CSE 326: Data Structures
Asymptotic Analysis

Hannah Tang and Brian Tjaden
Summer Quarter 2002

Today’s Outline

• How’s the project going?
• Finish up stacks, queues, lists, and bears, oh my!
• Math review and runtime analysis
• Pretty pictures
• Asymptotic analysis

Analyzing Algorithms: Why Bother?

From “Programming Pearls”, by Jon Bentley
Communications of the ACM, Nov 1984

Analyzing Algorithms

• Computer scientists analyze algorithms to precisely characterize an algorithm’s:
  – Time complexity (running time)
  – Space complexity (memory use)

• This allows us to get a better sense of the various tradeoffs between several algorithms
  – For instance, do we know how complex the 1984 algorithm is, compared to the 1945 algorithm?

A problem’s input size is indicated by a number \( n \)
  – Sometimes have multiple inputs, e.g. \( m \) and \( n \)

• The running time of an algorithm is a function of \( n \)
  – \( n \), \( 2^n \), \( n \log n \), \( 18 + 3n(\log n)^2 + 5n^3 \)

Hannah Takes a Break

bool ArrayFind(int array[], int n, int key)
{
  // Insert your algorithm here
}

What algorithm would you choose to implement this code snippet?

Hannah Takes a Break: Simplifying assumptions

• Ideal single-processor machine (serialized operations)
• “Standard” instruction set (load, add, store, etc.)
• All operations take 1 time unit (including, for our purposes, each Java or C++ statement
HTaB: Analyzing Code

- Basic Java/C++ operations
- Consecutive statements
- Conditionals
- Loops
- Function calls
- Recursive functions

HTaB: Linear Search Analysis

```java
bool ArrayFind( int array[], int n, int key )
{
    for( int i = 0; i < n; i++ )
    {
        // Found it!
        if( array[i] == key )
            return true;
    }
    return false;
}
```

• Exact Runtime:
• Best Case:
• Worst Case:

HTaB: Binary Search Analysis

```java
bool ArrayFind( int array[], int s, int e, int key ) {
    // The subarray is empty
    if( e - s <= 0 )
        return false;
    // Search this subarray
    int mid = (e - s) / 2;
    if( array[key] == array[mid] ) {
        return true;
    } else if( key < array[mid] ) {
        return ArrayFind( array, s, mid, key );
    } else {
        return ArrayFind( array, mid, e, key );
    }
}
```

• Exact Runtime:
• Best case:
• Worst case:

Back to work:
Solving Recurrence Relations

1. Determine the recurrence relation. What are the base case(s)?
2. “Expand” the original relation to find an equivalent general expression in terms of the number of expansions.
3. Find a closed-form expression by setting the number of expansions to a value which reduces the problem to a base case

Linear Search vs Binary Search

<table>
<thead>
<tr>
<th></th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst Case</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fast Computer vs. Slow Computer

So ... which algorithm is best?
What the tradeoffs did you make?
Fast Computer vs. Smart Programmer (round 1)

### Asymptotic Analysis

- Asymptotic analysis looks at the order of the running time of the algorithm
  - A valuable tool when the input gets “large”
  - Ignores the effects of different machines or different implementations of the same algorithm
- Intuitively, to find the asymptotic runtime, throw away the constants and low-order terms
  - Linear search: $T(n) = n \in O(n)$
  - Binary search: $T(n) = 4 \log_2 n + 1 \in O(\log n)$

**Order Notation: Intuition**

Although not yet apparent, as $n$ gets “sufficiently large”, $f(n)$ will be “greater than or equal to” $g(n)$

**Order Notation: Definition**

$O(f(n))$ is a set of functions

$g(n) \in O(f(n)) \iff$

There exist $c$ and $n_0$, such that $g(n) \leq c f(n)$

for all $n \geq n_0$

Example:

$100n^2 + 1000 \leq 5(n^3 + 2n^2)$ for all $n \geq 19$

So $g(n) \in O(f(n))$

Sometimes, you’ll see the notation $g(n) = O(f(n))$. This equivalent to $g(n) \in O(f(n))$. However, the notation $O(f(n)) = g(n)$ is not correct

**Order Notation: Example**

$100n^2 + 1000 \leq 5(n^3 + 2n^2)$ for all $n \geq 19$

So $g(n) \in O(f(n))$
“\( O(f(n)) \) is a set of functions”

\[
\begin{align*}
2n^2 + 10 & \in O(n^3) \\
10n^2 \log n & \in O(n^3)
\end{align*}
\]

So we say both:

\[
\begin{align*}
100n^2 \log n = O(n^3) & \quad \text{and} \\
100n^2 \log n \in O(n^3)
\end{align*}
\]

Big-O Common Names

- constant: \( O(1) \)
- logarithmic: \( O(\log n) \)
- linear: \( O(n) \)
- log-linear: \( O(n \log n) \)
- superlinear: \( O(n^{1+c}) \) (c is a constant, where \( 0 < c < 1 \))
- quadratic: \( O(n^2) \)
- polynomial: \( O(n^k) \) (k is a constant)
- exponential: \( O(c^n) \) (c is a constant > 1)

Meet the Family

- \( O(f(n)) \) is the set of all functions asymptotically less than or equal to \( f(n) \)
- \( o(f(n)) \) is the set of all functions asymptotically strictly less than \( f(n) \)
- \( \Omega(f(n)) \) is the set of all functions asymptotically greater than or equal to \( f(n) \)
- \( \omega(f(n)) \) is the set of all functions asymptotically strictly greater than \( f(n) \)
- \( \Theta(f(n)) \) is the set of all functions asymptotically equal to \( f(n) \)

Mathematics Relation

<table>
<thead>
<tr>
<th>Asymptotic Notation</th>
<th>Mathematics Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O )</td>
<td>( \leq )</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>( \geq )</td>
</tr>
<tr>
<td>( \Theta )</td>
<td>( = )</td>
</tr>
<tr>
<td>( o )</td>
<td>( &lt; )</td>
</tr>
<tr>
<td>( \omega )</td>
<td>( &gt; )</td>
</tr>
</tbody>
</table>

Meet the Family Formally

- \( g(n) \in O(f(n)) \) if and only if:
  - There exist \( c \) and \( n_0 \) such that \( g(n) \leq c f(n) \) for all \( n \geq n_0 \)
  - \( g(n) \in o(f(n)) \) if:
    - There is \( n_0 \) such that \( g(n) < c f(n) \) for all \( c \) and \( n \geq n_0 \)
- \( g(n) \in \Omega(f(n)) \) if and only if:
  - There exist \( c \) and \( n_0 \) such that \( g(n) \geq c f(n) \) for all \( n \geq n_0 \)
  - \( g(n) \in \omega(f(n)) \) if:
    - There is \( n_0 \) such that \( g(n) > c f(n) \) for all \( c \) and \( n \geq n_0 \)
- \( g(n) \in \Theta(f(n)) \) if and only if:
  - \( g(n) \in O(f(n)) \) and \( g(n) \in \Omega(f(n)) \)
**True or False?**

- $10,000 n^2 + 25n \in \Theta(n^2)$
- $10^{10} n^3 \in \Theta(n^3)$
- $n^3 + 4 \in \Theta(n^2)$
- $n \log n \in \Theta(2^n)$
- $n \log n \in \Omega(n)$
- $n^3 + 4 \in \omega(n^2)$

**Another Kind of Analysis**

- **Runtime may depend on actual input, not just length of input**
- **Analysis based on input type:**
  - Worst case
    - Your worst enemy is choosing input
  - Average case
    - Assume a probability distribution of inputs
  - Best case
    - Not too useful
- **Amortized analysis**
  - Runtime over many runs, regardless of underlying probability for inputs

**HTaB: Pros and Cons of Asymptotic Analysis**

**To Do**

- Start project 1
  - Due Monday, July 1st at 10 PM sharp!
- Sign up for 326 mailing list(s)
  - Don’t forget to use the new web interfaces!
- Prepare for tomorrow’s quiz
  - Possible topics:
    - Math concepts from 321 (skim section 1.2 in Weiss)
    - Lists, stacks, queues, and the tradeoffs between various implementations
    - Whatever asymptotic analysis stuff we covered today
    - Possible middle names for Brian C. Tjaden, Hannah C. Tang, and Albert J. Wong
- Read chapter 2 (algorithm analysis), section 4.1 (introduction to trees), and sections 6.1-6.4 (priority queues and binary heaps)