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

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

Key insight. [Knuth 1970s]
Use the scientific method to understand performance.

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.

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.
- Hypothesis must be falsifiable.

Algorithmic Successes

Sorting
- Rearrange array of $N$ items in ascending order.
- Applications: databases, scheduling, statistics, genomics, ...
- Brute force: $N^2$ steps.
- Mergesort: $N \log N$ steps, enables new technology.

Amazon.com

Google

Ebay
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.

Freidrich Gauss 1805

Algorithmic Successes

N-body Simulation.
- Simulate gravitational interactions among \( N \) bodies.
- Applications: cosmology, semiconductors, fluid dynamics, ...
- Brute force: \( N^2 \) steps.
- Barnes-Hut algorithm: \( N \log N \) steps, enables new research.

Three-Sum Problem

Three-sum problem. Given \( N \) integers, how many triples sum to 0?

Context. Deeply related to problems in computational geometry.

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

Empirical Analysis

Empirical analysis. Run the program for various input sizes.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \text{time} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.03</td>
</tr>
<tr>
<td>1,024</td>
<td>0.26</td>
</tr>
<tr>
<td>2,048</td>
<td>2.16</td>
</tr>
<tr>
<td>4,096</td>
<td>17.18</td>
</tr>
<tr>
<td>8,192</td>
<td>136.76</td>
</tr>
</tbody>
</table>

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

Caveat. If \( N \) is too small, you will measure mainly noise.
Stopwatch

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

```
% java ThreeSum < 2Kints.txt
% java ThreeSum < 2Kints.txt
```

```
319350676 -763182495 371251819
-326747290 802433422 -475684132
```

Stopwatch

Q. How to time a program?
A. System.currentTimeMillis() function

```
public static void main(String[] args) {
  int[] a = StdArrayIO.readInt1D();
  long start = System.currentTimeMillis();
  StdOut.println(count(a));
  StdOut.println((System.currentTimeMillis() - start) / 1000.0);
}
```

Empirical Analysis

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

![Graph showing data analysis](image)

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

Empirical Analysis

Initial hypothesis. Running time approximately obeys a power law $T(N) = a N^b$.

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

Consequence. Power law yields straight line.

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 $^1$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.033</td>
<td>-</td>
</tr>
<tr>
<td>1,024</td>
<td>0.26</td>
<td>7.88</td>
</tr>
<tr>
<td>2,048</td>
<td>2.16</td>
<td>8.43</td>
</tr>
<tr>
<td>4,096</td>
<td>17.18</td>
<td>7.96</td>
</tr>
<tr>
<td>8,192</td>
<td>136.76</td>
<td>7.96</td>
</tr>
</tbody>
</table>

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

Performance Challenge 1

Let $T(N)$ be running time of `main()` as a function of input size $N$.

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

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

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

1. $N^N$, $N^2$, $N^3$, $N^4$, $2^N$
Performance Challenge 2

Let $T(N)$ be running time of `main()` as a function of input size $N$.

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

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

1. $N$  
2. $N^2$  
3. $N^3$  
4. $2^N$

Mathematical Analysis

Donald Knuth

Turing award ’74

Prediction and Validation

Hypothesis: Running time is about $a N^3$ for input of size $N$.

Q. How to estimate $a$?

A. Run the program!

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

Prediction. 1,100 seconds for $N = 16,384$.

Observation. 17.17 seconds for $N = 16,384$.

Mathematical Analysis

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

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>$N$+1</th>
<th>$N$+2</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable declaration</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable assignment</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>less than comparison</td>
<td>$N+1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal to comparison</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>array access</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>increment</td>
<td>$&lt;2N$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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. $6N^3 + 100N^4 + 56 \sim 6N^3$

Ex 3. $6N^3 + 17N^2 \log N \sim 6N^3$

Technical definition. $f(N) \sim g(N)$ means $\lim_{n \to \infty} \frac{f(N)}{g(N)} = 1$ for large $N$. (if work)

discard lower order terms (e.g., $N$: 1,000; $N^3$: 1,000,000; ... if work)}
Mathematical Analysis

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

```
public static int count(int[] a) {
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int k = j = 0; 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 $\sim aN^b$.

Exponent $b$ depends on:
- Algorithm.
- Input data.
- Caching.
- Machine.
- Compiler.
- Garbage collection.
- Just-in-time compilation.
- CPU use by other applications.

Leading 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

<table>
<thead>
<tr>
<th>order of growth description</th>
<th>function</th>
<th>factor for doubling problem size</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>

Commonly encountered growth functions

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>linearithmic</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 if computer speed is increased by a factor of 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>10</td>
</tr>
<tr>
<td>linearithmic</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
**Binary Search**

Sequential Search vs. Binary Search

- **Sequential search in an unordered array.**
  - Examine each entry until finding a match (or reaching the end).
  - Takes time proportional to length of array in worst case.

- **Binary search in an ordered array.**
  - Examine the middle entry.
  - If equal, return index.
  - If too large, search in left half (recursively).
  - If too small, search in right half (recursively).

**Binary Search: Java Implementation**

**Invariant.** If key appears in the array, then a[lo] ≤ key ≤ a[hi].

```java
// precondition: array a[] is sorted
public static int search(int key, int[] a) {
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid; // not found
    }
    return -1; // not found
}
```

**Java library implementation.** Arrays.binarySearch().

**Binary Search: Mathematical Analysis**

**Proposition.** Binary search in an ordered array of size $N$ takes at most $1 + \log_2 N$ 3-way compares.

**Pf.** After each 3-way compare, problem size decreases by a factor of 2.

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \ldots \rightarrow 1$

**Q.** How many times can you divide $N$ by 2 until you reach 1?

**A.** About $\log_2 N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>\rightarrow N/2</th>
<th>\rightarrow N/4</th>
<th>\rightarrow N/8</th>
<th>\rightarrow 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>128</td>
<td>64</td>
</tr>
</tbody>
</table>

**Searching Challenge 1**

**Q.** A credit card company needs to whitelist 100 million customer account numbers, processing 10,000 transactions per second.

Using sequential search, what kind of computer is needed?

- A. Toaster.
- B. Cell phone.
- C. Your laptop.
- D. Supercomputer.
- E. Google server farm.

**Searching Challenge 2**

**Q.** A credit card company needs to whitelist 100 million customer account numbers, processing 10,000 transactions per second.

Using binary search, what kind of computer is needed?

- A. Toaster.
- B. Cell phone.
- C. Your laptop.
- D. Supercomputer.
- E. Google server farm.