CMSC 132: Object-Oriented Programming II

Algorithmic Complexity I

Department of Computer Science
University of Maryland, College Park

Algorithm Efficiency

- Efficiency
  - Amount of resources used by algorithm
    - Time, space
- Measuring efficiency
  - Benchmarking
    - Approach
      - Pick some desired inputs
      - Actually run implementation of algorithm
      - Measure time & space needed
  - Asymptotic analysis

Benchmarking

- Advantages
  - Precise information for given configuration
    - Implementation, hardware, inputs
- Disadvantages
  - Affected by configuration
    - Data sets (often too small)
      - Dataset that was the right size 3 years ago is likely too small now
    - Hardware
    - Software
  - Affected by special cases (biased inputs)
  - Does not measure intrinsic efficiency

Asymptotic Analysis

- Approach
  - Mathematically analyze efficiency
  - Calculate time as function of input size n
    - $T \approx O(f(n))$
    - T is on the order of f(n)
    - “Big O” notation
- Advantages
  - Measures intrinsic efficiency
  - **Dominates efficiency for large input sizes**
  - Programming language, compiler, processor irrelevant

Search Comparison

- For number between 1…100
  - Simple algorithm = 50 steps
  - Binary search algorithm = log2(n) = 7 steps
- For number between 1…100,000
  - Simple algorithm = 50,000 steps
  - Binary search algorithm = log2(n) (about 17 steps)
- Binary search is much more efficient!

Asymptotic Complexity

- Comparing two linear functions

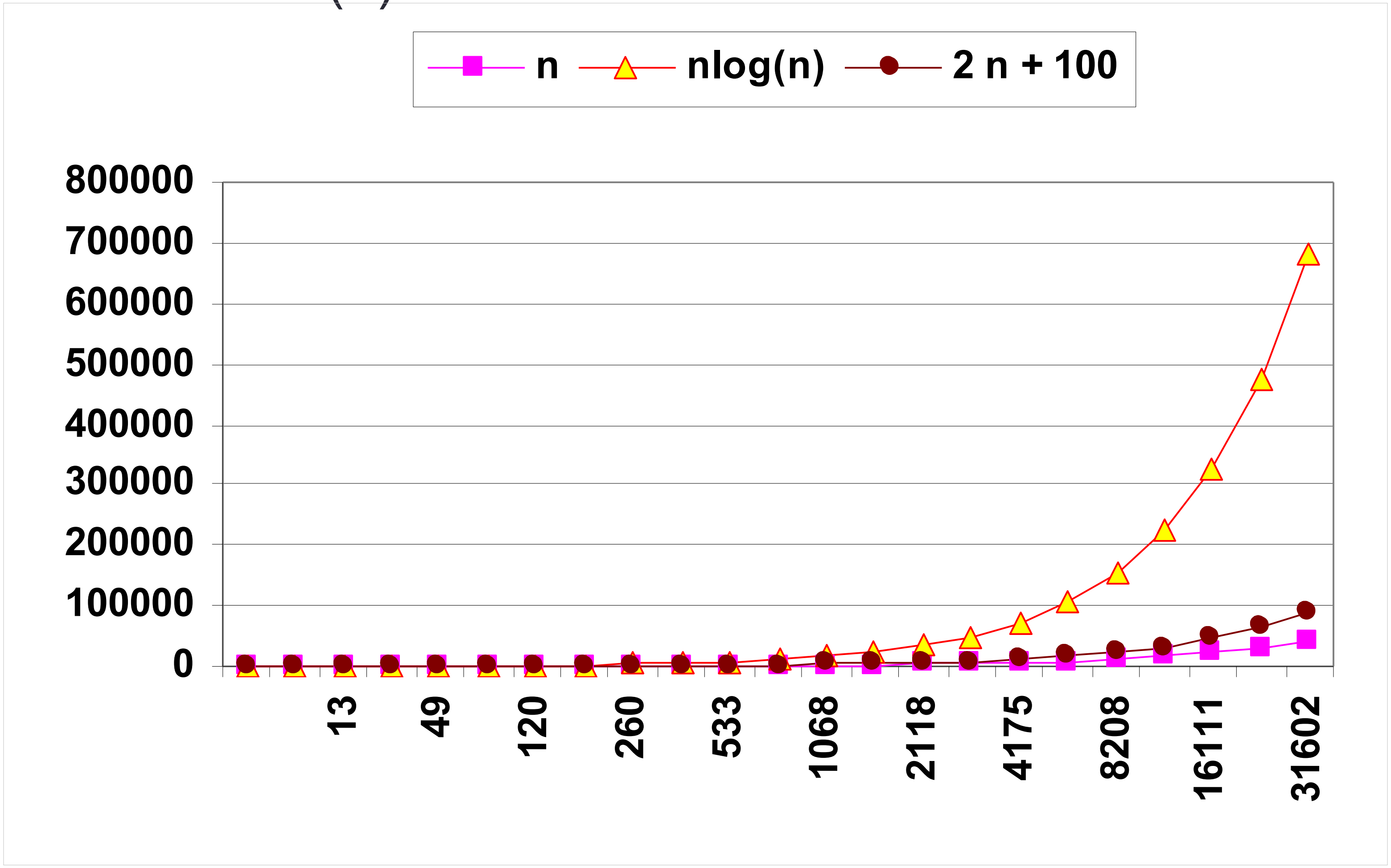
Size	Running Time	
	n/2	4n+3
64	32	259
128	64	515
256	128	1027
512	256	2051

Asymptotic Complexity

- Comparing two functions
  - n/2 and 4n+3 behave similarly
  - Run time roughly doubles as input size doubles
  - Run time increases linearly with input size
- For large values of n
  - Time(2n) / Time(n) approaches exactly 2
- Both are O(n) programs
- Example: 2n + 100 □ O(n) (next slide)

Complexity Example

- 2n + 100 ⇒ O(n)



Asymptotic Complexity

- Comparing two quadratic functions

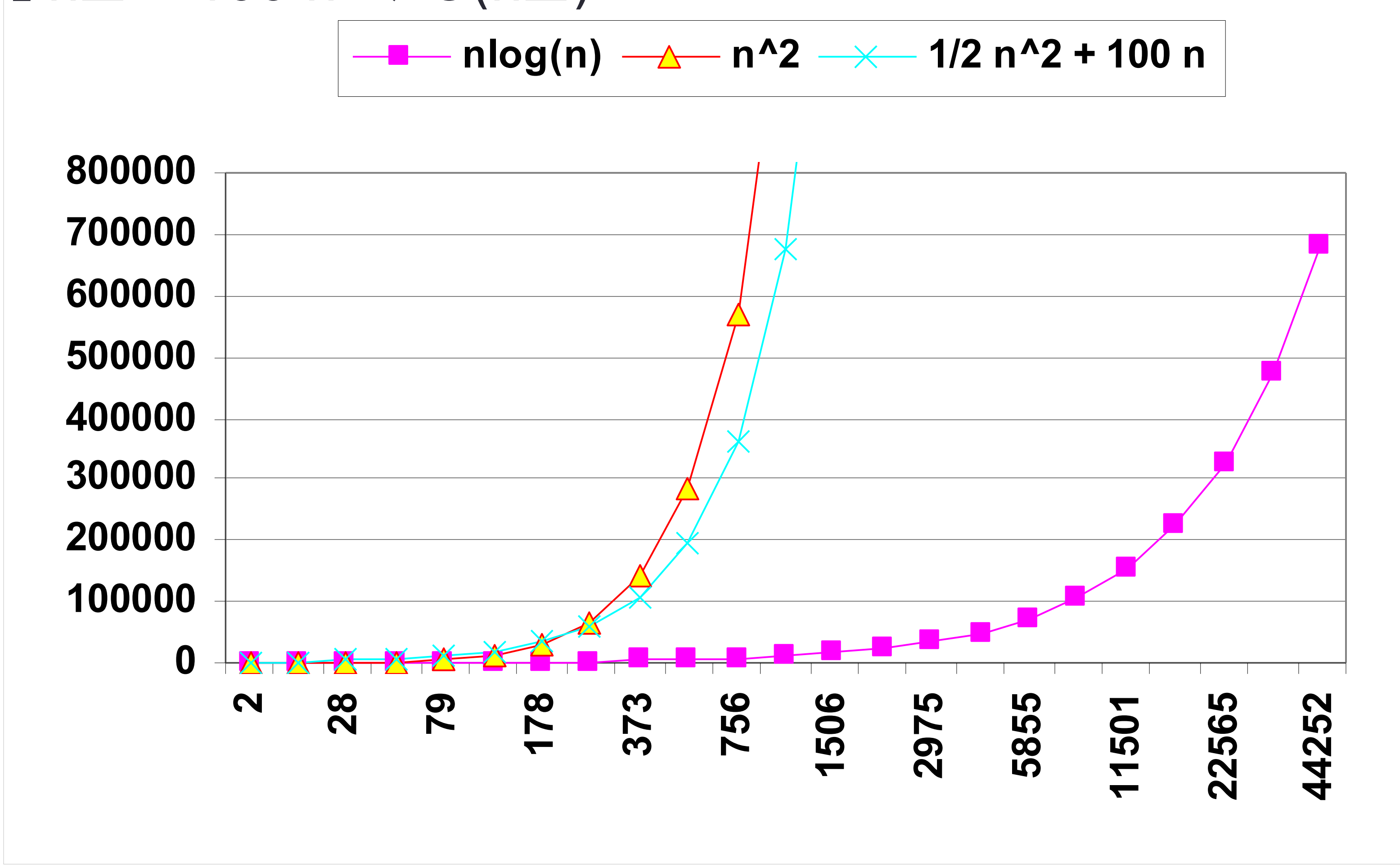
Size	Running Time	
	n2	2 n2 + 8
2	4	16
4	16	40
8	64	132
16	256	520

Asymptotic Complexity

- Comparing two functions
  - n2 and 2 n2 + 8 behave similarly
  - Run time roughly increases by 4 as input size doubles
  - Run time increases quadratically with input size
- For large values of n
  - Time(2n) / Time(n) approaches 4
- Both are O(n2) programs
- **Example:** ½ n2 + 100 n □ O(n2) (next slide)

Complexity Examples

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



Asymptotic Complexity

- Comparing two log functions

Size	Running Time	
	log2(n)	5 * log2(n) + 3
64	6	33
128	7	38
256	8	43
512	9	48

Asymptotic Complexity

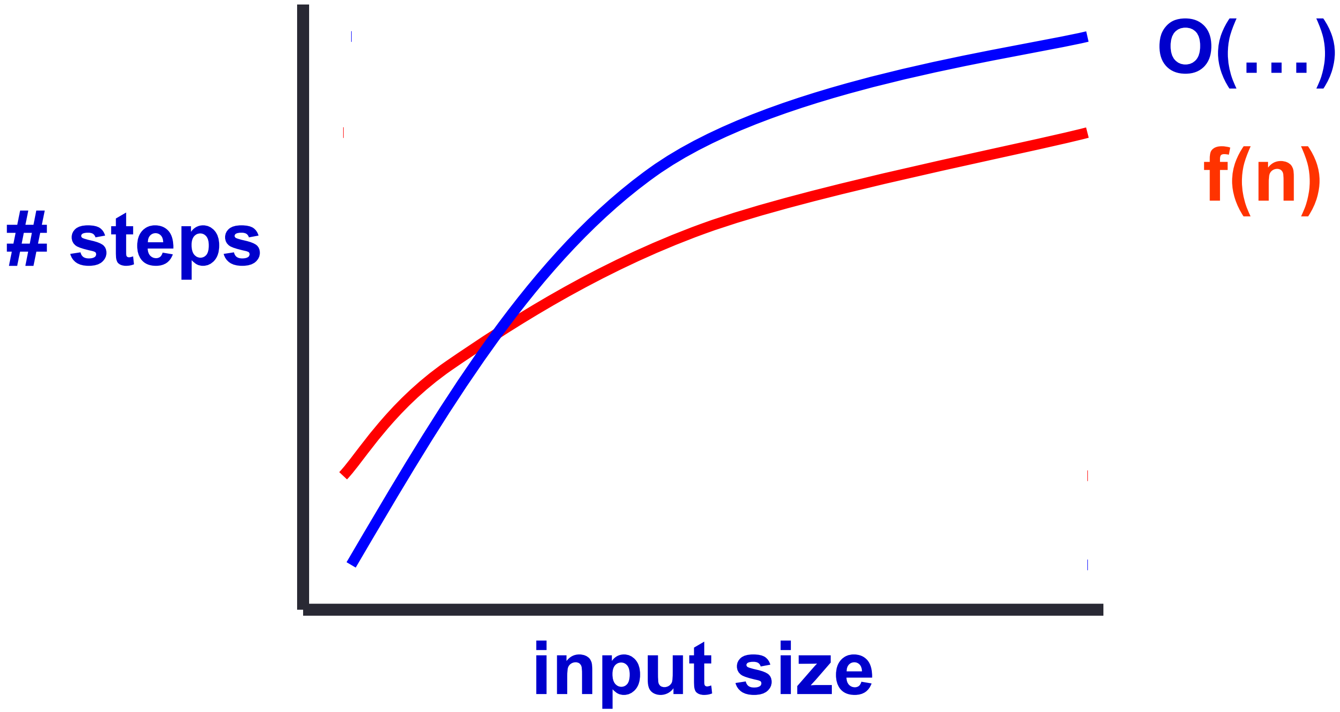
- Comparing two functions
  - log2(n) and 5 * log2(n) + 3 behave similarly
  - Run time roughly increases by constant as input size doubles
  - Run time increases logarithmically with input size
- For large values of n
  - Time(2n) – Time(n) approaches constant
  - Base of logarithm does not matter
    - Simply a multiplicative factor

    logaN = (logbN) / (logba)

    - Both are O(log(n)) programs

Big-O Notation

- Represents
  - Upper bound on number of steps in algorithm
    - For sufficiently large input size
  - Intrinsic efficiency of algorithm for large inputs



Formal Definition of Big-O

- Function f(n) is O(g(n)) if
  - For some positive constants M, N0
  - $M \times g(n) \geq f(n)$, for all $n \geq N0$
- Intuitively
  - For some coefficient M & all data sizes $\geq$ N0
    - $M \times g(n)$ is always greater than f(n)

Big-O Examples

- 2n2 + 10n + 1000 ⇒ O(n2)
  - Select M = 4, N0 = 100
  - For n ≥ 100
    - 4n2 ≥ 2n2 + 10n + 1000 is always true
  - Example ⇒ for n = 100
    - 40000 ≥ 20000 + 1000 + 1000

Observations

- For large values of n
  - Any O(log(n)) algorithm is faster than O(n)
  - Any O(n) algorithm is faster than O(n2)
- Asymptotic complexity is fundamental measure of efficiency
- Big-O results only valid for big values of n

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(n2)	Quadratic	2D Matrix addition
O(n3)	Cubic	2D Matrix multiply
O(nk)	Polynomial	Linear programming
O(kn)	Exponential	Integer programming
O(n!)	Factorial	Brute-force search TSP
O(nn)	N to the N	

From smallest to largest, for size n, constant k > 1

Complexity Category Example

Complexity Category Example

10

Calculating Asymptotic Complexity

- As **n** increases
  - Highest complexity term dominates
  - Can ignore lower complexity terms
- Examples
  - 2n + 100 ⇒ O(n)
  - 10n + nlog(n) ⇒ O(nlog(n))
  - 100n + ½n2 ⇒ O(n2)
  - 100n2 + n3 ⇒ O(n3)
  - 1/1002n + 100n4 ⇒ O(2n)

Types of Case Analysis

- Can analyze different types (cases) of algorithm behavior
- Types of analysis
  - Best case
  - Worst case
  - Average case
  - Amortized

Best/Worst Case Analysis

- Best case
  - Smallest number of steps required
  - Not very useful
  - Example ⇒ Find item in first place checked
- Worst case
  - Largest number of steps required
  - Useful for upper bound on worst performance
    - Real-time applications (e.g., multimedia)
    - Quality of service guarantee
  - Example ⇒ Find item in last place checked

Quicksort Example

- Quicksort
  - One of the fastest comparison sorts
  - Frequently used in practice
- Quicksort algorithm
  - Pick pivot value from list
  - Partition list into values smaller & bigger than pivot
  - Recursively sort both lists
- Quicksort properties
  - Average case = O(nlog(n))
  - Worst case = O(n2)
    - Pivot ≈ smallest / largest value in list
    - Picking from front of nearly sorted list
- Can avoid worst-case behavior
  - Select random pivot value

Average Case Analysis

- Average case analysis
  - Number of steps required for “typical” case
  - Most useful metric in practice
  - Different approaches: average case, expected case
- Average case
  - Average over all possible inputs
    - Assumes all inputs have the same probability
  - Example
    - Case 1 = 10 steps, Case 2 = 20 steps
    - Average = 15 steps
- Expected case
  - Weighted average over all possible inputs
    - Based on probability of each input
  - Example
    - Case 1 (90%) = 10 steps, Case 2 (10%) = 20 steps
    - Average = 11 steps

Amortized Analysis

- Approach
  - Applies to worst-case sequences of operations
  - Finds average running time per operation
  - Example
    - Normal case = 10 steps
    - Every 10th case may require 20 steps
    - Amortized time = 11 steps
- Assumptions
  - Can predict possible sequence of operations
  - Know when worst-case operations are needed
    - Does not require knowledge of probability
- By using amortized analysis we can show the best way to grow an array is by doubling its size (rather than increasing by adding one entry at a time)

Complexity Category Example

30

Complexity Category Example

10