

CMSC 132: Object-Oriented Programming II



Algorithmic Complexity II

Department of Computer Science
University of Maryland, College Park

1

Overview

- Critical sections
- Comparing complexity
- Types of complexity analysis

2

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

3

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

4

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 \Rightarrow
- B \Rightarrow
- C \Rightarrow

■ Time \Rightarrow

5

Critical Section Example 1

■ Code (for input size n)

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

critical
section

■ Code execution

- A \Rightarrow once
- B \Rightarrow n times
- C \Rightarrow once

■ Time $\Rightarrow 1 + n + 1 = O(n)$

6

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
- B \Rightarrow
- C \Rightarrow
- D \Rightarrow

■ Time \Rightarrow

7

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

**critical
section**

■ Code execution

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

■ Time $\Rightarrow 1 + n + n^2 + 1 = O(n^2)$


8

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
 - B \Rightarrow
- Time \Rightarrow

9

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**



10

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
 - B \Rightarrow
- Time \Rightarrow

11

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 $\Rightarrow 1 + 10000 n = O(n)$
- critical section**


12

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
- `B` \Rightarrow

■ Time \Rightarrow

13

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 $\Rightarrow n^2 + n^2 = O(n^2)$

14

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
- B \Rightarrow

■ Time \Rightarrow

15

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

← critical section

■ Code execution

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

■ Time $\Rightarrow \log(n) + 1 = O(\log(n))$

16

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
- DoWork($n/2$) \Rightarrow

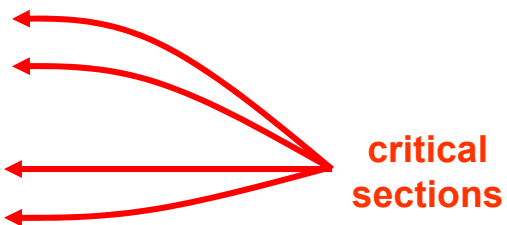
■ Time(1) \Rightarrow Time(n) =

17

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

18

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

19

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)
```

base
case



recursive
cases



20

Asymptotic Complexity Categories

Complexity	Name	Example
■ $O(1)$	Constant	Array access
■ $O(\log(n))$	Logarithmic	Binary search
■ $O(n)$	Linear	Largest element
■ $O(n \log(n))$	$N \log N$	Optimal sort
■ $O(n^2)$	Quadratic	2D Matrix addition
■ $O(n^3)$	Cubic	2D Matrix multiply
■ $O(n^k)$	Polynomial	Linear programming
■ $O(k^n)$	Exponential	Integer programming

From smallest to largest

For size n , constant $k > 1$

21

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

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

22

Complexity Comparison Examples

■ $\log(n)$ vs. $n^{1/2}$

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

■ 1.001^n vs. n^{1000}

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

Not clear, use
L'Hopital's Rule

23

Additional Complexity Measures

■ Upper bound

- Big-O $\Rightarrow O(\dots)$
- Represents upper bound on # steps

■ Lower bound

- Big-Omega $\Rightarrow \Omega(\dots)$
- Represents lower bound on # steps

■ Combined bound

- Big-Theta $\Rightarrow \Theta(\dots)$
- Represents combined upper/lower bound on # steps
- Best possible asymptotic solution

24

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

25

Additional Complexity Categories

- | ■ Name | Description |
|---------------|---------------------------------------|
| ■ NP | Nondeterministic polynomial time (NP) |
| ■ PSPACE | Polynomial space |
| ■ EXPSPACE | Exponential space |
| ■ Decidable | Can be solved by finite algorithm |
| ■ Undecidable | Not solvable by finite algorithm |
-
- Mostly of academic interest only
 - Quadratic algorithms usually too slow for large data
 - Use fast **heuristics** to provide non-optimal solutions

26

NP Time Algorithm

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

27

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

28

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

29

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

30