Complexity Analysis
2501ICT/7421ICT Nathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012
Outline

1. Measuring Complexity
   - Overview
   - Measuring the Efficiency of Algorithms

2. Big-O Notation
   - Big-O Analysis
   - Common Complexities

Measuring Complexity
Big-O Notation

René Hexel
Complexity Analysis
Overview

- Efficiency of Algorithms
- Time and Space
- Measuring Efficiency
- Big-O Analysis
- Examples and Case Studies
- Search Algorithms
- Sort Algorithms
Why Analyse Complexity?

- Improve Algorithm Efficiency
  - need to assess efficiency
  - can Make a Big Difference
- Large data sets
- Computationally intense problems
- But: Correctness is most important
  - make correct algorithm efficient, not vice versa!
Algorithm Efficiency

- Algorithms Consume Resources
  - processing time
  - memory space

- Compromise
  - speed up at expense of memory
  - slower, but more memory efficient algorithms

→ application dependent!
Problem Size

- Problem Size is the data set size an algorithm works on.

Example (Sum Integers in an Array)

<table>
<thead>
<tr>
<th>12</th>
<th>4</th>
<th>17</th>
<th>⋅⋅⋅</th>
<th>29</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>⋅⋅⋅</td>
<td>n − 1</td>
<td>n</td>
</tr>
</tbody>
</table>

- \( n \) Integers:
  - \( \rightarrow \) problem size is \( n \)
Measuring Efficiency

- Time a program using the computer clock
  - `time command` on the command line
  - Takes too long for large data sets
    - May not even complete!
  - System dependent (compiler, hardware, ...)
- Count block or instruction *iterations*
  - Better overall indicator
  - Works with abstract representations
    - Constant and variable figures (loops)
Execution Time

- AAAA: Abstractly assess an algorithm → without executing an actual program

Example (How many times is $S_1$ executed?)

```plaintext
for (i = 0; i < 10; i++)
for (i = 0; i < N; i++)
    $S_1$;
```

- $S_1$ is the Privileged Instruction
  - if $S_1$ takes a constant amount of time $k_1$, how long will the loop take to execute?
    - ⇒ execution time $t = N \cdot k_1$
Example (Adding a constant Overhead)

\[ S_0; \]
\[ \text{for (} i = 0; i < N; i++ \text{)} \]
\[ S_1; \]
\[ S_2; \]

- Assuming \( S_0 \) and \( S_2 \) together take a constant time \( k_2 \):
  \[ \Rightarrow \text{total execution time} \ t = N \cdot k_1 + k_2 \]
Complexity and Problem Size

- **Small $n$**
  - time differences are small
  - most algorithms perform similarly

- **Large $n$**
  - huge difference
  - several seconds vs. thousands of years or more!

- Complexity figure is significant for large values of $n$
### Large Values of $n$

#### Example (comparing different efficiencies)

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n \log_{10} n$</th>
<th>$n^2$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3.49</td>
<td>25</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>1,024</td>
</tr>
<tr>
<td>100</td>
<td>200</td>
<td>10,000</td>
<td>$10^{30}$</td>
</tr>
<tr>
<td>1,000</td>
<td>3,000</td>
<td>1,000,000</td>
<td>$10^{301}$</td>
</tr>
<tr>
<td>10,000</td>
<td>40,000</td>
<td>100,000,000</td>
<td>$10^{3010}$</td>
</tr>
</tbody>
</table>
### Influence of Lesser Terms

#### Example (Lesser Terms become Insignificant)

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^2 + 14n + 26$</th>
<th>Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>266</td>
<td>166 %</td>
</tr>
<tr>
<td>100</td>
<td>10,000</td>
<td>11,426</td>
<td>14.3 %</td>
</tr>
<tr>
<td>1,000</td>
<td>1,000,000</td>
<td>1,014,026</td>
<td>1.4 %</td>
</tr>
<tr>
<td>10,000</td>
<td>100,000,000</td>
<td>100,140,026</td>
<td>0.14 %</td>
</tr>
</tbody>
</table>

- $n^2$ has the biggest effect
  - ⇒ lesser terms become insignificant!

René Hexel
Complexity Analysis
Big-O Notation

- **Less Significant Terms are Ignored**
  - \( n^2 + 14n + 26 \) becomes \( O(n^2) \)
  - \( O(n^2) \) is read “Order n Squared”

- **General Case**
  - factor with highest significance determines Order of Magnitude
  - \( O(n^k): a + bn + cn^2 + \cdots + n^k \)
  - \( O(k^n): a + bn + cn^2 + \cdots + n^k + k^n \)
Magnitude of Complexity

Example (most significant term matters)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2n^2 + 6$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>$n^3 + 3000n^2 + 6$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>$2^n + 4n^{100} + 5n$</td>
<td>$O(2^n)$</td>
</tr>
</tbody>
</table>

- Placement of data may influence algorithm complexity
  - Best-Case, Average, and Worst-Case
    - Significant: Average and Worst-Case
Big-O Limitations

- Less Significant Terms
  → can be quite significant for small and medium size data sets!
- Constant of Proportionality
  → $800n^2$
    - almost 3 orders of magnitude more complex than $n^2$
### Common Orders of Magnitude

<table>
<thead>
<tr>
<th>Big-O Notation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(1)</td>
<td>Constant</td>
</tr>
<tr>
<td>O(log n)</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>O(n)</td>
<td>Linear</td>
</tr>
<tr>
<td>O(n log n)</td>
<td>Log Linear</td>
</tr>
<tr>
<td>O(n^2)</td>
<td>n-Squared</td>
</tr>
<tr>
<td>O(2^n)</td>
<td>Exponential</td>
</tr>
</tbody>
</table>
Common Cases

- Single loop processing $n$ items
  - $O(n)$
- Two nested loops of $n$ items
  - $O(n^2)$
- Binary Search
  - $O(\log_2 n)$
- Efficient Sort
  - $O(n \log_2 n)$
Logarithms

- $\log_2 n$ complexity is very common!
  - $\log n$

- Examples:
  - Repeated doubling
    - Start at 1, double until size reaches $n$
  - Repeated halving
    - Start at $n$, half data set until 1 is reached
  - Binary searches, efficient sorting, . . .