Comparison of Algorithms

- How do we compare the efficiency of different algorithms?
- Comparing execution time: Too many assumptions, varies greatly between different computers
- Compare number of instructions: Varies greatly due to different languages, compilers, programming styles...

Big-O Notation

- The best way is to compare algorithms by the amount of work done in a critical loop, as a function of the number of input elements (*N*)
- **Big-O:** A notation expressing execution time (complexity) as the term in a function that increases most rapidly relative to *N*
- Consider the *order of magnitude* of the algorithm

Common Orders of Magnitude

- O(1): Constant or *bounded* time; not affected by *N* at all
- O($\log_2 N$): Logarithmic time; each step of the algorithm cuts the amount of work left in half
- O(N): Linear time; each element of the input is processed
- O($N \log_2 N$): $N \log_2 N$ time; apply a logarithmic algorithm N times or vice versa

Common Orders of Magnitude (cont.)

- $O(N^2)$: Quadratic time; typically apply a linear algorithm *N* times, or process every element with every other element
- $O(N^3)$: Cubic time; naive multiplication of two NxN matrices, or process every element in a three-dimensional matrix
- $O(2^N)$: Exponential time; computation increases dramatically with input size

What About Other Factors?

- Consider f(N) = $2N^4 + 100N^2 + 10N + 50$
- We can ignore $100N^2 + 10N + 50$ because $2N^4$ grows so quickly
- Similarly, the 2 in $2N^4$ does not greatly influence the growth
- The final order of magnitude is $O(N^4)$
- The other factors may be useful when comparing two very similar algorithms

Elephants and Goldfish

- Think about buying elephants and goldfish and comparing different pet suppliers
- The price of the goldfish is trivial compared to the cost of the elephants
- Similarly, the growth from $100N^2 + 10N + 50$ is trivial compared to $2N^4$
- The smaller factors are essentially noise

Example: Phone Book Search

- Goal: Given a name, find the matching phone number in the phone book
- Algorithm 1: Linear search through the phone book until the name is found
- Best case: O(1) (it's the first name in the book)
- Worst case: O(N) (it's the final name)
- Average case: The name is near the middle, requiring N/2 steps, which is O(N)

Example: Phone Book Search (cont.)

Algorithm 2: Since the phone book is sorted, we can use a more efficient search

1) Check the name in the middle of the book
2) If the target name is less than the middle name, search the first half of the book
3) If the target name is greater, search the last half
4) Continue until the name is found

Example: Phone Book Search (cont.)

Algorithm 2 Characteristics:

- Each step reduces the search space by half
- Best case: O(1) (we find the name immediately)
- Worst case: $O(\log_2 N)$ (we find the name after cutting the space in half several times)
- Average case: $O(\log_2 N)$ (it takes a few steps to find the name)

Example: Phone Book Search (cont.)

Which algorithm is better?

- For very small *N*, algorithm may be faster
- For target names in the very beginning of the phone book, algorithm 1 can be faster
- Algorithm 2 will be faster in every other case
- Success of algorithm 2 relies the fact that the phone book is sorted
  - Data structures matter!