csci 210: Data Structures

Program Analysis
Summary

• **Topics**
  - commonly used functions
  - analysis of algorithms
    • experimental
    • asymptotic notation
    • asymptotic analysis
    • big-O
    • big-Omega
    • big-theta

• **READING:**
  • GT textbook chapter 4
Analysis of Algorithms

- Goal: analyze the cost of a computer program
  - cost = time: How much time does this algorithm require?
  - cost = space: How much space (i.e. main memory) does this algorithm require?
  - cost = space + time
  - etc
- Analysis of algorithms and data structure is the major force that drives the design of solutions
  - there are many solutions to a problem
  - pick the one that is the most efficient
  - How to compare various algorithms?? Analyze algorithms.

- The primary efficiency concern for an algorithm is: time
  - all concepts that we discuss for time analysis apply also to space analysis

- running time of an algorithm
  - increases with input size
  - on inputs of same size, can vary from input to input
    - e.g.: linear search an un-ordered array
  - depends on hardware
    - CPU speed, hard-disk, caches, bus, etc
  - depends on OS, language, compiler, etc
Analysis of Algorithms

• Everything else being equal
  • we’d like to compare between algorithms
  • we’d like to study the relationship running time :-: size of input

• How to measure running time of an algorithm?
  • 1. experimental studies
  • 2. theoretical analysis

• Experimental studies
  • implement
  • chose various input sizes
  • for each input size, chose various inputs
    • run algorithm
    • time
    • compute average
    • plot
Experimental Analysis

- Limitations
  - need to implement the algorithm
    - need to implement all algorithms that we want to compare
  - need many experiments
  - need same platform

- Advantages
  - find the best algorithm in practice
  - We would like to analyze algorithms without having to implement them
  - Basically, we would like to be able to look at two algorithms flowcharts and decide which one is better
Theoretical Analysis

• Model:
  • Assume all operations cost the same
• Running time (efficiency) of an algorithm:
  • the number of operations executed by the algorithm

• Does this reflect actual running time?
  • multiply nb. of instructions by processor speed
    • 1GHz processor ==> $10^9$ instructions/second

• Is this accurate?
  • Not all instructions take the same...
  • various other effects
  • But, it is a very good predictor of running time in most cases
• **Notation:**
  • $n =$ size of the input to the problem

• **Running time:**
  • number of operations/instructions on an input of size $n$
  • expressed as function of $n$: $f(n)$

• For an input of size $n$, running time may be smaller on some inputs than on others
• **Best case running time:**
  • the smallest number of operations on an input of size $n$

• **Worst-case running time:**
  • the largest number of operations on an input of size $n$

• for any $n$
  • best-case running time($n$) $\leq$ running time($n$) $\leq$ worst-case running time ($n$)

• ideally, want to compute average-case running time
  • hard to model
Running times

- Expressed as functions of n: \( f(n) \)
- The most common functions for running times are the following:
  - constant time:
    - \( f(n) = c \)
  - logarithmic time
    - \( f(n) = \lg n \)
  - linear time
    - \( f(n) = n \)
  - \( n \lg n \)
    - \( f(n) = n \lg n \)
  - quadratic
    - \( f(n) = n^2 \)
  - cubic
    - \( f(n) = n^3 \)
  - exponential
    - \( f(n) = a^n \)
Constant time

- $f(n) = c$
  - Meaning: for any $n$, $f(n)$ is a constant $c$

- **Elementary operations**
  - arithmetic operations
  - boolean operations
  - assignment statement
  - function call
  - access to an array element $a[i]$
  - etc
Logarithmic time

- \( x = \log_b n \) if and only if \( b^x = n \)
- by definition, \( \log_b 1 = 0 \)

- \( \log_2 10 \)

- in algorithm analysis, compute \([\log_b n]\)
  - the ceiling (the smallest integer \( \geq \log_b n \))
  - \([\log_b n]\) is the number of times you can divide \( n \) by \( b \) until we get a number \( \leq 1 \)
  - e.g.
    - \([\log_2 8]\) = 3
    - \( \log_2 10 = 4 \)

- Notation: \( \lg n = \log_2 n \)

- Logarithm rules

- Note: computing \( \lg n \) on your calculator
  - \( \lg n = \log_{10} n / \log_{10} 2 \)
Exercises

- $\lg 2n =$
- $\lg (n/2) =$
- $\lg n^3 =$
- $\lg 2^n$
- $\log_4 n =$
- $2^{\lg n}$
Binary search

• searching a sorted array

```java
public static binarySearch(int[] a, int key) {
    int left = 0;
    int right = a.length-1;
    while (left <= right) {
        int mid = left + (right-left)/2;
        if (key < a[mid]) right = mid-1;
        else if (key > a[mid]) left = mid+1;
        else return mid;
    }
}
```

• running time:
  • best case: constant
  • worst-case: \( \lg n \)

Why? input size halves at every iteration of the loop
Linear running time

- $f(n) = n$

- **Example:**
  - doing one pass through an array of $n$ elements
  - e.g.
  - finding min/max/average in an array
  - computing sum in an array
  - search an un-ordered array (worst-case)

```java
int sum = 0
for (int i=0; i< a.length; i++)
    sum += a[i]
```
n-lg-n running time

- \( f(n) = n \lg n \)
- grows faster than \( n \) (i.e. it is slower than \( n \))
- grows slower than \( n^2 \)

Examples
- performing \( n \) binary searches in an ordered array
- sorting
Quadratic time

- $f(n) = n^2$
- appears in nested loops
- enumerating all pairs of $n$ elements

Examples:

```plaintext
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        //do something

//selection sort:
for (i=0; i<n; i++)
    minIndex = index-of-smallest element in a[i..n-1]
    swap a[i] with a[minIndex]
```

- **Running time:**
  - index-of-smallest element in $a[i..j]$ takes $j-i+1$ operations
  - $n + (n-1) + (n-2) + (n-3) + ... + 3 + 2 + 1$

  - this is $n^2$
• Lemma:
    • $1 + 2 + 3 + 4 + \ldots + (n-2) + (n-1) + n = \frac{n(n+1)}{2}$ (arithmetic sum)

• Proof:
• Cubic running time:  \( f(n) = n^3 \)
• In general, a polynomial running time is:  \( f(n) = n^d, \ d>0 \)
• Examples:
  • nested loops
  • Enumerate all triples of elements
  • Imagine cities on a map. Are there 3 cities that no two are not joined by a road?
    • Solution: enumerate all subsets of 3 cities. There are \( n \text{ chose } 3 \) different subsets, which is order \( n^3 \).

• Exponential running time:  \( f(n) = a^n, \ a > 1 \)
• Examples:
  • running time of Tower of Hanoi
    • moving \( n \) disks from A to B requires at least \( 2^n \) moves; which means it requires at least this much time
  • Exponent rules:
Comparing Growth-Rates

- $1 < \lg n < n < n \lg n < n^2 < n^3 < a^n$
Asymptotic analysis

- Focus on the growth of rate of the running time, as a function of $n$
- That is, ignore the constant factors and the lower-order terms
- Focus on the big-picture
- Example: we’ll say that $2n$, $3n$, $5n$, $100n$, $3n+10$, $n + \log n$, are all linear

- Why?
  - constants are not accurate anyways
  - operations are not equal
  - capture the dominant part of the running time

- Notations:
  - Big-Oh:
    - express upper-bounds
  - Big-Omega:
    - express lower-bounds
  - Big-Theta:
    - express tight bounds (upper and lower bounds)
Big-Oh

- **Definition:** $f(n)$ is $O(g(n))$ if there exists $c > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

- **Intuition:**
  - big-oh represents an upper bound
  - when we say $f$ is $O(g)$ this means that
    - $f \leq g$ asymptotically
    - $g$ is an upper bound for $f$
    - $f$ stays below $g$ as $n$ goes to infinity

- **Examples:**
  - $2n$ is $O(n)$
  - $100n$ is $O(n)$
  - $10n + 50$ is $O(n)$
  - $3n + \log n$ is $O(n)$
  - $\log n$ is $O(\log_{10} n)$
  - $\log_{10} n$ is $O(\log n)$
  - $5n^4 + 3n^3 + 2n^2 + 7n + 100$ is $O(n^4)$
Big-Oh

- $2n^2 + n \log n + n + 10$
  - is $O(n^2 + n \log n)$
  - is $O(n^3)$
  - is $O(n^4)$
  - is $O(n^2)$

- $3n + 5$
  - is $O(n^{10})$
  - is $O(n^2)$
  - is $O(n + \log n)$

- Let’s say you are 2 minutes away from the top and you don’t know that. You ask: How much further to the top?
  - Answer 1: at most 3 hours (True, but not that helpful)
  - Answer 2: just a few minutes.

- When finding an upper bound, find the best one possible.
Exercises

Write Big-Oh upper bounds for each of the following.

- $10n - 2$
- $5n^3 + 2n^2 + 10n + 100$
- $5n^2 + 3n \log n + 2n + 5$
- $20n^3 + 10n \log n + 5$
- $3n \log n + 2$
- $2^{n+2}$
- $2n + 100 \log n$
Big-Omega

- **Definition:**
  - $f(n)$ is Omega($g(n)$) if there exists $c > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

- **Intuition:**
  - Big-omega represents a lower bound
  - When we say $f$ is Omega($g$) this means that
    - $f \geq g$ asymptotically
    - $g$ is a lower bound for $f$
    - $f$ stays above $g$ as $n$ goes to infinity

- **Examples:**
  - $3n \log n + 2n$ is Omega($n \log n$)
  - $2n + 3$ is Omega($n$)
  - $4n^2 + 3n + 5$ is Omega($n$)
  - $4n^2 + 3n + 5$ is Omega($n^2$)
Big-Theta

• Definition:
  - $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f$ is $\Omega(g(n))$
  - i.e. there are constants $c'$ and $c''$ such that $c' g(n) \leq f(n) \leq c'' g(n)$

• Intuition:
  - $f$ and $g$ grow at the same rate, up to constant factors
  - $\Theta$ captures the order of growth

• Examples:
  - $3n + \log n + 10$ is $O(n)$ and $\Omega(n) \implies$ is $\Theta(n)$
  - $2n^2 + n \log n + 5$ is $\Theta(n^2)$
  - $3\log n + 2$ is $\Theta(\log n)$
Asymptotic Analysis

- Find tight bounds for the best-case and worst-case running times
  - running time is \( \Omega(\text{best-case running time}) \)
  - running time is \( O(\text{worst-case running time}) \)

- Example:
  - binary search is \( \Theta(1) \) in the best case
  - binary search is \( \Theta(\lg n) \) in the worst case
  - binary search is \( \Omega(1) \) and \( O(\lg n) \)

- Usually we are interested the worst-case running time
  - a \( \Theta \)-bound for the worst-case running time

- Example:
  - worst-case binary search is \( \Theta(\lg n) \)
  - worst-case linear search is \( \Theta(n) \)
  - worst-case insertion sort is \( \Theta(n^2) \)
  - worst-case bubble-sort is \( O(n^2) \)
  - worst-case find-min in an array is \( \Theta(n) \)

- Note: it is correct to say worst-case binary search is \( O(\lg n) \), but a \( \Theta \)-bound is better
Asymptotic Analysis

• Suppose we have two algorithms for a problem:
  • Algorithm A has a running time of $O(n)$
  • Algorithm B has a running time of $O(n^2)$

• Which is better?
Suppose we have two algorithms for a problem:

- Algorithm A has a running time of $O(n)$
  - assume the worst-case running time is $\Theta(n)$
- Algorithm B has a running time of $O(n^2)$
  - assume the worst-case running time is $\Theta(n^2)$

Which is better?
- order classes of functions
  - $\Theta(1) < \Theta(\log n) < \Theta(n) < \Theta(n \log n) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n)$
- $\Theta(n)$ is better than $\Theta(n^2)$
- etc

Cannot distinguish between algorithms in the same class
- two algorithms that are $\Theta(n)$ worst-case are equivalent theoretically
- optimization of constants can be done at implementation-time
Growth-rate matters

- Example:
  - Say \( n = 10^9 \) (1 billion elements)
  - 10 MHz computer \( \implies \) 1 instruction takes \( 10^{-7} \) seconds
  - Binary search would take
    - \( \Theta(\lg n) = \lg 10^9 \times 10^{-7} \text{ sec} = 30 \times 10^{-7} \text{ sec} = 3 \text{ microsec} \)
  - Sequential search would take
    - \( \Theta(n) = 10^9 \times 10^{-7} \text{ sec} = 100 \text{ seconds} \)
  - Finding all pairs of elements would take
    - \( \Theta(n^2) = (10^9)^2 \times 10^{-7} \text{ sec} = 10^{11} \text{ seconds} = 3170 \text{ years} \)
  
- Imagine \( \Theta(n^3) \)

- Imagine \( \Theta(2^n) \)
### Efficiency matters

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\lg n$</th>
<th>$n$</th>
<th>$n \lg n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>4,096</td>
<td>65,536</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1,024</td>
<td>32,768</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4,096</td>
<td>262,144</td>
<td>1.8 x 10^19</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>128</td>
<td>896</td>
<td>16,384</td>
<td>2,097,152</td>
<td>3.40 x 10^38</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>256</td>
<td>2,048</td>
<td>65,536</td>
<td>16,777,216</td>
<td>1.15 x 10^77</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
<td>512</td>
<td>4,608</td>
<td>262,144</td>
<td>134,217,728</td>
<td>1.34 x 10^154</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
<td>1024</td>
<td>262,144</td>
<td>1024^2</td>
<td>1,048,576</td>
<td></td>
</tr>
<tr>
<td>$10^{9}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Assume we have a 1 GHz computer.
• This means an instruction takes 1 microsecond ($10^{-9}$ seconds).

• We have 3 algorithms:
• A: $400n$
• B: $2n^2$
• C: $2^n$

• What is the maximum input size that can be solved with each algorithm in:
  • 1 second
  • 1 minute
  • 1 hour

<table>
<thead>
<tr>
<th>Running time (in microseconds)</th>
<th>1 sec</th>
<th>1 min</th>
<th>1 hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>$400n$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2n^2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Problem**

- We have an array \( X \) containing a sequence of numbers. We want to compute another array \( A \) such that \( A[i] \) represents the average \( X[0] + X[1] + \ldots + X[i] \)/ (i+1).

  - \( A[0] = X[0] \)
  - \( A[1] = (X[0] + X[1])/ 2 \)
  - \( \ldots \)

- The first \( i \) values of \( X \) are referred to as the \( i \)-prefix of \( X \). \( X[0] + \ldots + X[i] \) is called prefix-sum, and \( A[i] \) prefix average.

- Application: In Economics. Imagine that \( X[i] \) represents the return of a mutual fund in year \( i \). \( A[i] \) represents the average return over \( i \) years.

- Write a function
  
  ```java
double[] computePrefixAverage(double[] X)
```

  that creates, computes and returns the prefix averages.

- Analyze your algorithm (worst-case running time).
Asymptotic Analysis: Overview

- running time = number of instructions
  - assume all instructions are equal

- usually interested in worst-case running time as a function of input size
  - the largest number of instructions on an input of size n

- find the order of growth of the worst-case running time
  - a Theta-bound

- classification of growth rates
  - \( \Theta(1) < \Theta(\log n) < \Theta(n) < \Theta(n \log n) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n) \)

- At the algorithm design level, we want to find the most efficient algorithm in terms of growth rate
- We can optimize constants at the implementation step