CMPE 160: Introduction to Object Oriented Programming

Algorithm Analysis

MATHEMATICAL REVIEW

Mathematical Review
- Exponents
- Logarithms
- Recursive Definitions
- Function Growth
- Proofs

Exponents
- \( X^0 = 1 \) by definition
- \( X^a X^b = X^{(a+b)} \)
- \( X^a / X^b = X^{(a-b)} \)
  - Show that: \( X^{-n} = 1 / X^n \)
- \( (X^a)^b = X^{ab} \)
Logarithms

- \( \log_a X = Y \iff a^Y = X \), \( a > 0, X > 0 \)
  
  * e.g.: \( \log_2 8 = 3; \ 2^3 = 8 \)

- \( \log_a 1 = 0 \) because \( a^0 = 1 \)

  * \( \log X \) means \( \log_2 X \)
  
  * \( \lg X \) means \( \log_{10} X \)
  
  * \( \ln X \) means \( \log_e X \),
  
  where 'e' is the natural number

Logarithms

- \( \log_a (XY) = \log_a X + \log_a Y \)
  
  * e.g.: \( \log_2 8 = 3; \ 2^3 = 8 \)

- \( \log_a (X/Y) = \log_a X - \log_a Y \)
  
  * \( \log_a (X^n) = n \log_a X \)

- \( \log_a b = (\log_2 b)/(\log_2 a) \)

- \( a^{\log_a x} = x \)

Recursive Definitions

- **Basic idea:** To define objects, processes and properties in terms of and properties in terms of simpler objects, simpler processes or properties of simpler objects/processes.

Recursive Definitions

- **Basic idea:** To define objects, processes and properties in terms of simpler objects, simpler processes or properties of simpler objects/processes.

- **Terminating rule - defining the object explicitly.**

- **Recursive rules - defining the object in terms of a simpler object.**
Example

- **Factorial**

  \[ f(n) = n! \]

  \[ f(0) = 1 \text{ i.e. } 0! = 1 \]

  \[ f(n) = n \times f(n-1) \text{ i.e. } n! = n \times (n-1)! \]

Example

- **Fibonacci numbers**

  \[ F(0) = 1 \]

  \[ F(1) = 1 \]

  \[ F(k+1) = F(k) + F(k-1) \]

  \[ 1, 1, 2, 3, 5, 8, \ldots \]

Function Growth

- \[ \lim_{n \to \infty} (n) = \infty \]
- \[ \lim_{n \to \infty} (n^a) = \infty, a > 0 \]
- \[ \lim_{n \to \infty} \left( \frac{1}{n} \right) = 0 \]
- \[ \lim_{n \to \infty} \left( \frac{1}{n^a} \right) = 0, a > 0 \]
- \[ \lim_{n \to \infty} \log(n) = \infty \]
- \[ \lim_{n \to \infty} a^n = \infty, a > 0 \]

- \[ \lim_{n \to \infty} \left( f(x) + g(x) \right) = \lim_{n \to \infty} f(x) + \lim_{n \to \infty} g(x) \]
- \[ \lim_{n \to \infty} \left( f(x) \times g(x) \right) = \lim_{n \to \infty} f(x) \times \lim_{n \to \infty} g(x) \]
- \[ \lim_{n \to \infty} \left( \frac{f(x)}{g(x)} \right) = \lim_{n \to \infty} f(x) / \lim_{n \to \infty} g(x) \]
- \[ \lim_{n \to \infty} \left( \frac{f'(x)}{g'(x)} \right) = \lim_{n \to \infty} \frac{f'(x)}{g'(x)} \]
**Example**

- \( \lim \left( \frac{n}{n^2} \right) = 0, \ n \to \infty \)
- \( \lim \left( \frac{n^2}{n} \right) = \infty, \ n \to \infty \)
- \( \lim \left( \frac{n^2}{n^3} \right) = 0, \ n \to \infty \)
- \( \lim \left( \frac{n^3}{n^2} \right) = \infty, \ n \to \infty \)
- \( \lim \left( \frac{n}{((n+1)/2)} \right) = 2, \ n \to \infty \)

**Some Derivatives**

- \( (\log_a n)' = \frac{1}{n \ln(a)} \)
- \( (a^n)' = (a^n) \ln(a) \)

**Proofs**

- Direct proof
- Proof by induction
- Proof by counterexample
- Proof by contradiction
- Proof by contraposition

**Direct Proof**

- Based on the definition of the object/property

**Example:**

Prove that if a number is divisible by 6 then it is divisible by 2.

Proof: Let \( m \) divisible by 6.

Therefore, there exists \( q \) such that \( m = 6q \)

\( 6 = 2 \cdot 3 \)

\( m = 6q = 2 \cdot 3 \cdot q = 2r, \) where \( r = 3q \)

Therefore \( m \) is divisible by 2.
Proof by Induction

• We use proof by induction when our claim concerns a sequence of cases, which can be numbered.

  • Inductive base:
    - Show that the claim is true for the smallest case, usually $k = 0$ or $k = 1$.

  • Inductive hypothesis:
    - Assume that the claim is true for some $k$.
    - Prove that the claim is true for $k+1$.

Example of Proof by Induction

Prove by induction that $S(N) = \sum_{i=0}^{N} 2^i = 2^{(N+1)} - 1$, for any integer $N \geq 0$.

Inductive base
Let $n = 0$. $S(0) = 2^0 = 1$.
On the other hand, by the formula $S(0) = 2^{(0+1)} - 1 = 1$.
Therefore the formula is true for $n = 0$.

2. Inductive hypothesis
Assume that $S(k) = 2^{(k+1)} - 1$.
We have to show that $S(k+1) = 2^{(k+2)} - 1$.
By the definition of $S(n)$:
$S(k+1) = S(k) + 2^{(k+1)} = 2^{(k+1)} - 1 + 2^{(k+1)} = 2 \cdot 2^{(k+1)} - 1 = 2^{(k+2)} - 1$.

Proof by Counterexample

• Used when we want to prove that a statement is false.
  Types of statements: a claim that refers to all members of a class.

• Example: The statement "all odd numbers are prime" is false.
  A counterexample is the number 9: it is odd and it is not prime.

Proof by Contradiction

• Assume that the statement is false, i.e. its negation is true.

• Show that the assumption implies that some known property is false - this would be the contradiction.

• Example: Prove that there is no largest prime number.
Proof by Contraposition
Used when we have to prove a statement of the form $P \rightarrow Q$.
Instead of proving $P \rightarrow Q$, we prove its equivalent $\sim Q \rightarrow \sim P$
Example: Prove that if the square of an integer is odd then the integer is odd
We can prove using direct proof the statement:
If an integer is even then its square is even.

Good News / Bad News
• Good news: You now know OOP
• Bad news: You need to learn good programming

Big-Oh and Other Notations in Algorithm Analysis

Example
• Question: Given a group of N numbers, how do you find the $k^{th}$ largest?
  - Sol #1: Read into an array, sort, return $k^{th}$
  - Sol #2: Use an array of size $k$ and while reading always keep the largest $k$ values; return $k^{th}$
• Problem: What if $N=1,000,000$ and $k=500,000$?
**Big-Oh and Other Notations in Algorithm Analysis**

- Classifying Functions by Their Asymptotic Growth
- Theta, Little oh, Little omega
- Big Oh, Big Omega
- Rules to manipulate Big-Oh expressions
- Typical Growth Rates

**Classifying Functions by Their Asymptotic Growth**

- Asymptotic growth: The rate of growth of a function

- Given a particular differentiable function $f(n)$, all other differentiable functions fall into three classes:
  - growing with the same rate
  - growing faster
  - growing slower

---

**Theta**

- $f(n)$ and $g(n)$ have the same rate of growth, if

  \[ \lim (f(n)/g(n)) = c, \quad 0 < c < \infty, \quad n \to \infty \]

- Notation: $f(n) = \Theta(g(n))$
  - pronounced "theta"

**Little oh**

- $f(n)$ grows slower than $g(n)$ (or $g(n)$ grows faster than $f(n)$)

  \[ \lim (f(n)/g(n)) = 0, \quad n \to \infty \]

- Notation: $f(n) = o(g(n))$
  - pronounced "little oh"
**Little omega**

- \( f(n) \) grows faster than \( g(n) \)
- (or \( g(n) \) grows slower than \( f(n) \))
- if

\[
\lim( f(n) / g(n) ) = \infty, \quad n \to \infty
\]

Notation: \( f(n) = \omega( g(n) ) \)
- pronounced "little omega"

**Little omega and Little oh**

- if \( g(n) = o( f(n) ) \)
- then \( f(n) = \omega( g(n) ) \)

Examples: Compare \( n \) and \( n^2 \)

\[
\lim( n/n^2 ) = 0, \quad n \to \infty, \quad n = o(n^2)
\]

\[
\lim( n^2/n ) = \infty, \quad n \to \infty, \quad n^2 = \omega(n)
\]

---

**Theta: Relation of Equivalence**

R: "having the same rate of growth":
- relation of equivalence gives a partition over the set of all differentiable functions - classes of equivalence.

Functions in one and the same class are equivalent with respect to their growth.

**Algorithms with Same Complexity**

- Two algorithms have same complexity, if the functions representing the number of operations have same rate of growth.

- Among all functions with same rate of growth we choose the simplest one to represent the complexity.
### Example

**Compare** $n$ and $(n+1)/2$

\[
\lim \left( \frac{n}{(n+1)/2} \right) = 2, \quad \text{same rate of growth}
\]

\[
(n+1)/2 = \Theta(n)
\]

- rate of growth of a linear function

### Example

**Compare** $n^2$ and $n^2 + 6n$

\[
\lim \left( \frac{n^2}{n^2 + 6n} \right) = 1 \quad \text{same rate of growth.}
\]

\[
n^2 + 6n = \Theta(n^2)
\]

rate of growth of a quadratic function

### Example

**Compare** $\log n$ and $\log n^2$

\[
\lim \left( \frac{\log n}{\log n^2} \right) = \frac{1}{2} \quad \text{same rate of growth.}
\]

\[
\log n^2 = \Theta(\log n)
\]

*logarithmic* rate of growth

### Example

**$\Theta(n^3)$:**

\[
\begin{align*}
n^3 \\
5n^3 + 4n \\
105n^3 + 4n^2 + 6n
\end{align*}
\]

**$\Theta(n^2)$:**

\[
\begin{align*}
n^2 \\
5n^2 + 4n + 6 \\
n^2 + 5
\end{align*}
\]

**$\Theta(\log n)$:**

\[
\begin{align*}
\log n \\
\log n^2 \\
\log (n + n^3)
\end{align*}
\]
Comparing Functions

- same rate of growth: \( g(n) = \Theta(f(n)) \)
- different rate of growth:
  
  either \( g(n) = o(f(n)) \)
  
  \( g(n) \) grows slower than \( f(n) \), and hence \( f(n) = \omega(g(n)) \)

  or \( g(n) = \omega(f(n)) \)
  
  \( g(n) \) grows faster than \( f(n) \), and hence \( f(n) = o(g(n)) \)

Example

\( n+5 = \Theta(n) = O(n) = O(n^2) \)

= \( O(n^3) = O(n^5) \)

the closest estimation: \( n+5 = \Theta(n) \)

the general practice is to use
the Big-Oh notation:

\( n+5 = O(n) \)

The Big-Oh Notation

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

if \( f(n) \) grows with
same rate or slower than \( g(n) \).

\( f(n) = \Theta(g(n)) \) or
\( f(n) = o(g(n)) \)

The Big-Omega Notation

The inverse of Big-Oh is \( \Omega \)

If \( g(n) = O(f(n)) \),
then \( f(n) = \Omega(g(n)) \)

\( f(n) \) grows faster or with the same rate as \( g(n) \):

\( f(n) = \Omega(g(n)) \)
Rules to manipulate Big-Oh expressions

Rule 1:

a. If

\[ T_1(N) = O(f(N)) \]

and

\[ T_2(N) = O(g(N)) \]

then

\[ T_1(N) + T_2(N) = \max( O(f(N)), O(g(N)) ) \]

b. If

\[ T_1(N) = O(f(N)) \]

and

\[ T_2(N) = O(g(N)) \]

then

\[ T_1(N) \times T_2(N) = O(f(N) \times g(N)) \]

Rule 2:

If \( T(N) \) is a polynomial of degree \( k \),

then

\[ T(N) = \Theta(N^k) \]

Rule 3:

\( \log^k N = O(N) \) for any constant \( k \).

i.e., logarithms grow very slowly

Examples

\[ n^2 + n = O(n^2) \]

we disregard any lower-order term

\[ n \log(n) = O(n \log(n)) \]

\[ n^2 + n \log(n) = O(n^2) \]
## Typical Growth Rates

<table>
<thead>
<tr>
<th>Function</th>
<th>Growth Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Constant, we write $O(1)$</td>
</tr>
<tr>
<td>logN</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>log(^2)N</td>
<td>Log-squared</td>
</tr>
<tr>
<td>N</td>
<td>Linear</td>
</tr>
<tr>
<td>N logN</td>
<td>N log-squared</td>
</tr>
<tr>
<td>N(^2)</td>
<td>Quadratic</td>
</tr>
<tr>
<td>N(^3)</td>
<td>Cubic</td>
</tr>
<tr>
<td>2(^N)</td>
<td>Exponential</td>
</tr>
<tr>
<td>N!</td>
<td>Factorial</td>
</tr>
</tbody>
</table>

## Problems

<table>
<thead>
<tr>
<th>Expression</th>
<th>Growth Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N^2$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>$2N$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>$N$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>$N^2$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>$2N$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>$N$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>

## Running Time Calculations

**APPLICATION OF BIG-OH TO PROGRAM ANALYSIS**
Background

• The work done by an algorithm, i.e. its complexity, is determined by the number of the basic operations necessary to solve the problem.

The Task

Determine how the number of operations depend on the size of input:

- N - size of input
- F(N) - number of operations

Basic operations in an algorithm

• Problem: Find x in an array

• Operation: Comparison of x with an entry in the array

• Size of input: The number of the elements in the array

Basic operations in an algorithm

Problem: Multiplying two matrices with real entries

Operation: Multiplication of two real numbers

Size of input: The dimensions of the matrices
Basic operations in an algorithm

Problem: Sort an array of numbers

Operation: Comparison of two array entries plus moving elements in the array

Size of input: The number of elements in the array

Counting the number of operations

A. for loops $O(n)$
   The running time of a for loop is at most the running time of the statements inside the loop times the number of iterations.

for loops

```c
sum = 0;
for (i=0; i<n; i++)
    sum = sum + i;
```

The running time is $O(n)$.

B. Nested loops
   The total running time is the running time of the inside statements times the product of the sizes of all the loops.
Nested loops

\[
\text{sum} = 0; \\
\text{for} \ (i=0; \ i<n; \ i++) \\
\quad \text{for} \ (j=0; \ j<n; \ j++) \\
\quad \text{sum}++; \\
\]

The running time is \(O(n^2)\)

Counting the number of operations

C. Consecutive program fragments

Total running time:
- the maximum of the running time of the individual fragments

Consecutive program fragments

\[
\text{sum} = 0; \\
\quad \text{O}(n) \\
\text{for} \ (i=0; \ i < n; \ i++) \\
\quad \text{sum} = \text{sum} + i; \\
\text{sum} = 0; \\
\quad \text{O}(n^2) \\
\text{for} \ (i=0; \ i < n; \ i++) \\
\quad \text{for} \ (j=0; \ j<2n; \ j++) \\
\quad \text{sum}++; \\
\]

The maximum is \(O(n^2)\)

D. If statement

\[
\text{if C} \\
\quad \text{S1;} \\
\text{else} \\
\quad \text{S2;} \\
\]

The running time is the maximum of the running times of \textit{S1} and \textit{S2}.
**Example**

\[ O(n^3) \]

```plaintext
sum = 0;
for (i=0; i<n; i++)
    for (j=0; j<n*n; j++)
        sum++;
```

**Example**

\[ O(n^2) \]

```plaintext
sum = 0;
for (i=0; i<n; i++)
    for (j=0; j<i; j++)
        sum++;
```

**Example**

\[ O(n^3 \times \log n) \]

```plaintext
for (j = 0; j < n*n; j++)
    compute_val(j);
```

The complexity of `compute_val(x)` is given to be \( O(n\log n) \)

**Example**

\[ O(n) \]

```plaintext
for (i=0; i<n; i++)
    if (a[i] == x)
        return 1;
return -1;
```

**Search in an unordered array of elements**
Search in a table nxm

\[ O(n \times m) \]

\[
\text{for } (i=0; i<n; i++)
    \text{ for } (j=0; j<m; j++)
        \text{ if } (a[i][j]==x)
            \text{ return 1 ;}
\text{ return -1;}
\]