Algorithm Analysis

Reading: Sections 2.1, 2.2 and 2.4
Algorithm Design

- Formulate the problem precisely
- Design an algorithm to solve the problem
- Prove the algorithm is correct
- Analyze the algorithm’s running time
Running Time Analysis

- *Mathematical* analysis of *worst-case* running time of an algorithm as *function of input size*
  - Mathematical: focus on the algorithm. Avoids hard-to-control experimental factors (CPU, programming language, quality of implementation)
  - Function of input size: allows predictions.
Types of Analysis

- Worst case: running time guarantee for any input of size $n$
- Average-case: Expected running time for a random input
- Probabilistic: Expected running time of an randomized algorithm
What is Efficiency?

- Better than brute force

- Scaling property: when input size increases by a factor of 2, the algorithm should only slow down by some constant factor $c$.

- An algorithm is *efficient* when it has a *polynomial* running time
  - Almost all practically efficient algorithms have this property
  - Usually distinguishes a clever algorithm from a “brute force” approach
  - Refutable: gives us a way of saying an algorithm is not efficient, or that no efficient algorithm exists.
Comparing Some Numbers

<table>
<thead>
<tr>
<th>n</th>
<th>$n$</th>
<th>$n \log_2 n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$1.5^n$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 sec</td>
</tr>
<tr>
<td>30</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>$10^{25}$ years</td>
</tr>
<tr>
<td>50</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>11 min</td>
<td>36 years</td>
<td>very long</td>
</tr>
<tr>
<td>100</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>12,892 years</td>
<td>$10^{17}$ years</td>
<td>very long</td>
</tr>
<tr>
<td>1,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>18 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>10,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>2 min</td>
<td>12 days</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>100,000</td>
<td>&lt; 1 sec</td>
<td>2 sec</td>
<td>3 hours</td>
<td>32 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1 sec</td>
<td>20 sec</td>
<td>12 days</td>
<td>31,710 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
</tbody>
</table>

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds $10^{25}$ years, we simply record the algorithm as taking a very long time.
Asymptotic Order of Growth

- **Upper bounds.** $T(n)$ is $O(f(n))$ if there exists constants $c, n_0 > 0$ such that for all $n > n_0$ we have $T(n) \leq c \cdot f(n)$

- **Lower bounds.** $T(n)$ is $\Omega(f(n))$ if there exists constants $c, n_0 > 0$ such that for all $n > n_0$ we have $T(n) \geq c \cdot f(n)$

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

- $O(f(n))$ is a set of functions
\texttt{sum} = 0 \\
\textbf{for} i= 1 \textbf{to} n \textbf{do} \\
\hspace{7mm} \textbf{for} j= 1 \textbf{to} n \textbf{do} \\
\hspace{14mm} \text{sum} += A[i]*A[j] \\
\hspace{7mm} \textbf{end for} \\
\textbf{end for} \\

- What is the running time? How many “primitive steps” are executed?
Is This Analysis Correct?

Algorithm foo

\[
\begin{align*}
\text{for } i &= 1 \text{ to } n \text{ do} \\
\quad &\text{for } j = 1 \text{ to } n \text{ do} \\
\quad &\quad x += 1 \\
\quad &\text{end for} \\
\text{end for}
\end{align*}
\]

- Fact: run time is $O(n^3)$

Algorithm bar

\[
\begin{align*}
\text{for } i &= 1 \text{ to } n \text{ do} \\
\quad &\text{for } j = 1 \text{ to } n \text{ do} \\
\quad &\quad \text{for } k = 1 \text{ to } n \text{ do} \\
\quad &\quad &\quad x += 1 \\
\quad &\quad \text{end for} \\
\quad &\text{end for} \\
\text{end for}
\end{align*}
\]

- Fact: run time is $O(n^3)$
Algorithm Sum-Product

`sum = 0`

`for` `i = 1` `to` `n` `do`
  `for` `j = i` `to` `n` `do`
  `end for`
`end for`

- It is $O(n^2)$. Is this the best bound?
Bounds for Common Functions

- Polynomials: Only highest degree term matters. $f(n)$ is $O(n^d)$ when
  \[ f(n) = p_0 + p_1 n + p_2 n^2 + \ldots + p_d a^d \]

- Logarithms: $\log_a n$ is $\Theta(\log_b n)$ for every $a > 1$ and every $b > 1$

- Log vs. Poly: $\log_b n$ is $O(n^d)$ for all $b$ and $a$
  - All polynomials grow faster than logarithm of any base

- Poly vs Expo: $n^d$ is $O(r^n)$ when $r > 1$
  - Exponential functions grow faster than polynomials
Common Running Times

![Graph showing common running times: constant, logarithmic, linear, quadratic, exponential.](image)
Common Running Times

- Constant: $O(1)$
- Logarithmic: $O(\log n)$
- Linear: $O(n)$
- Linearithmic: $O(n \log n)$
- Quadratic: $O(n^2)$
- Cubic: $O(n^3)$
- Exponential: $O(2^n)$