

Algorithmic Complexity 3



Fawzi Emad
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Overview

- **Recurrence relations**
- **Complexity categories**
- **Comparing complexity**
- **Types of complexity analysis**

Critical Section Example 7

■ Code (for input size n)

1. DoWork (int n)

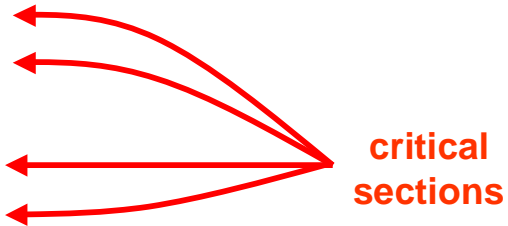
2. if ($n == 1$)

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



Recurrence Relations

■ Definition

- Value of a function at a point is given in terms of its value at other points

■ Examples

- $T(n) = T(n-1) + k$
- $T(n) = T(n-1) + n$
- $T(n) = T(n-1) + T(n-2)$
- $T(n) = T(n/2) + k$
- $T(n) = 2 \times T(n/2) + k$

Recurrence Relations

■ Base case

- Value of function at some specified points
- Also called boundary values / boundary conditions

■ Base case example

- $T(1) = 0$
- $T(1) = 1$
- $T(2) = 1$
- $T(2) = k$


Solving Recurrence Equations

■ Back substitution (iteration method)

- Iteratively substitute recurrence into formula

■ Example

- $T(1) = 5$
- $T(n) = T(n-1) + 1$


$$\begin{aligned} &= (T(n-2) + 1) + 1 \\ &= ((T(n-3) + 1) + 1) + 1 \\ &= (((T(n-4) + 1) + 1) + 1) + 1 \\ &= \dots \\ &= (\dots (T(1) + 1) + \dots) + 1 \\ &= (\dots (5 + 1) + \dots) + 1 \\ &= n + 4 \end{aligned}$$

Example Recurrence Solutions

■ Examples

- $T(n) = T(n-1) + k \Rightarrow O(n)$
- $T(n) = T(n-1) + n \Rightarrow O(n^2)$
- $T(n) = T(n-1) + T(n-2) \Rightarrow O(n!)$
- $T(n) = T(n/2) + k \Rightarrow O(\log(n))$
- $T(n) = 2 \times T(n/2) + k \Rightarrow O(n)$
- $T(n) = 2 \times T(n-1) + k \Rightarrow O(2^n)$

■ Many additional issues, solution methods

- Take CMSC 351 – Introduction to Algorithms

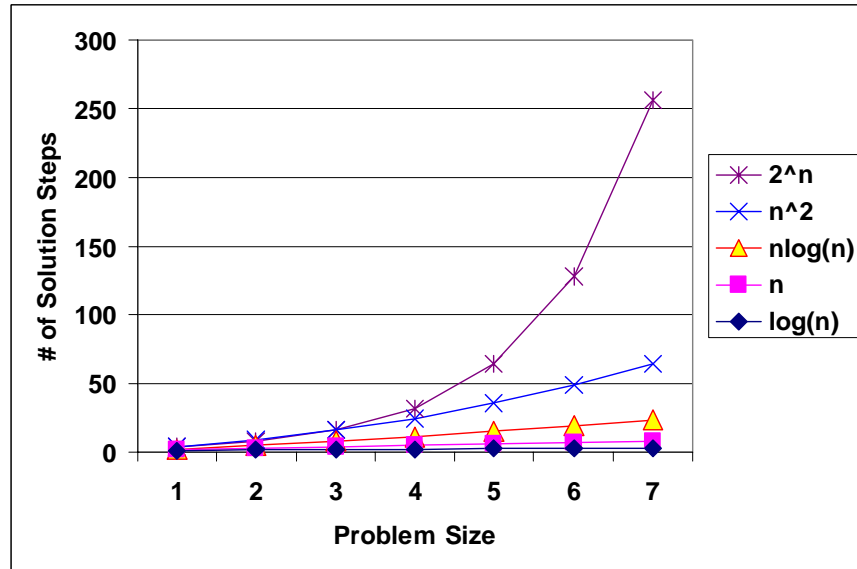
Asymptotic Complexity Categories

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

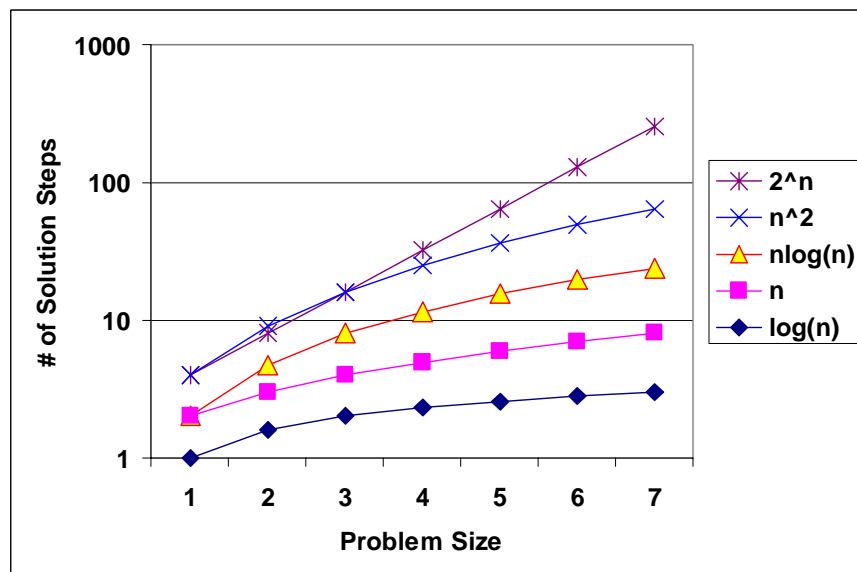
From smallest to largest

For size n , constant $k > 1$

Complexity Category Example



Complexity Category Example



Calculating Asymptotic Complexity

■ As n increases

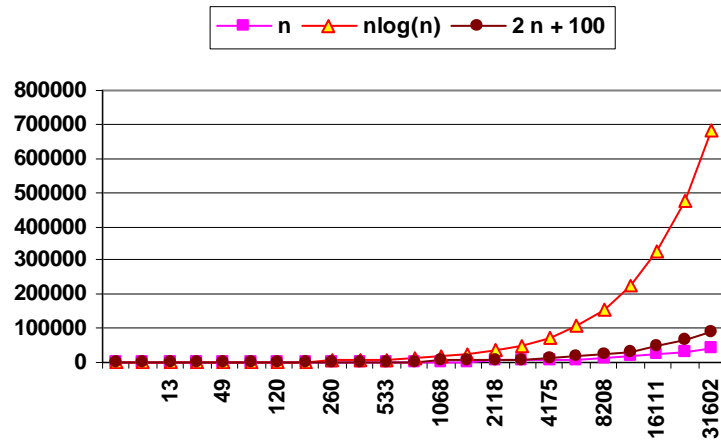
- Highest complexity term dominates
- Can ignore lower complexity terms

■ Examples

- $2n + 100 \Rightarrow O(n)$
- $n \log(n) + 10n \Rightarrow O(n \log(n))$
- $\frac{1}{2}n^2 + 100n \Rightarrow O(n^2)$
- $n^3 + 100n^2 \Rightarrow O(n^3)$
- $\frac{1}{100}2^n + 100n^4 \Rightarrow O(2^n)$

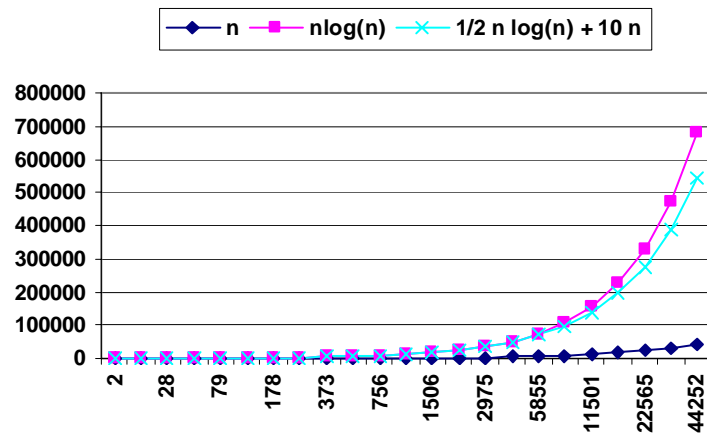
Complexity Examples

■ $2n + 100 \Rightarrow O(n)$



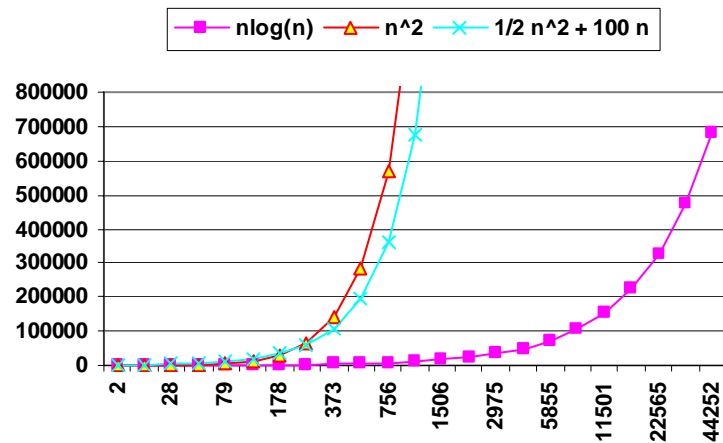
Complexity Examples

■ $\frac{1}{2} n \log(n) + 10 n \Rightarrow O(n \log(n))$



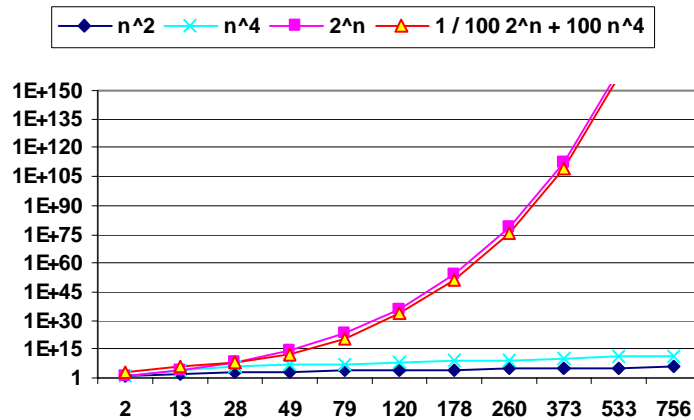
Complexity Examples

■ $\frac{1}{2} n^2 + 100 n \Rightarrow O(n^2)$



Complexity Examples

■ $1/100 2^n + 100 n^4 \Rightarrow O(2^n)$



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. $n^{1/2}$

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

- 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

L'Hopital's Rule

- If ratio is indeterminate
 - $0/0$ or ∞/∞
- Ratio of **derivatives** computes same value

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

- Can simplify ratio by repeatedly taking derivatives of numerator & denominator

Using L'Hopital's Rule

■ 1.001^n vs. n^{1000}

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &\rightarrow \lim_{n \rightarrow \infty} \frac{1.001^n}{n^{1000}} \\ &\rightarrow \lim_{n \rightarrow \infty} \frac{n \cdot 1.001^{n-1}}{1000 n^{999}} \\ &\rightarrow \lim_{n \rightarrow \infty} \frac{n(n-1) \cdot 1.001^{n-2}}{1000 \times 999 n^{998}} \\ &\rightarrow \dots \rightarrow \infty \end{aligned}$$

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 |
|---------------|---------------------------------------|
| ■ 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

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

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

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

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