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
Different viewpoints

programmer needs to develop a working solution

client wants to solve problem efficiently

theoretician seeks to understand

student (you) will play all of these roles in this course
“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)

Ada Lovelace’s algorithm to compute Bernoulli numbers on Analytic Engine (1843)
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
```

<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. Connected with problems in computational geometry.
```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));
    }
}
```

3-SUM: brute-force algorithm

- check distinct triples
- for simplicity, ignore integer overflow
1.4 Analysis of Algorithms

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

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

**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.
Doubling 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) †</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>

\[
\frac{T(n)}{T(n/2)} = \frac{a n^b}{a(n/2)^b} = 2^b \\
\implies b = \log_2 \frac{T(n)}{T(n/2)}
\]

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

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

Hypothesis. Running time is about \( T(n) = a n^b \), with \( b = \log_2 \) ratio.
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) $\dagger$</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.

almost identical hypothesis to one obtained via regression (but less work)
Estimate the running time to solve a problem of size $n = 96,000$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>39 seconds</td>
</tr>
<tr>
<td>B.</td>
<td>52 seconds</td>
</tr>
<tr>
<td>C.</td>
<td>117 seconds</td>
</tr>
<tr>
<td>D.</td>
<td>350 seconds</td>
</tr>
</tbody>
</table>

<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.

\[
\text{determines exponent } b \\
\text{in power law } a \cdot n^b
\]

System dependent effects.

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

\[
\text{determines constant } a \\
\text{in power law } a \cdot n^b
\]

Bad news. Sometimes difficult to get accurate measurements.
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 $\times$ frequency for all operations.

- Need to analyze program to determine set of operations.
- Frequency depends on algorithm and input data.
- Cost depends on CPU, compiler, operating system, ....

**Warning**: No general-purpose method (e.g., halting problem).
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++;
```

Example: 1-SUM

<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?

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

```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++;
```
Example: 2-SUM

Q. How many operations 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++;
```

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

\[
= \binom{n}{2}
\]

<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) \text{ to } n^2 )</td>
</tr>
</tbody>
</table>

\( \frac{1}{4} n^2 + 13/20 n + 13/10 \) ns

\( \frac{3}{10} n^2 + 3/5 n + 13/10 \) ns

(tedious to count exactly)
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>

Array accesses, compares, floating-point operations, disk accesses, API calls, ...

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>$4 n^5 + 20 n + 16$</td>
<td>$\sim 4 n^5$</td>
<td>$\Theta(n^5)$</td>
</tr>
<tr>
<td>$7 n^2 + 100 n^{4/3} + 56$</td>
<td>$\sim 7 n^2$</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>emoji</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>😍</td>
<td>constant</td>
<td><code>a = b + c;</code></td>
<td>statement</td>
<td>add two numbers</td>
<td>1</td>
</tr>
</tbody>
</table>
| $\Theta(\log n)$| 😎    | logarithmic| `for (int i = n; i > 0; i /= 2)`
  `{ ... }`          | divide in half | binary search           | $\sim 1$    |
| $\Theta(n)$     | 😊    | 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                | $\sim 2$      |
| $\Theta(n^2)$   | 😞    | 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++;
```

A. \( \sim n^2 \) array accesses.

\[
0 + 1 + 2 + \ldots + (n-1) = \frac{1}{2} n(n-1) = \binom{n}{2}
\]
Example: 3-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++)
        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.

$$\binom{n}{3} = \frac{n(n-1)(n-2)}{3!} \approx \frac{1}{6} n^3$$

*Bottom line.* Use **cost model** and **asymptotic notation** to simplify analysis.
Some useful discrete sums and approximations

**Triangular sum.** \[ 1 + 2 + 3 + \ldots + n \sim \frac{1}{2} n^2 \]

**Harmonic sum.** \[ 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \sim \int_{x=1}^{n} \frac{1}{x} \, dx = \ln n \]

**Geometric sum.** \[ 1 + 2 + 4 + 8 + \ldots + n = 2n - 1 \]

$n$ a power of $2$
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. $\sim \frac{3}{2} n^2 \log_2 n$

C. $\sim \frac{1}{2} n^3$

D. $\sim \frac{3}{2} n^3$
What is order of growth of running time as a function of $n$?

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

https://algs4.cs.princeton.edu
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)**

- Array overhead = 24 bytes
- Wasteful (but \(\sim 36n\) in Python 3)

<table>
<thead>
<tr>
<th>type</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean[][]</td>
<td>(\sim 1 n^2)</td>
</tr>
<tr>
<td>int[][]</td>
<td>(\sim 4 n^2)</td>
</tr>
<tr>
<td>double[][]</td>
<td>(\sim 8 n^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. Round up memory of each object to be a multiple of 8 bytes.

Ex 1. Each Date object uses 32 bytes of memory.

```java
public class Date {
    private int day;
    private int month;
    private int year;
    ...
}
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>4 bytes (int)</td>
</tr>
<tr>
<td>month</td>
<td>4 bytes (int)</td>
</tr>
<tr>
<td>year</td>
<td>4 bytes (int)</td>
</tr>
<tr>
<td>padding</td>
<td>4 bytes (padding)</td>
</tr>
<tr>
<td>total</td>
<td>32 bytes</td>
</tr>
</tbody>
</table>

16 bytes (object overhead)
Analysis of algorithms: quiz 5

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

A. \( \sim 4 \ n \ \text{bytes} \)
B. \( \sim 8 \ n \ \text{bytes} \)
C. \( \sim 4 \ n^2 \ \text{bytes} \)
D. \( \sim 8 \ n^2 \ \text{bytes} \)

```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.

\[ \sum_{h=0}^{\lfloor \log n \rfloor} \left\lfloor \frac{n}{2^h + 1} \right\rfloor h \sim n \]

This course. Learn to use both.
"It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then given them various weights.” — Alan Turing (1947)