2.1 Computational Tractability

Charles Babbage (1864)
Analytic Engine (schematic)

Polynomial-Time

- Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.
  - Typically takes $2^N$ time or worse for inputs of size $N$.
  - Unacceptable in practice.

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor $C$.

Worst-Case Analysis

- Worst-case running time. Obtain bound on largest possible running time of algorithm on input of a given size $N$.
  - Generally captures efficiency in practice.
  - Draconian view, but hard to find effective alternative.

Average-case running time. Obtain bound on running time of algorithm on random input as a function of input size $N$.

- Hard (or impossible) to accurately model real instances by random distribution.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Worst-Case Polynomial-Time

- Def. An algorithm is efficient if its running time is polynomial.

Justification: It really works in practice.
- Although $6.02 \times 10^{23} = N!$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.
- Some polynomial-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

- simplex method
- Unix grep
### 2.2 Asymptotic Order of Growth

**Upper bounds.** \( T(n) = O(f(n)) \) if there exist constants \( c > 0 \) and \( n_0 \geq 0 \) such that for all \( n \geq n_0 \) we have \( T(n) \leq c \cdot f(n) \).

**Lower bounds.** \( T(n) = \Omega(f(n)) \) if there exist constants \( c > 0 \) and \( n_0 \geq 0 \) such that for all \( n \geq n_0 \) we have \( T(n) \geq c \cdot f(n) \).

**Tight bounds.** \( T(n) = \Theta(f(n)) \) if \( T(n) \) is both \( O(f(n)) \) and \( \Omega(f(n)) \).

**Ex:** \( T(n) = 32n^2 + 17n + 32 \).
- \( T(n) \) is \( O(n^2) \), \( O(n^3) \), \( \Omega(n^2) \), \( \Omega(n) \), and \( \Theta(n^2) \).
- \( T(n) \) is not \( O(n) \), \( \Omega(n^3) \), \( \Theta(n) \), or \( \Theta(n^3) \).

### Notation

**Slight abuse of notation.** \( T(n) = O(f(n)) \).
- Not transitive: 
  - \( f(n) = 5n^3; g(n) = 3n^2 \)
  - \( f(n) = O(n^3) = g(n) \)
  - But \( f(n) \neq g(n) \).
- Better notation: \( T(n) \in O(f(n)) \).

**Meaningless statement.** Any comparison-based sorting algorithm requires at least \( O(n \log n) \) comparisons.
- Statement doesn’t “type check.”
- Use \( \Omega \) for lower bounds.

### Big-\( \Theta \) and limits

**LEMMA:** If the limit as \( x \to \infty \) of the quotient \( |f(x)/g(x)| \) is a constant then \( f(x) = \Theta(g(x)) \).

**EG:** \( 3x^3 + 5x^2 - 9 = \Theta(x^3) \). Compute:

\[
\lim_{x \to \infty} \frac{3x^3 + 5x^2 - 9}{x^3} = \lim_{x \to \infty} \frac{3 + 5/x - 9/x^2}{1} = 3
\]

### Little-\( o \) and limits

**DEF:** If the limit as \( x \to \infty \) of the quotient \( |f(x)/g(x)| \) = 0 then \( f(x) = o(g(x)) \).

**EG:** \( 3x^3 + 5x^2 - 9 = o(x^{3.1}) \). Compute:

\[
\lim_{x \to \infty} \frac{3x^3 + 5x^2 - 9}{x^{3.1}} = \lim_{x \to \infty} \frac{3/x^{0.1} + 5/x^{1.1} - 9/x^{1.1}}{1} = 0
\]

### Big-\( \Omega \) and Big-\( O \)

**Big-\( O \):** \( f = O(g) \) if \( f = \Theta(g) \) or \( f = o(g) \)

**Big-\( \Omega \):** reverse of big-\( O \)

\( f(x) = \Omega(g(x)) \iff g(x) = O(f(x)) \)

**small-\( o \):** reverse of small-\( o \)

\( f(x) = o(g(x)) \iff g(x) = \omega(f(x)) \)

**Big-\( \Theta \):** domination in both directions. I.e.

\( f(x) = \Theta(g(x)) \iff f(x) = \Omega(g(x)) \iff f(x) = O(g(x)) \)

### Big-\( \Omega \) and Big-\( O \)

**Big-\( O \):** \( f = O(g) \) if \( f = \Theta(g) \) or \( f = o(g) \)

**Big-\( \Omega \):** reverse of big-\( O \)

\( f(x) = \Omega(g(x)) \iff g(x) = O(f(x)) \)

**small-\( o \):** reverse of small-\( o \)

\( f(x) = o(g(x)) \iff g(x) = \omega(f(x)) \)

**Big-\( \Theta \):** domination in both directions. I.e.

\( f(x) = \Theta(g(x)) \iff f(x) = \Omega(g(x)) \iff f(x) = O(g(x)) \)
Useful facts

Any polynomial is $\Theta$ of its largest term.
EG: $x^4/100000 + 3x^3 + 5x^2 - 9 = \Theta(x^4)$

The sum of two functions is $\Theta$ of the biggest.
EG: $x^4 \ln(x) + x^5 = \Theta(x^5)$

Non-zero constants are irrelevant.
EG: $17x^4 \ln(x) = \Theta(x^4 \ln(x))$

Properties

Transitivity.
- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additivity.
- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.

Asymptotic Bounds for Some Common Functions

Polynomials: $a_0 + a_1n + \ldots + a_dn^d$ is $\Theta(n^d)$ if $a_d > 0$.

Polynomial time. Running time is $O(nd)$ for some constant $d$ independent of the input size $n$.

Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$.

Logarithms. For every $x > 0$, $\log(n) = O(nx)$.

Exponentials. For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$.

Complexity

Comparison: time complexity of algorithms A and B

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>5,000n</td>
<td>11^n</td>
</tr>
<tr>
<td>10</td>
<td>50,000</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>500,000</td>
<td>13,781</td>
</tr>
<tr>
<td>1,000</td>
<td>5,000,000</td>
<td>2.5 \times 10^4</td>
</tr>
<tr>
<td>1,000,000</td>
<td>5 \times 10^7</td>
<td>4.8 \times 10^8</td>
</tr>
</tbody>
</table>

Travelling Salesman Problem Joke

BRUTE-FORCE SOLUTION: $O(n!)$

DYNAMIC PROGRAMMING ALGORITHM: $O(n^2!)$

SELLING ON EBay: $O(1)$

Still working on your route?

Shut the hell up!