CMSC 132:
OBJECT-ORIENTED PROGRAMMING II

Algorithmic Complexity II

Department of Computer Science
University of Maryland, College Park
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. }
  5. C

• Code execution
  • A \( \Rightarrow \) once
  • B \( \Rightarrow \) \( n \) times
  • C \( \Rightarrow \) once

• Time \( \Rightarrow 1 + n + 1 = O(n) \)
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.     }
  7. }
  8. 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
  5.     }
  6. }

• 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
  5.     }
  6. }

• Code execution
  • A \implies \text{once}
  • B \implies 10000 \text{ n times}

• Time \implies 1 + 10000 \text{ n = 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 $\Rightarrow n^2$ times
  • B $\Rightarrow n^2$ times

• Time $\Rightarrow n^2 + n^2 = O(n^2)$
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)
  7. }

• Code execution
  • A ⇒ 1 times
  • DoWork(n/2) ⇒ 2 times

• Time(1) ⇒ 1

Time(n) = 2 × 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)
7. }

base case
recursive cases
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

• $\log(n)$ vs. $n^{\frac{1}{2}}$

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

• $1.001^n$ vs. $n^{1000}$

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

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