The Efficiency of Algorithms
An Introduction

Prof. David Bernstein
James Madison University
Computer Science Department
bernstdh@jmu.edu

An Obvious Way to Measure Efficiency

- The Process:
  - Write the necessary code
  - Measure the running time and space requirements

- An Important Caveat:
  - It would be a big mistake to run the application only once
  - Instead, one must develop a sampling plan (i.e., a way of developing/selecting sample inputs) and test the application on a large number of inputs

Problems with This Approach

- It is often very difficult to test algorithms on a large number of representative inputs.
- The efficiency can't be known a priori and it if often prohibitively expensive to implement and test an algorithm only to learn that it is not efficient.

An Alternative Approach: Bounds

- The Idea:
  - Obtain an upper and/or lower bound on the time and space requirements

- An Observation:
• Bounds are used in a variety of contexts

Bounds (cont.)

• A Common Question at Job Interviews:
  o How many hairs do you have on your head?

• Obtaining a Bound:
  o Suppose a human hair occupies at least a square of 100 microns on a side (where one micron is one millionth of a meter)
  o Then, each hair occupies at least 0.00000001 m² in area
  o Suppose further that your head has at most 400 cm² in total area (or 0.04 m²)
  o Then, your head has at most 4 million hairs

Bounds (cont.)

• Worst Case Analysis:
  o Instead of trying to determine the "actual" performance experimentally, one instead attempts to find theoretical upper bounds on performance

• Some Notation:
  o The worst case running time for an input of size \( n \) is denoted by \( T(n) \)
  o The worst case space requirement for an input of size \( n \) is denoted by \( S(n) \)

Comparing Different Algorithms

• Which is Better?
  o An algorithm with worst case time of \( n^2 + 5 \)
  o An algorithm with worst case time of \( n^2 + 2 \)

• Do You Care When \( n = 1000 \)?
  o 1,000,005
  o 1,000,002

Asymptotic Dominance

• The Idea:
  o When comparing algorithms based on their worst case time or space it seems best to consider their asymptotic performance (i.e., their performance on "large" problems)

• Applying this Idea:
  o Consider the limit of the worst case time or space
I Hate Math - Does Efficiency Really Matter?

- Compare Two Algorithms When:
  
  - One requires $n^3$ iterations and one requires $2n$ iterations
  - Each iteration requires one microsecond (i.e., one millionth of a second)

- Different Problem Sizes:

<table>
<thead>
<tr>
<th>$T(n)$</th>
<th>$n = 10$</th>
<th>$n = 25$</th>
<th>$n = 50$</th>
<th>$n = 75$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3$</td>
<td>0.001 sec</td>
<td>0.016 sec</td>
<td>0.125 sec</td>
<td>0.422 sec</td>
</tr>
<tr>
<td>$2n$</td>
<td>0.001 sec</td>
<td>33.554 sec</td>
<td>35.702 yrs</td>
<td>11,979,620.707 centuries</td>
</tr>
</tbody>
</table>

Landau Numbers

- Defining "little o":

  - $T = o[f(n)]$ iff $\lim_{n \to \infty} T(n)f(n) = 0$

- An Example: $n^2 + 5$

  - "little o" of $n^3$ since $\lim_{n \to \infty} [(n^2 + 5)n^3] = 0$

"big O" Notation

- Interpreting "little o":

  - When $T$ is "little o" of $f$ it means that $T$ is of lower order of magnitude than $f$

- Defining "big O":

  - $T = O[f(n)]$ iff there exist positive $k$ and $n_0$ such that $T(n) \leq kf(n)$ for all $n \geq n_0$

- Interpreting "big O":

  - $T$ is not of higher order of magnitude than $f$

"big O" Notation (cont.)

- Show:

  - $n^2 + 5$ is $O(n^2)$

- Choose $k = 3$ and $n_0 = 2$ and observe:

  - $n^2 + 5 \leq 3n^2$ for all $n \geq 2$

Lower Bounds on Growth Rates

- Defining "Big Omega":
• T = \Omega \left[ g \left( n \right) \right] \text{iff there exists a positive constant } c \text{ such that } T \left( n \right) \geq c \, g \left( n \right) \text{ for an infinite number of values of } n

• An Example: \( n^2 + 5 \)
  
  o Is \( \Omega \left( n^2 \right) \) since, setting \( c = 1 \):
  
  \[ n^2 + 5 \geq n^2 \text{ for all } n \geq 0 \]

### Categorizing Performance

<table>
<thead>
<tr>
<th>Asymptotic Bound</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O \left( 1 \right) )</td>
<td>Constant</td>
</tr>
<tr>
<td>( O \left( \log n \right) )</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>( O \left( n \right) )</td>
<td>Linear</td>
</tr>
<tr>
<td>( O \left( n^2 \right) )</td>
<td>Quadratic</td>
</tr>
<tr>
<td>( O \left( n^3 \right) )</td>
<td>Cubic</td>
</tr>
<tr>
<td>( O \left( a \cdot n \right) )</td>
<td>Exponential</td>
</tr>
<tr>
<td>( O \left( n! \right) )</td>
<td>Factorial</td>
</tr>
</tbody>
</table>

### Determining Asymptotic Bounds

1. If \( T_1 = O \left[ f_1 \left( m \right) \right] \) then \( k \, T_1 = O \left[ f_1 \left( m \right) \right] \) for any constant \( k \)

2. If \( T_1 = O \left[ f_1 \left( m \right) \right] \) and \( T_2 = O \left[ f_2 \left( m \right) \right] \) then \( T_1 \, T_2 = O \left[ f_1 \left( m \right) \right] \, O \left[ f_2 \left( m \right) \right] \)

3. If \( T_1 = O \left[ f_1 \left( m \right) \right] \) and \( T_2 = O \left[ f_2 \left( m \right) \right] \) then \( T_1 + T_2 = \max \left\{ O \left[ f_1 \left( m \right) \right], O \left[ f_2 \left( m \right) \right] \right\} \)

### Determining Asymptotic Bounds (cont.)

Suppose an algorithm has three "steps" with running times of \( O \left( n^4 \right) \), \( O \left( n^2 \right) \), and \( O \left( \log n \right) \).

Then, it follows from rule 3 that the running time for the entire algorithm is \( O \left( n^4 \right) \).

### Determining Asymptotic Bounds (cont.)

Suppose an algorithm has one "step" with a running time of \( O \left( n \right) \) and that it repeats this "step" (in a loop) 1000 times.

Then, it follows from rule 1 that the running time for the entire algorithm is \( O \left( n \right) \).

### An Example: Factorials

An Example: Factorials
What is the asymptotic bound on the worst case time efficiency of the following recursive algorithm?

```c
int factorial(int n)
{
    // Check for valid input
    if (n > 12) return -1;

    if ((n == 0) || (n == 1)) return 1;

    return n*factorial(n-1);
}
```

**An Example: Factorials (cont.)**

What is the asymptotic bound on the worst case time efficiency of the following iterative algorithm?

```c
int factorial(int n)
{
    int value = 1;
    for (i=2; i<=n; i++) {
        value *= i;
    }

    return value;
}
```

**An Example: Factorials (cont.)**
What is the asymptotic bound on the worst case time efficiency of the following algorithm? What about the space efficiency?

```c
int factorial(int n)
{
    int f[13]={1,1,2,6,24,120,720,5040,40320,
                362880,3628800,39916800,479001600};

    // Check for valid input
    if (n > 12) return -1;

    return f[n];
}
```