CMSC 132: Object-Oriented Programming II

Algorithmic Complexity II

Department of Computer Science
University of Maryland, College Park
Overview

- Critical sections
- Comparing complexity
- Types of complexity analysis
Analyzing Algorithms

Goal
- Find asymptotic complexity of algorithm

Approach
- Ignore less frequently executed parts of algorithm
- Find critical section of algorithm
- Determine how many times critical section is executed as function of problem size
Critical Section of Algorithm

- Heart of algorithm
- Dominates overall execution time

Characteristics
- Operation central to functioning of program
- Contained inside deeply nested loops
- Executed as often as any other part of algorithm

Sources
- Loops
- Recursion
Critical Section Example 1

Code (for input size n)

1. A
2. for (int i = 0; i < n; i++)
3. B
4. C

Code execution

- A ⇒ once
- B ⇒ n times
- C ⇒ once

Time ⇒ 1 + n + 1 = O(n)
Critical Section Example 2

Code (for input size \( n \))

1. A
2. for (int i = 0; i < n; i++)
3. B
4. for (int j = 0; j < n; j++)
5. C
6. D

Code execution

- A \( \Rightarrow \) once
- B \( \Rightarrow \) \( n \) times
- C \( \Rightarrow \) \( n^2 \) times
- D \( \Rightarrow \) once

Time \( \Rightarrow 1 + n + n^2 + 1 = O(n^2) \)
Critical Section Example 3

Code (for input size $n$)
1. A
2. for (int $i = 0; i < n; i++$)
3. for (int $j = i+1; j < n; j++$)
4. B

Code execution
- A $\Rightarrow$ once
- B $\Rightarrow \frac{1}{2} n (n-1)$ times

Time $\Rightarrow 1 + \frac{1}{2} n^2 = O(n^2)$
Critical Section Example 4

Code (for input size $n$)

1. A
2. for (int i = 0; i < n; i++)
3. for (int j = 0; j < 10000; j++)
4. B

Code execution

- A $\Rightarrow$ once
- B $\Rightarrow$ 10000 $n$ times

Time $\Rightarrow$ $1 + 10000n = O(n)$
Critical Section Example 5

Code (for input size n)

1. for (int i = 0; i < n; i++)
2. for (int j = 0; j < n; j++)
3. A
4. for (int i = 0; i < n; i++)
5. for (int j = 0; j < n; j++)
6. B

Code execution

A ⇒ n^2 times
B ⇒ n^2 times

Time ⇒ n^2 + n^2 = O(n^2)
**Critical Section Example 6**

**Code (for input size $n$)**
1. $i = 1$
2. while $(i < n)$
3. A
4. $i = 2 \times i$
5. B

**Code execution**
- A $\Rightarrow \log(n)$ times
- B $\Rightarrow 1$ times

**Time** $\Rightarrow \log(n) + 1 = O(\log(n))$
Critical Section Example 7

Code (for input size $n$)

1. DoWork (int $n$)
2. if ($n == 1$)
3. A
4. else
5. DoWork($n/2$)
6. DoWork($n/2$)

Code execution

- A $\Rightarrow$ 1 times
- DoWork($n/2$) $\Rightarrow$ 2 times

Time(1) $\Rightarrow$ 1
Time($n$) = $2 \times$ Time($n/2$) + 1
Recursive Algorithms

Definition

An algorithm that calls itself

Components of a recursive algorithm

1. Base cases
   - Computation with no recursion

2. Recursive cases
   - Recursive calls
   - Combining recursive results
Recursive Algorithm Example

Code (for input size n)
1. DoWork (int n)
2. if (n == 1)
3. A
4. else
5. DoWork(n/2)
6. DoWork(n/2)
# Asymptotic Complexity Categories

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(1))</td>
<td>Constant</td>
<td>Array access</td>
</tr>
<tr>
<td>(O(\log(n)))</td>
<td>Logarithmic</td>
<td>Binary search</td>
</tr>
<tr>
<td>(O(n))</td>
<td>Linear</td>
<td>Largest element</td>
</tr>
<tr>
<td>(O(n \log(n)))</td>
<td>N log N</td>
<td>Optimal sort</td>
</tr>
<tr>
<td>(O(n^2))</td>
<td>Quadratic</td>
<td>2D Matrix addition</td>
</tr>
<tr>
<td>(O(n^3))</td>
<td>Cubic</td>
<td>2D Matrix multiply</td>
</tr>
<tr>
<td>(O(n^k))</td>
<td>Polynomial</td>
<td>Linear programming</td>
</tr>
<tr>
<td>(O(k^n))</td>
<td>Exponential</td>
<td>Integer programming</td>
</tr>
</tbody>
</table>

From smallest to largest

For size \(n\), constant \(k > 1\)
Comparing Complexity

- Compare two algorithms
  - \( f(n), g(n) \)

- Determine which increases at faster rate
  - As problem size \( n \) increases

- Can compare ratio
  - If \( \infty \), \( f() \) is larger
  - If 0, \( g() \) is larger
  - If constant, then same complexity

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)}
\]
Complexity Comparison Examples

1. \( \log(n) \) vs. \( n^{\frac{1}{2}} \)

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} \quad \rightarrow \quad \lim_{n \to \infty} \frac{\log(n)}{n^{\frac{1}{2}}} \quad \rightarrow \quad 0
\]

2. \( 1.001^n \) vs. \( n^{1000} \)

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} \quad \rightarrow \quad \lim_{n \to \infty} \frac{1.001^n}{n^{1000}} \quad \rightarrow \quad ??
\]

Not clear, use L’Hopital’s Rule
Additional Complexity Measures

- **Upper bound**
  - Big-O  \( \Rightarrow O(\ldots) \)
  - Represents upper bound on # steps

- **Lower bound**
  - Big-Omega  \( \Rightarrow \Omega(\ldots) \)
  - Represents lower bound on # steps

- **Combined bound**
  - Big-Theta  \( \Rightarrow \Theta(\ldots) \)
  - Represents combined upper/lower bound on # steps
  - Best possible asymptotic solution
2D Matrix Multiplication Example

- **Problem**
  \[ C = A \times B \]

- **Lower bound**
  \[ \Omega(n^2) \]
  Required to examine 2D matrix

- **Upper bounds**
  \[ O(n^3) \] Basic algorithm
  \[ O(n^{2.807}) \] Strassen’s algorithm (1969)
  \[ O(n^{2.376}) \] Coppersmith & Winograd (1987)

- **Improvements still possible (open problem)**
  Since upper & lower bounds do not match
## Additional Complexity Categories

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Nondeterministic polynomial time (NP)</td>
</tr>
<tr>
<td>PSPACE</td>
<td>Polynomial space</td>
</tr>
<tr>
<td>EXPSPACE</td>
<td>Exponential space</td>
</tr>
<tr>
<td>Decidable</td>
<td>Can be solved by finite algorithm</td>
</tr>
<tr>
<td>Undecidable</td>
<td>Not solvable by finite algorithm</td>
</tr>
</tbody>
</table>

### Mostly of academic interest only

- Quadratic algorithms usually too slow for large data
- Use fast **heuristics** to provide non-optimal solutions
NP Time Algorithm

- Polynomial solution possible
  - If make correct guesses on how to proceed
- Required for many fundamental problems
  - Boolean satisfiability
  - Traveling salesman problem (TLP)
  - Bin packing
- Key to solving many optimization problems
  - Most efficient trip routes
  - Most efficient schedule for employees
  - Most efficient usage of resources
NP Time Algorithm

Properties of NP
- Can be solved with exponential time
- Not proven to require exponential time
- Currently solve using heuristics

NP-complete problems
- Representative of all NP problems
- Solution can be used to solve any NP problem
- Examples
  - Boolean satisfiability
  - Traveling salesman
P = NP?

Are NP problems solvable in polynomial time?

- Prove $P=NP$
  - Show polynomial time solution exists for any NP-complete problem
- Prove $P \neq NP$
  - Show no polynomial-time solution possible
  - The expected answer

Important open problem in computer science

- $1$ million prize offered by Clay Math Institute
Algorithmic Complexity Summary

- Asymptotic complexity
  - Fundamental measure of efficiency
  - Independent of implementation & computer platform

- Learned how to
  - Examine program
  - Find critical sections
  - Calculate complexity of algorithm
  - Compare complexity