

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



**critical
section**

4. 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. 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)

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

critical
section



■ 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

critical
section



■ Code execution

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

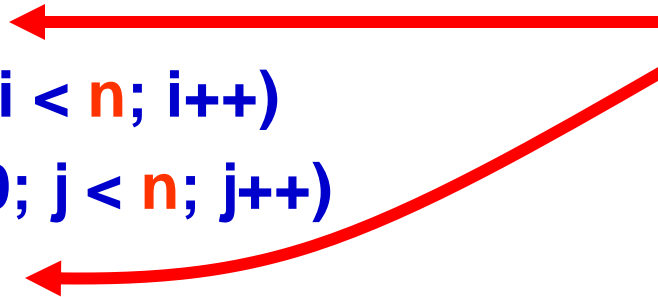
■ Time $\Rightarrow 1 + 10000 n = O(n)$

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. $i = 2 \times i$

 }

5. B



**critical
section**

■ Code execution

■ $A \Rightarrow \log(n)$ times

■ $B \Rightarrow 1$ times

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

Critical Section Example 7

■ Code (for input size n)

1. DoWork (int n)

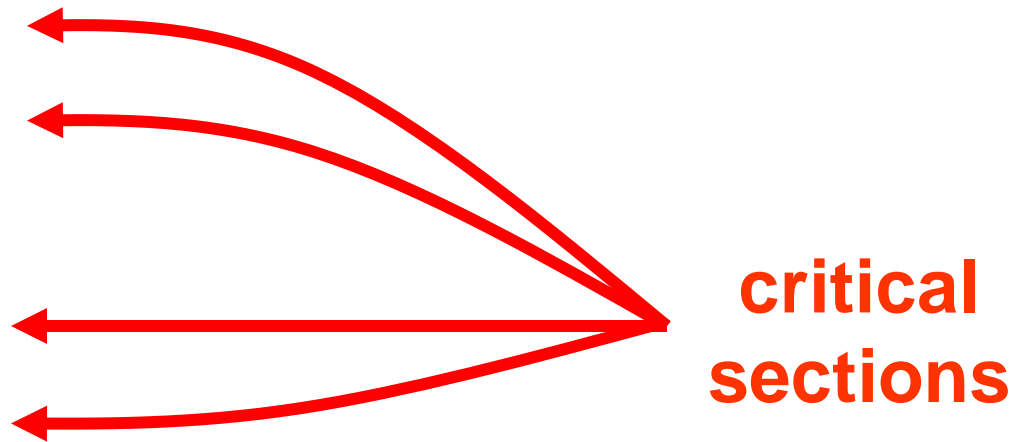
2. if ($n == 1$)

3. A

4. else

5. DoWork($n/2$)

6. DoWork($n/2$)



■ Code execution

■ $A \Rightarrow 1$ times

■ $\text{DoWork}(n/2) \Rightarrow 2$ times

■ $\text{Time}(1) \Rightarrow 1$ $\text{Time}(n) = 2 \times \text{Time}(n/2) + 1$

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)**

**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

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

Complexity Comparison Examples

■ $\log(n)$ vs. $n^{1/2}$

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

■ 1.001^n vs. n^{1000}

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

Not clear, use
L'Hopital's Rule

Additional Complexity Measures

■ Upper bound

- Big-O $\Rightarrow O(\dots)$

- 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

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

Additional Complexity Categories

- | Name | Description |
|----------------------|-----------------------------------|
| ■ P | Deterministic polynomial time |
| ■ NP | Nondeterministic polynomial time |
| ■ PSPACE | Polynomial space |
| ■ EXPSPACE | Exponential space |
| ■ Decidable | Can be solved by finite algorithm |
| ■ Undecidable | Not solvable by finite algorithm |
-
- If a problem has a algorithm that solves it in time X , then the problem is said to be in X
 - e.g., matrix multiplication is in P

Why do we care?

- If a problem can't be solved in P time, then no algorithm can solve *all* cases exactly in a *reasonable* amount of time
 - But there might be an algorithm that always quickly gives close approximations
 - Or an algorithm that gives quick exact answers for the cases we care about
- Issues such as PSPACE vs. EXPSPACE are interesting theoretical issues, and sometimes provide practice insight

NP Time Algorithms

■ Two ways of thinking about it

■ First way:

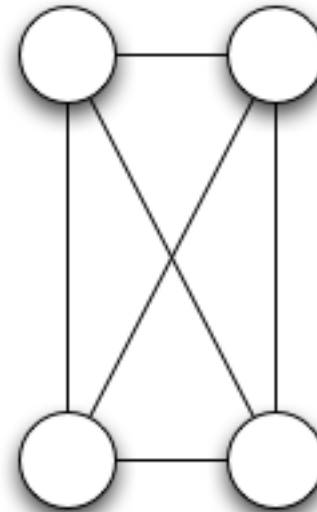
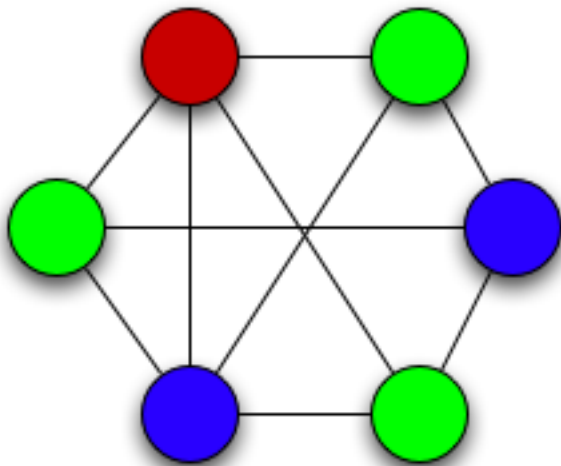
- Given a problem, and a potential answer, can we verify that the answer is correct in polynomial time?

■ Second way:

- Whenever the algorithm wants, it can clone itself in two
- If either clone finds an answer, the entire system produces an answer

Graph 3-coloring

- Given a graph (vertices and undirected edges)
- Can you find a way to color each vertex either blue, red, or green
- Such that no two vertices connected by an edge have the same color?



Some
graphs are
hard

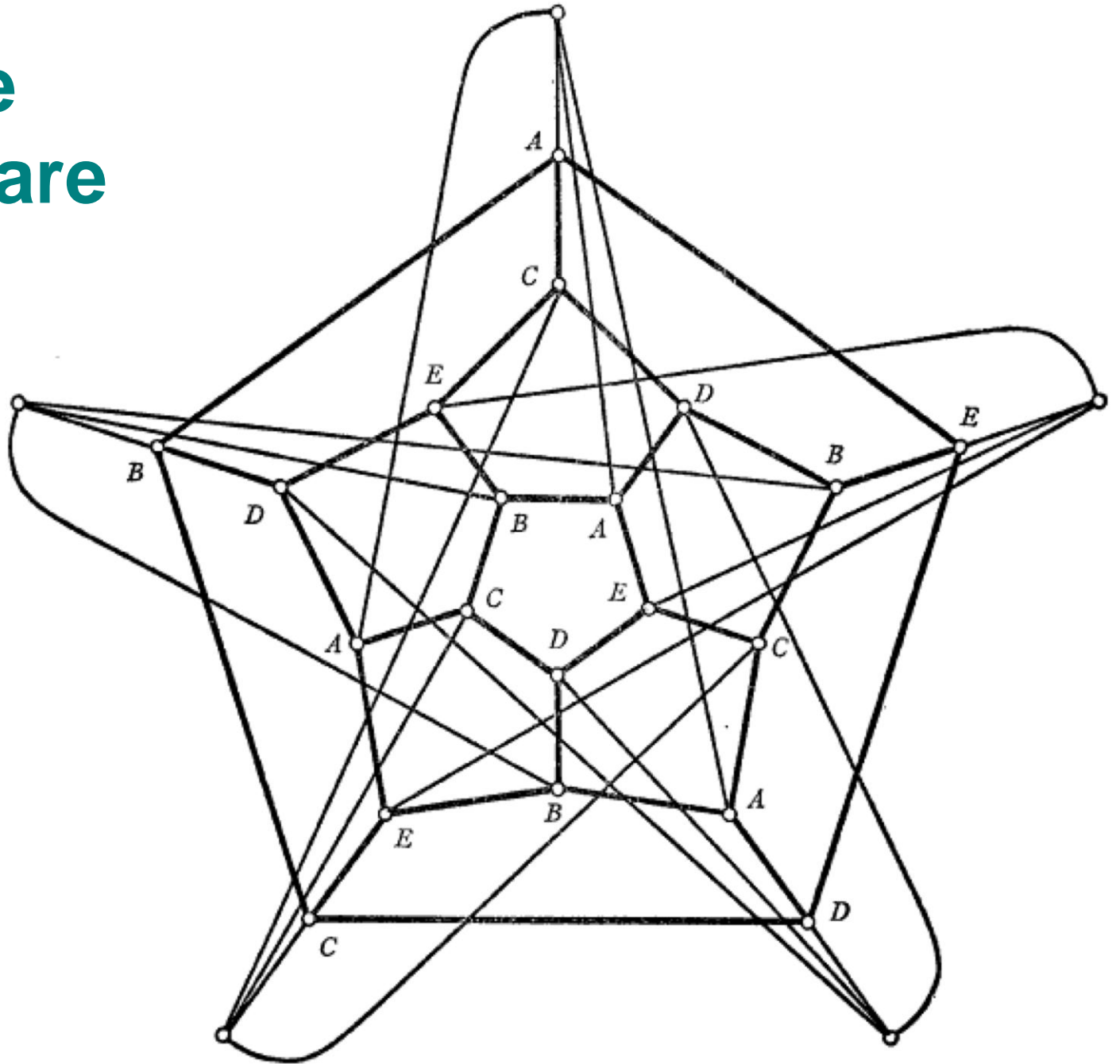


FIG. 1

3 coloring a graph is in NP

- **There exist NP algorithms to 3 color a graph**
 - **Guess a 3 coloring**
 - **Verify that the coloring is valid**
 - **Easy to do in $O(n)$ time**
- **No one knows if there exists a polynomial time algorithm to find a 3 coloring for an arbitrary graph**
 - **If you come up with one, even one that runs in time $O(n^{100})$, you win one million dollars**
 - **Seriously**

NP Time Algorithm

- **Many interesting problems are solvable with an NP algorithm, but not known to be solvable with a P algorithm**
 - **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 Complete problems

- **Some problems are NP-complete**
 - They are in NP
 - And if a polynomial time algorithm existed for the program
 - Then every single last problem in NP could be solved in polynomial time
- **Almost all problems that are in NP but are not known to be in P are NP-complete**
 - But not all

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 for some problem in NP
 - The expected answer
- **The most important open problem in CS**
 - \$1 million prize offered by Clay Math Institute
 - Plus front page, NY Times, job offers galore, instant Ph.D. in Computer Science

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**