CMSC 132:
Object-Oriented Programming II

Algorithmic Complexity II

Department of Computer Science
University of Maryland, College Park

Announcements

- Regarding Friday before spring break

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
  - Usually contained inside deeply nested loops
- Sources
  - Loops
  - Recursion

Critical Section Example 1

- Code (for input size n)
  - A
  - for (int i = 0; i < n; i++) {
  - B
  - }
  - C
- Code execution
  - A ⇒ once
  - B ⇒ n times
  - C ⇒ once
- Time ⇒ 1 + n + 1 = O(n)

critical section

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
- Code execution
  - A ⟹ once
  - B ⟹ n times
  - C ⟹ n2 times
  - D ⟹ once
- Time ⟹ 1 + n + n2 + 1 = O(n2)

critical section

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
  - }
  - }
- Code execution
  - A ⇒ once
  - B ⇒ ½ n (n-1) times
- Time ⇒ 1 + ½ n2 – ½ n = O(n2)

critical section

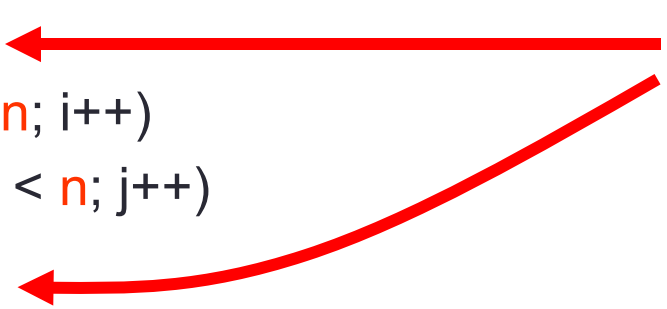
Critical Section Example 4

- Code (for input size n)
  - A
  - for (int i = 0; i < n; i++) {
  - for (int j = 0; j < 10000; j++) {
  - B
  - }
  - }
- Code execution
  - A ⇒ once
  - B ⇒ 10000 n times
- Time ⇒ 1 + 10000 n = O(n)

critical section

Critical Section Example 5

- Code (for input size n)
  - for (int i = 0; i < n/2; i++)
  - for (int j = 0; j < n/2; j++)
  - A
  - for (int i = 0; i < n; i++)
  - for (int j = 0; j < n; j++)
  - B



- Code execution
  - A ⇒ n2/4 times
  - B ⇒ n2 times
- Time ⇒ n2 + n2 = O(n2)

Critical Section Example 6

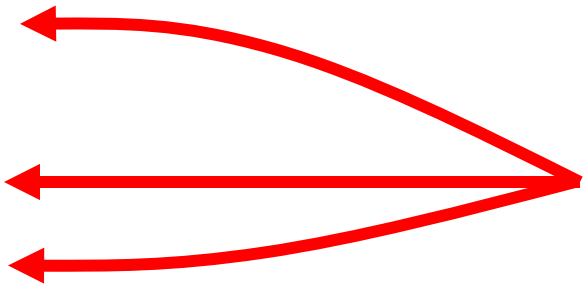
- Code (for input size n)
  - i = 1
  - while (i < n) {
  - A
  - i = 2 × i

    }
    - B
  - Code execution
    - i = 1 ⟹ 1 times
    - A ⟹ log(n) times
    - B ⟹ 1 times
  - Time ⟹ 1 + log(n) + 1 = O(log(n))

critical section

Critical Section Example 7 (Recursion)

- Code (for input size n)
  - DoWork (int n)
  - if (n == 1)
  - A
  - else {
  - DoWork(n/2)
  - DoWork(n/2)
  - }



critical sections

- Code execution
  - A ⇒ 1 times
  - DoWork(n/2) ⇒ 2 times
- Time(1) ⇒ 1 Time(n) = 2 × Time(n/2) + 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 ∞, f() is larger
  - If 0, g() is larger
  - If constant, then same complexity

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

- Example (**log(n) vs. $n^{1/2}$**)

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

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

2D Matrix Multiplication Example

- Problem
  - C = A * B
- Lower bound
  - $\Omega(n2)$ Required to examine 2D matrix
- Upper bounds
  - O(n3) Basic algorithm
  - O(n2.807) Strassen's algorithm (1969)
  - O(n2.376) Coppersmith & Winograd (1987)
- Improvements still possible (open problem)
  - Since upper & lower bounds do not match