Analysis of Algorithms
CS 372

Lecture 1

Based on slides by Monica Nicolescu

Why Study Algorithms?

- Necessary in any computer programming problem
  - Improve algorithm efficiency: run faster, process more data, do something that would otherwise be impossible
  - Solve problems of significantly large sizes
  - Technology only improves things by a constant factor
- Compare algorithms
- Algorithms as a field of study
  - Learn about a standard set of algorithms
  - New discoveries arise
  - Numerous application areas
- Learn techniques of **algorithm design** and **analysis**

Applications

- Multimedia
  - CD player, DVD, MP3, JPG, DivX, HDTV
- Internet
  - Packet routing, data retrieval (Google)
- Communication
  - Cell-phones, e-commerce
- Computers
  - Circuit layout, file systems
- Science
  - Human genome
- Transportation
  - Airline crew scheduling, UPS deliveries

Roadmap

- Different problems
  - Sorting
  - Searching
  - String processing
  - Graph problems
  - Geometric problems
  - Numerical problems
- Different design paradigms
  - Divide-and-conquer
  - Dynamic programming
  - Greedy algorithms

4

Analyzing Algorithms

- Predict the amount of resources required:
  - memory: how much space is needed?
  - computational time: how fast the algorithm runs?
- FACT: running time grows with the size of the input
- Input size (number of elements in the input)
  - Size of an array, polynomial degree, # of elements in a matrix, # of bits in the binary representation of the input, # of vertices and edges in a graph

Def: Running time = the number of primitive operations (steps) executed before termination

  - Arithmetic operations (+, -, *), data movement, control, decision making (*if*, *while*), comparison take constant time

5

Algorithm Efficiency vs. Speed

E.g.: sorting n numbers Sort 10^6 numbers!

Friend's computer = 10^9 instructions/second
Friend's algorithm = $2n^2$ instructions

Your computer = 10^7 instructions/second
Your algorithm = $50n\lg n$ instructions

Your friend = $\frac{2*(10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}} = 2000 \text{seconds}$

You = $\frac{50*(10^6)\lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 100 \text{seconds}$

20 times better!!

6

Algorithm Analysis: Example

- *Alg.:* MIN (a[1], …, a[n])

```
m ← a[1];
for i ← 2 to n
    if a[i] < m
     then m ← a[i];
```

- **Running time**:
  - the number of primitive operations (steps) executed before termination
  - each line takes constant time

$T(n) = \ldots$

7

Order of growth

- Order (rate) of growth:
  - The leading term of the formula
  - Expresses the asymptotic behavior of the algorithm

$$T(n) = c1 + c2 \cdot n + c3 \cdot (n-1) + c4 \cdot (n-1) =$$
$$= n \cdot (c2 + c3 + c4) + (c1 - c3 - c4) =$$
$$= \Theta(n)$$

8

Typical Running Time Functions

- 1 (constant running time):
  - Instructions are executed once or a few times
- logN (logarithmic)
  - A big problem is solved by cutting the original problem in smaller sizes, by a constant fraction at each step
- N (linear)
  - A small amount of processing is done on each input element
- N logN
  - A problem is solved by dividing it into smaller problems, solving them independently and combining the solution

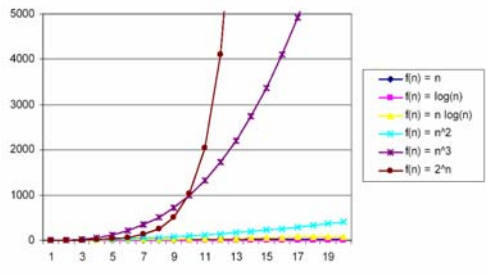
9

Typical Running Time Functions

- N^2 (quadratic)
  - Typical for algorithms that process all pairs of data items (double nested loops)
- N^3 (cubic)
  - Processing of triples of data (triple nested loops)
- N^K (polynomial)
- 2^N (exponential)
  - Few exponential algorithms are appropriate for practical use

10

Why Faster Algorithms?



11