take a discrete mathematics course (COS 340).
Estimating a discrete sum

Q. How to estimate a discrete sum?

A2. Replace the sum with an integral; use calculus!

Ex 1. \(1 + 2 + \ldots + n.\)

(triangular sum)

\[
\sum_{i=1}^{n} i \sim \int_{x=1}^{n} x \, dx \sim \frac{1}{2} n^2
\]

Ex 2. \(1 + 1/2 + 1/3 + \ldots + 1/n.\)

(harmonic sum)

\[
\sum_{i=1}^{n} \frac{1}{i} \sim \int_{x=1}^{n} \frac{1}{x} \, dx \sim \ln n
\]

Ex 3. 3-sum triple loop.

\[
\sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} 1 \sim \int_{x=1}^{n} \int_{y=x+1}^{n} \int_{z=y+1}^{n} dz \, dy \, dx \sim \frac{1}{6} n^3
\]

Ex 4. \(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\)

(geometric sum)

\[
\int_{x=0}^{\infty} \left(\frac{1}{2}\right)^x \, dx = \frac{1}{\ln 2} \approx 1.4427
\]

\[
\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = 2
\]

integral trick doesn't always work!
Estimating a discrete sum

Q. How to estimate a discrete sum?
A3. Use Maple or Wolfram Alpha.
Analysis of algorithms: quiz 3

How many array accesses as a function of $n$?

```java
int count = 0;
for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
        for (int k = 1; k <= n; k = k*2)
            if (a[i] + a[j] >= a[k])
                count++;
```

A. $\sim n^2 \log_2 n$
B. $\sim 3/2 \; n^2 \log_2 n$
C. $\sim 1/2 \; n^3$
D. $\sim 3/2 \; n^3$
What is order of growth of running time as a function of \( n \)?

```java
int count = 0;
for (int i = n; i >= 1; i = i/2)
    for (int j = 1; j <= i; j++)
        count++;
```

A. \( \Theta(n) \)
B. \( \Theta(n \log n) \)
C. \( \Theta(n^2) \)
D. \( \Theta(2^n) \)
1.4 ANALYSIS OF ALGORITHMS

- introduction
- running time (experimental analysis)
- running time (mathematical models)
- memory usage
Basics

**Bit.** 0 or 1.  
**Byte.** 8 bits.  
**Megabyte (MB).** 1 million or $2^{20}$ bytes.  
**Gigabyte (GB).** 1 billion or $2^{30}$ bytes.

**64-bit machine.** We assume a 64-bit machine with 8-byte pointers.

NIST  
most computer scientists

some JVMs “compress” ordinary object pointers to 4 bytes to avoid this cost
**Typical memory usage for primitive types and arrays**

<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>

**primitive types**

<table>
<thead>
<tr>
<th>type[]</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean[]</td>
<td>(1n + 24)</td>
</tr>
<tr>
<td>int[]</td>
<td>(4n + 24)</td>
</tr>
<tr>
<td>double[]</td>
<td>(8n + 24)</td>
</tr>
</tbody>
</table>

one-dimensional arrays (length n)

- wasteful (but \(\sim 36n\) in Python 3)
- array overhead = 24 byte

<table>
<thead>
<tr>
<th>type[][]</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean[][]</td>
<td>(\sim 1n^2)</td>
</tr>
<tr>
<td>int[][]</td>
<td>(\sim 4n^2)</td>
</tr>
<tr>
<td>double[][]</td>
<td>(\sim 8n^2)</td>
</tr>
</tbody>
</table>

two-dimensional arrays (n-by-n)
Typical memory usage for objects in Java

Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Memory of each object rounded up to use a multiple of 8 bytes.

Ex 1. A Date object uses 32 bytes of memory.
Total memory usage for a data type value in Java:

- Primitive type: 4 bytes for int, 8 bytes for double, ...
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable.
- Padding: round up memory of each object to be a multiple of 8 bytes.

**Note.** Depending on application, we often count the memory for any referenced objects (recursively).
Analysis of algorithms: quiz 5

How much memory does a `WeightedQuickUnionUF` use as a function of \( n \)?

A. \( \sim 4n \) bytes

B. \( \sim 8n \) bytes

C. \( \sim 4n^2 \) bytes

D. \( \sim 8n^2 \) bytes

```java
public class WeightedQuickUnionUF {
    private int[] parent;
    private int[] size;
    private int count;

    public WeightedQuickUnionUF(int n) {
        parent = new int[n];
        size = new int[n];

        count = 0;
        for (int i = 0; i < n; i++)
            parent[i] = i;
        for (int i = 0; i < n; i++)
            size[i] = 1;
    }

    ...
}
```
Turning the crank: summary

Empirical analysis.
- Execute program to perform experiments.
- Assume power law.
- Formulate a hypothesis for running time.
- Model enables us to make predictions.

Mathematical analysis.
- Analyze algorithm to count frequency of operations.
- Use tilde and big-Theta notations to simplify analysis.
- Model enables us to explain behavior.

This course. Learn to use both.