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

critical section

- 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)

```
A
for (int i = 0; i < n; i++) {
    B
    for (int j = 0; j < n; j++) {
        C
    }
}
D
```

critical section

- 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)

```
A
for (int i = 0; i < n; i++) {
   for (int j = i+1; j < n; j++) {
      B
   }
}
```

critical section

- Code execution
  - A $\Rightarrow$ once
  - B $\Rightarrow$ ½ n $(n-1)$ times
- Time $\Rightarrow 1 + ½\, n^2 = O(n^2)$

Critical Section Example 4

- Code (for input size n)

```
A
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 10000; j++) {
        B
    }
}
```

critical section

- Code execution
  - A $\Rightarrow$ once
  - B $\Rightarrow$ $10000\,n$ times
- Time $\Rightarrow 1 + 10000\,n = O(n)$

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

critical sections

- 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 × i

     }
  5. B

critical section

- Code execution
  - A ⇒ log(n) times
  - B ⇒ 1 times
- Time ⇒ log(n) + 1 = O(log(n))

Critical Section Example 7

- Code (for input size n)

```
DoWork (int n)
   if (n == 1)
      A
   else {
      DoWork(n/2)
      DoWork(n/2)
   }
```

critical sections

- 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)

```
DoWork (int n)
   if (n == 1)
      A
   else {
      DoWork(n/2)
      DoWork(n/2)
   }
```

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 ∞, 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. $n^{1/2}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \rightarrow \lim_{n\to\infty} \frac{\log(n)}{n^{1/2}} \rightarrow 0$$

- 1.001^n vs. n^{1000}

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

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 * 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