Algorithm Analysis - CS 312

Professor Tony Martinez
Algorithm Analysis - Class Goals

- Analysis of Algorithms from both practical (programming) and theoretical (mathematical) points of view
- Efficiency of Algorithms - Time and Space
- Explore Fundamental Algorithmic Paradigms
  - Divide and Conquer
  - Graph Exploration
  - Greedy Algorithms
  - Dynamic Programming
  - Linear Programming
  - Local Search
  - Intelligent Search
  - Probabilistic Algorithms
Additional Goals

- Develop your general problem solving and programming skills
- Use Visual Studio and C#
- Become a better person - Always remember the big picture
Before we ask what is an Algorithm, We ask:

What is Computation?
What is Computation?

\[ F(X) = Y \]

Input
X

Output
Y
Deterministic mappings

- What Features would we like for such computational machines
- Humans?, Machine vs Human Intelligence?, come to CS 478
What is an Algorithm?

- No single formal definition
- A typical informal definition: “A finite sequence of well defined steps for solving a problem.”
- This definition has been influenced by the types of computational machines we have used.
- Is all computation done by algorithms so defined?
  - Sequence of operations?
  - Termination?
3 Basic Questions about any Algorithm

Start with a problem and then propose an algorithmic solution:

1. Is it Correct (not as obvious as it sounds)?
2. How long does it take as a function of \( n \), the problem size?
   - Can also ask how much space (memory) does it require
   - Average, worst, and best case
3. Can we do Better?
   - Efficiency - Time and Space
   - Correctness/Accuracy
   - Simplicity - Intelligibility
   - Etc.

Example - Is a certain name in a list of names of length \( n \)?
A Bit of History

- Abu Jafar Muhammad ibn Musa, c. 780 - 850
  - “Al Khwarizmi” (of Kwarizm) his place of origin (Uzbekistan)
  - Persian scholar in Baghdad
  - Used decimal (positional system as contrasted with Roman numerals) and proposed numerous arithmetic algorithms
  - Latin version of this name is Algoritmi
  - From this came the word Algorithm

- Leonardo of Pisa (Fibonacci) 1170-1250
  - Pushed the use of decimal system in Europe
  - Most remembered for the Fibonacci series of numbers
Fibonacci Series

\[ F_n = \begin{cases} 
F_{n-1} + F_{n-2} & \text{if } n > 1 \\
1 & \text{if } n = 1 \\
0 & \text{if } n = 0 
\end{cases} \]

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
An Algorithm - fib1

Function for computing the \( n \)-th Fibonacci number \( F_n \)

```
function fib1(n)
  if n = 0: return 0
  if n = 1: return 1
  return fib1(n-1) + fib1(n-2)
```

- Is it correct?
- How much time does it take, as a function of \( n \)?
- Can we do better?
How long does it take

- Note that with each call there are two more calls (doubling each time)
- What is the depth of the calling tree
- Thus exponential time
- Since tree is binary, but not full it is somewhat less than, but close to $O(2^n)$
- How bad is that?
How long does it take

- Note that with each call there are two more calls (doubling each time)
- What is the depth of the calling tree
- Thus exponential time
- Since tree is binary, but not full it is somewhat less than, but close to $O(2^n)$
- How bad is that? Calculating $F(200)$ takes longer on current fastest computer than the life of the sun.
- Moore’s Law?
- Parallelism?
- Exponential is Intractable!
How long does it take - Bit more Careful

- Assume $T(n)$ is the time necessary to compute $F(n)$ for fib1. What is the recurrence relation?
- $T(n) =$ ?

```python
function fib1(n)
    if n = 0: return 0
    if n = 1: return 1
    return fib1(n-1) + fib1(n-2)
```
How long does it take - Bit more Careful

- Assume $T(n)$ is the time necessary to compute $F(n)$ for fib1. What is the recurrence relation?
- $T(n) = T(n-1) + T(n-2) + 3$ when $n > 1$

```python
function fib1(n)
    if n = 0: return 0
    if n = 1: return 1
    return fib1(n-1) + fib1(n-2)
```
How long does it take - a Bit more Careful

- Assume $T(n)$ is the time necessary to compute $F(n)$ for fib1. What is the recurrence relation?
- $T(n) = T(n-1) + T(n-2) + 3$ when $n > 1$
- Note that $T(n) > F(n)$
- $F(n) \approx 2^{0.694n} \approx 1.6^n$, where .694 are the first 3 digits of an irrational number
- $T(n) \approx 1.6^n$
- We will show how to get the exact value for $T(n)$ in this type of case in our discussion of recurrence relations in a couple of weeks
Can we do better?
Approximation

- Sometimes we can find a closed-form solution
- $\text{Fib}(n) \approx 2^{0.694n} \approx 1.6^n$, where .694 are the first 3 digits of an irrational number
- This direct calculation $\text{Fib}(n) \approx 2^{0.694n}$ trades off correctness for speed
- Approximate solution – very common for many problems as we shall see later
Can we do better?
Can we do better?

- Store the intermediate results and avoid the doubling

```python
function fib2(n)
    if n=0: return 0
    create an array f[0..n]
    f[0] = 0, f[1] = 1
    for i = 2 to n:
        f[i] = f[i-1] + f[i-2]
    return f[n]
```

- Dynamic Programming approach
- Space complexity? - Tradeoffs
- Ask the questions
Asymptotic Complexity - Big-O analysis

- We want to consider significant algorithm differences
- Independent of the particular machine
- Consider time complexity in terms of number of elementary operations (we will define more carefully) and as a function of input size \( n \).
- Space complexity in terms of number of bits required
- \( \Theta(n) \) includes: \( 2n, 4n+1 \), etc.
- Forms an equivalence class
- Are these constants important in real applications?
- We will focus in this class on paradigms which allow you to jump all the way into more efficient complexity classes
# Orders of Growth – Some Complexity Classes

<table>
<thead>
<tr>
<th>$n$</th>
<th>log$_2n$</th>
<th>$n$</th>
<th>$n\log_2n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>10</td>
<td>3.3*10</td>
<td>10$^2$</td>
<td>10$^3$</td>
<td>10$^3$</td>
<td>3.6*10$^6$</td>
</tr>
<tr>
<td>100</td>
<td>6.6</td>
<td>10$^2$</td>
<td>6.6*10$^2$</td>
<td>10$^4$</td>
<td>10$^6$</td>
<td>1.3*10$^{30}$</td>
<td>9*10$^{157}$</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>10$^3$</td>
<td>1.0*10$^4$</td>
<td>10$^6$</td>
<td>10$^9$</td>
<td>1.1*10$^{301}$</td>
<td>4*10$^{2567}$</td>
</tr>
<tr>
<td>10000</td>
<td>13</td>
<td>10$^4$</td>
<td>1.3*10$^5$</td>
<td>10$^8$</td>
<td>10$^{12}$</td>
<td>2.0*10$^{3010}$</td>
<td>2.8*10$^{35659}$</td>
</tr>
<tr>
<td>10$^5$</td>
<td>17</td>
<td>10$^5$</td>
<td>1.7*10$^6$</td>
<td>10$^{10}$</td>
<td>10$^{15}$</td>
<td>10$^{30102}$</td>
<td>2.9*10$^{45673}$</td>
</tr>
<tr>
<td>10$^6$</td>
<td>20</td>
<td>10$^6$</td>
<td>2.0*10$^7$</td>
<td>10$^{12}$</td>
<td>10$^{18}$</td>
<td>10$^{301029}$</td>
<td>8.3*10$^{556708}$</td>
</tr>
</tbody>
</table>

Efficient | Not Efficient
# Orders of Growth – Some Complexity Classes

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log_2 n$</th>
<th>$n$</th>
<th>$n \log_2 n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>10</td>
<td>3.3*10</td>
<td>10²</td>
<td>10³</td>
<td>10³</td>
<td>3.6*10⁶</td>
</tr>
<tr>
<td>100</td>
<td>6.6</td>
<td>10²</td>
<td>6.6*10²</td>
<td>10⁴</td>
<td>10⁶</td>
<td>1.3*10³⁰</td>
<td>9*10¹⁵⁷</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>10³</td>
<td>1.0*10⁴</td>
<td>10⁶</td>
<td>10⁹</td>
<td>1.1*10³⁰¹</td>
<td>4*10²⁵⁶⁷</td>
</tr>
<tr>
<td>10000</td>
<td>13</td>
<td>10⁴</td>
<td>1.3*10⁵</td>
<td>10⁸</td>
<td>10¹²</td>
<td>2.0*10³⁰¹⁰</td>
<td>2.8*10³⁵⁶⁵⁹</td>
</tr>
<tr>
<td>10⁵</td>
<td>17</td>
<td>10⁵</td>
<td>1.7*10⁶</td>
<td>10¹⁰</td>
<td>10¹⁵</td>
<td>10³⁰¹⁰²</td>
<td>2.9*10⁴⁵⁶⁵⁷³</td>
</tr>
<tr>
<td>10⁶</td>
<td>20</td>
<td>10⁶</td>
<td>2.0*10⁷</td>
<td>10¹²</td>
<td>10¹⁸</td>
<td>10³⁰¹⁰²⁹</td>
<td>8.3*10⁵⁵⁵⁶⁵⁷⁰⁸</td>
</tr>
</tbody>
</table>

To calibrate, there are about $10^{57}$ atoms in the solar system and $10^{80}$ atoms in the current known universe.
Asymptotic Notation

Definition: given asymptotically nonnegative \( g(n) \),

\[
\Theta(g(n)) = \{ f(n) : \exists (c_1, c_2, n_0) > 0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0 \}
\]

An asymptotically nonnegative function \( f(n) \) belongs to the set \( \Theta(g(n)) \) if there exist positive constants \( c_1 \) and \( c_2 \) such that it can be “sandwiched” between \( c_1 g(n) \) and \( c_2 g(n) \) for sufficiently large \( n \).

Write \( f(n) = \Theta(g(n)) \) for \( f(n) \in \Theta(g(n)) \)
Examples

- Give some example $f(n) = \Theta(g(n)) = \Theta(n)$
- How would you prove that $f(n) = \Theta(n)$
- Note that $\Theta$ represents an equivalence class
  - if $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$
- Could we represent $\Theta(n)$ as $\Theta(3+n/2)$
  - Why don't we?

$$\Theta(g(n)) = \{ f(n) : \exists (c_1,c_2,n_0) > 0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0 \}$$
Asymptotic Notation

Definition: given asymptotically nonnegative $g(n)$,

$\Theta(g(n)) = \{ f(n) : \exists (c_1, c_2, n_0) > 0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0 \}$

$O(g(n)) = \{ f(n) : \exists (c, n_0) > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \forall n \geq n_0 \}$

$\Omega(g(n)) = \{ f(n) : \exists (c, n_0) > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \forall n \geq n_0 \}$
Asymptotic Notation

\[ f(n) = \Theta(g(n)) \]

\[ f(n) = O(g(n)) \]

\[ f(n) = \Omega(g(n)) \]
Asymptotic Analysis

- O and \( \Omega \) are sets but not equivalence classes like \( \Theta \).
- O\( (n) \) includes: \( 2n, \log(n), 4, \frac{1}{n}, \) etc. Thus, \( f = O(g) \) is like saying \( f \leq g \) in terms of Big-O complexity.
  - Thus \( \log(n) \in O(n) \), but \( n \notin O(\log(n)) \).
- \( f = \Omega(g) \) iff \( g = O(f) \), i.e. \( f \geq g \) (Duality Rule).
- \( f = \Theta(g) \) means that \( g = O(f) \) and \( f = O(g) \), i.e. \( f = g \).

Examples and some guidelines
- \( n^a \) dominates \( n^b \) for \( a > b \): \( n^a \) is a higher complexity class than \( n^b \).
- Any exponential dominates any polynomial (the great divide).
- Which means that any polynomial dominates any logarithm.

Usage warning - Many say big O when they really are thinking big \( \Theta \). However, often they are actually correct ...
"Sorting is ..."
The Limit Rule

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} \begin{cases} 
\in R^+ & \text{then } f(n) \in O(g(n)) \land g(n) \in O(f(n)) \\
= 0 & \text{then } f(n) \in O(g(n)) \land g(n) \not\in O(f(n)) \\
= +\infty & \text{then } f(n) \not\in O(g(n)) \land g(n) \in O(f(n))
\end{cases}
\]

Max Rule:

\[O(f(n) + g(n)) = O(\max(f(n), g(n)))\]

Some Examples

- \(f(n) = 4n\) \(g(n) = 2n\) \(\text{Try both ways}\)
- \(f(n) = n\) \(g(n) = n^2\) \(\text{Try both ways}\)
- \(f(n) = 3n^2 - 6\) \(g(n) = 2n^2 + 4n\) \(\text{Use max rule first}\)
- \(f(n) = \log n^2\) \(g(n) = \log n^3\)
- \(f(n) = \log_2 n\) \(g(n) = \log_3 n\) \(\log_a x = \log_b x / \log_b a\)
- \(f(n) = 2^n\) \(g(n) = 2^{n+1}\)
Complexity Analysis

- Best case, average case, worst case
  - Linear search
  - Binary search
  - Usually consider worst case, unless extremely rare
Another Example:
Proving a Function is in a Particular Efficiency Class

• Show that \( \frac{1}{2}n^2 - 3n = \Theta(n^2) \)

• Must find positive constants \( c_1, c_2, n_0 \) such that

\[
c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0
\]

• Divide through by \( n^2 \) to get

\[
c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2 \quad \forall n \geq n_0
\]

• Consider one inequality at a time
  – for \( n_0 = 7 \) we have \( c_1 \leq 1/14 \)
  – for \( n_0 = 6 \) we have \( c_2 > 1/2 \)

• This proof is \textit{constructive}
Another Example

• Show that $6n^3 \neq \Theta(n^2)$

• Suppose positive constants $c_2, n_0$ exist such that

$$6n^3 \leq c_2 n^2 \quad \forall n \geq n_0$$

• But this implies that

$$n \leq \frac{c_2}{6} \quad \forall n \geq n_0$$

• Can’t be true since $c_2$ is constant

• Proof by contradiction
Procedure \textit{select}(T[1..n])

\begin{align*}
& \text{for } i \leftarrow 1..n - 1 \text{ do} \\
& \quad \text{min}_j \leftarrow i; \ \text{min}_x \leftarrow T[i] \\
& \quad \text{for } j \leftarrow i + 1 \text{ to } n \text{ do} \\
& \quad \quad \text{if } T[j] < \text{min}_x \text{ then } \text{min}_j \leftarrow j \\
& \quad \quad \quad \text{min}_x \leftarrow T[j] \\
& \quad T[\text{min}_j] \leftarrow T[i] \\
& \quad T[i] \leftarrow \text{min}_x
\end{align*}
Analysis Example

Procedure *select*(\(T[1..n]\))

for \(i \leftarrow 1..n - 1\) do

\(\text{minj} \leftarrow i;\ \text{minx} \leftarrow T[i]\)

for \(j \leftarrow i + 1\) to \(n\) do

\(\text{if } T[j] < \text{minx} \text{ then } \text{minj} \leftarrow j\)

\(\text{minx} \leftarrow T[j]\)

\(T[\text{minj}] \leftarrow T[i]\)

\(T[i] \leftarrow \text{minx}\)
Analysis Example

Procedure \textit{select}(T[1..n])

\begin{verbatim}
for i ← 1..n − 1 do
    \textbf{minj} ← i; \textbf{minx} ← T[i]
    for j ← i + 1 to n do
        if T[j] < minx then
            \textbf{minj} ← j
            \textbf{minx} ← T[j]
    \end{verbatim}

T[minj] ← T[i]
T[i] ← minx
Analysis Example

Procedure `select(T[1..n])`
for $i \leftarrow 1..n - 1$ do

- $\text{minj} \leftarrow i$; $\text{minx} \leftarrow T[i]$

for $j \leftarrow i + 1$ to $n$ do

- if $T[j] < \text{minx}$ then $\text{minj} \leftarrow j$
  - $\text{minx} \leftarrow T[j]$
- $T[\text{minj}] \leftarrow T[i]$
- $T[i] \leftarrow \text{minx}$

rest of outer loop is constant time.
Analysis Example

Procedure `select(T[1..n])`

for $i \leftarrow 1..n - 1$ do

$minj \leftarrow i$; $minx \leftarrow T[i]$

for $j \leftarrow i + 1$ to $n$ do

if $T[j] < minx$ then $minj \leftarrow j$

$minx \leftarrow T[j]$

$T[minj] \leftarrow T[i]$

$T[i] \leftarrow minx$

Execute the loop $n$ times for each value $1..n$.

Given this algorithm and model of computation, running time is given by

$$
\sum_{i=1}^{n-1} b + (n - i) c
$$

Is this function a model of worst, best, or average case running time?
Analysis Example

Procedure `select(T[1..n])`

for $i \leftarrow 1..n-1$ do

$\text{min}_j \leftarrow i; \text{min}_x \leftarrow T[i]$

for $j \leftarrow i + 1$ to $n$ do

if $T[j] < \text{min}_x$ then

$\text{min}_j \leftarrow j$

$\text{min}_x \leftarrow T[j]$

$T[\text{min}_j] \leftarrow T[i]$

$T[i] \leftarrow \text{min}_x$

To model **best case** running time, let $c$ represent the time needed to only compare $T[j] < \text{min}_x$ and **not** make the assignments. To model **worst case** running time, let $c$ represent the time needed to compare $T[j] < \text{min}_x$ and make the assignments. What do we do to develop a model for the **average case** complexity?
Analysis Example

What is the order of growth for $t(n)$?

$t(n) = \sum_{i=1}^{n-1} (b + (n - i)c)$

$t(n) = \sum_{i=1}^{n-1} (b + cn - ci)$

$t(n) = \sum_{i=1}^{n-1} (b + cn) - c \sum_{i=1}^{n-1} i$

$t(n) = (n - 1)(b + cn) - c \frac{n(n - 1)}{2}$

$t(n) = \frac{c}{2} n^2 + \left( b - \frac{c}{2} \right) n - b$
Analysis Example

What is the order of growth for \( t(n) \)?

\[
\begin{align*}
t(n) &= \sum_{i=1}^{n-1} \left( b + (n - i) c \right) \\
&= \sum_{i=1}^{n-1} b + c n - c i \\
&= \sum_{i=1}^{n-1} b + c n - c \sum_{i=1}^{n-1} i \\
&= (n - 1) b + c n - c \frac{n(n-1)}{2} \\
&= \frac{c}{2} n^2 + \left( b - \frac{c}{2} \right) n - b
\end{align*}
\]

\( t(n) \) is in \( \Theta(n^2) \)!

Interpret the model--what does this mean? Will this sorting routine always perform worse than a (fictitious) solution with worst case run time of order \( \Theta(n) \)?