CMPSC 250
Analysis of algorithms

Dr. Aravind Mohan
Allegheny College

February 5, 2018
Lecture 06: Algorithm Analysis
What is a good Algorithm?

- Efficient:
  - Running time
  - Space used

- Efficiency as a function of input size
  - The number of bits in an input number
  - Number of data elements (numbers, points)
Measure the Running Time

How should we measure the running time of an algorithm?

Experimental Study

- Write a program that implement the algorithm.
- Run the program with datasets of varying size and composition.
- Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time.
- Use Java applet based code such as STDDraw to plot a graph to visualize the experimental results.
Limitations of Experimental Studies

- It is necessary to implement and test the algorithm in order to determine the running time.
- Experiments can be done only on a limited set of inputs, and may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments should be used.
Analysis of Algorithms

**Primitive Operations:** Low level operations independent of programming language can be identified in pseudo-code. For example:

- Data movement (assign)
- Control (branch, subroutine call, return)
- Arithmetic and logical operation (addition, comparison)

By inspecting the pseudo code we can count the number of primitive operation executed by an algorithm.
Example: Sorting

Input: Sequence of numbers $a_1, a_2, a_3, \cdots, a_n$

Output: a permutation of the sequence of numbers $b_1, b_2, b_3, \cdots, b_n$
Example: Sorting

Correctness (requirements for the output)
For any given input the algorithm halts with the output

- $b_1 < b_2 < b_3 < \cdots < b_n$
- $b_1, b_2, b_3, \cdots, b_n$ is a permutation of $a_1, a_2, a_3, \cdots, a_n$
Example: Sorting

Running time depends on
- number of elements \((n)\)
- how (partially) sorted they are
- algorithm
Insertion Sort

Strategy:
- Start empty handed
- Insert a card in the right position of the already sorted hand
- Continue until all cards are inserted or sorted
Insertion Sort

INPUT: A [1, ..., n] – an array of integers

for j ← 2 to n do
    Key ← A[j]
    Insert A[j] into sorted sequence A[1, ..., j-1]
    i ← j - 1
    While i > 0 and A[i] > key
    do A[i+1] ← A[i]
    i←
    A[i+1] ← KEY
Let us analyze the algorithm

- line 1: cost:$c_1$, times:$n$
- line 2: cost:$c_2$, times:$n-1$
- line 3: cost:0, times:$n-1$
- line 4: cost:$c_3$, times:$n-1$
- line 5: cost:$c_4$, times:$\sum_{j=2}^{n} t_j$
- line 6: cost:$c_5$, times:$\sum_{j=2}^{n} t_j - 1$
- line 7: cost:$c_6$, times:$\sum_{j=2}^{n} t_j - 1$
- line 8: cost:$c_7$, times: $n-1$
Let us analyze the algorithm

- **Total time** = \( n \left( c_1 + c_2 + c_3 + c_7 \right) + \sum_{j=2}^{n} t_j \left( c_4 + c_5 + c_6 \right) - \left( c_2 + c_3 + c_5 + c_6 + c_7 \right) \)
Let us analyze the algorithm

- **Best Case**: elements already sorted; \( t_j = 1 \), running time = \( f(n) \), linear time.

- **Worst Case**: elements are sorted in inverse order; \( t_j = j \), running time = \( f(n^2) \), quadratic time.

- **Average Case**: \( t_j = j/2 \), running time = \( f(n^2) \), quadratic time.
Asymptotic Analysis

- Goal: to simplify analysis of running time by getting rid of "details", which may be affected by specific implementation and hardware.
  - like "rounding": $1,000,001 = 1,000,000$
  - $3n^2 = n^2$

- Capturing the essence: how the running time of an algorithm increases with the size of the input in the limit.
  - Asymptotically more efficient algorithm are best for all but small inputs.
Asymptotic Notation

- The big Oh notation O-Notation
  - asymptotic upper bound
  - \( f(n) = O(g(n)) \), if there exists constants \( c \) and \( n_0 \), such that
    \[ f(n) \leq cg(n) \text{ for } n \geq n_0 \]
  - \( f(n) \) and \( g(n) \) are functions over non negative integers

- Used for worst-case analysis
Asymptotic Notation

Simple Rule: Drop lower order terms and constant factors

- $50n \log n$ is $O(n \log n)$
- $7n - 3$ is $O(n)$
- $8n^2 \log n + 5n^2 + n$ is $O(n^2 \log n)$

Note: Even though $(50n \log n)$ is $O(n^5)$, it is expected that such an approximation be of as small an order as possible.
Questions