CSE 373: Data Structures and Algorithms

Lecture 3: Math Review/Asymptotic Analysis
Motivation

• So much data!!
  – Human genome: $3.2 \times 10^9$ base pairs
    • If there are $6.8 \times 10^9$ on the planet, how many base pairs of human DNA?
  – Earth surface area: $1.49 \times 10^8$ km$^2$
    • How many photos if taking a photo of each m$^2$?
    • For every day of the year ($3.65 \times 10^2$)'

• But aren't computers getting faster and faster?
Why algorithm analysis?

• As problem sizes get bigger, analysis is becoming more important.

• The difference between good and bad algorithms is getting bigger.

• Being able to analyze algorithms will help us identify good ones without having to program them and test them first.
Measuring Performance: Empirical Approach

• Implement it, run it, time it (averaging trials)
  — Pros?

  — Cons?
Measuring Performance: Empirical Approach

• Implement it, run it, time it (averaging trials)
  – Pros?
    • Find out how the system effects performance
    • Stress testing – how does it perform in dynamic environment
    • No math!
  – Cons?
    • Need to implement code
    • Can be hard to estimate performance
    • When comparing two algorithms, all other factors need to be held constant (e.g., same computer, OS, processor, load)
Measuring Performance: Analytical Approach

• Use a simple model for basic operation costs

• Computational Model
  – has all the basic operations: 
    +, -, *, /, =, comparisons
  – fixed sized integers (e.g., 32-bit)
  – infinite memory
  – all basic operations take exactly one time unit (one CPU instruction) to execute
Measuring Performance: Analytical Approach

• Analyze steps of algorithm, estimating amount of work each step takes
  – Pros?
    • Independent of system-specific configuration
    • Good for estimating
    • Don't need to implement code
  – Cons?
    • Won't give you info exact runtimes optimizations made by the architecture (i.e. cache)
    • Only gives useful information for large problem sizes
    • In real life, not all operations take exactly the same time and have memory limitations
Analyzing Performance

• General “rules” to help measure how long it takes to do things:

  **Basic operations**  Constant time
  **Consecutive statements**  Sum of number of statements
  **Conditionals**  Test, plus larger branch cost
  **Loops**  Sum of iterations
  **Function calls**  Cost of function body
  **Recursive functions**  Solve recurrence relation…
Efficiency examples

statement1;
statement2;
statement3;

for (int i = 1; i <= N; i++) {
  statement4;
}

for (int i = 1; i <= N; i++) {
  statement5;
  statement6;
  statement7;
}
Efficiency examples

\[
\begin{align*}
\text{statement1;} \\
\text{statement2;} \\
\text{statement3;}
\end{align*}
\]

3

\[
\begin{align*}
\text{for (int } i = 1; i <= N; i++) \{ \\
\text{statement4;} \\
\}
\end{align*}
\]

N

\[
\begin{align*}
\text{for (int } i = 1; i <= N; i++) \{ \\
\text{statement5;} \\
\text{statement6;} \\
\text{statement7;} \\
\}
\end{align*}
\]

4N + 3

3N

10
Efficiency examples 2

```java
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        statement1;
    }
}

for (int i = 1; i <= N; i++) {
    statement2;
    statement3;
    statement4;
    statement5;
}
```
Efficiency examples 2

for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        statement1;
    }
}

for (int i = 1; i <= N; i++) {
    statement2;
    statement3;
    statement4;
    statement5;
}

• How many statements will execute if N = 10? If N = 1000?
Relative rates of growth

• most algorithms' runtime can be expressed as a function of the input size $N$

• rate of growth: measure of how quickly the graph of a function rises

• goal: distinguish between fast- and slow-growing functions
  – we only care about very large input sizes
    (for small sizes, most any algorithm is fast enough)
  – this helps us discover which algorithms will run more quickly or slowly, for large input sizes

• most of the time interested in worst case performance; sometimes look at best or average performance
Growth rate example

Consider these graphs of functions. Perhaps each one represents an algorithm:

\[ n^3 + 2n^2 \]
\[ 100n^2 + 1000 \]

- Which grows faster?
Growth rate example

• How about now?