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

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

• \( \log_a 1 = 0 \) because \( a^0 = 1 \)
  
  logX means \( \log_2 X \)
  
  lgX means \( \log_{10} X \)
  
  lnX means \( \log_e X \),
  
  where ‘e’ is the natural number

Logarithms

• \( \log_a (XY) = \log_a X + \log_a Y \)
• \( \log_a (X/Y) = \log_a X - \log_a Y \)
• \( \log_a (X^n) = n \log_a X \)
• \( \log_a b = (\log_2 b)/ (\log_2 a) \)
• \( a^{\log_a x} = x \)

Recursive Definitions

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

Recursive Definitions

• Terminating rule - defining the object explicitly.

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

• Factorial
  \[ f(n) = n! \]
  \[ f(0) = 1 \]
i.e. \( 0! = 1 \)
  \[ f(n) = n \times f(n-1) \]
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, ...

Function Growth

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

Function Growth

• \( \lim (f(x) + g(x)) = \lim (f(x)) + \lim (g(x)) \)
• \( \lim (f(x) \times g(x)) = \lim (f(x)) \times \lim (g(x)) \)
• \( \lim \left( \frac{f(x)}{g(x)} \right) = \lim \left( \frac{f(x)}{g(x)} \right) \)
• \( \lim \left( \frac{f(x)}{g(x)} \right) = \lim \left( \frac{f'(x)}{g'(x)} \right) \)
Example

<table>
<thead>
<tr>
<th>Expression</th>
<th>Limit</th>
<th>n → ∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lim (\frac{n}{n^2}) )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( \lim (\frac{n^2}{n}) )</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>( \lim (\frac{n^2}{n^3}) )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( \lim (\frac{n^3}{n^2}) )</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>( \lim (\frac{n}{\frac{n+1}{2}}) )</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Some Derivatives

<table>
<thead>
<tr>
<th>Expression</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{d}{dx} \log_a x )</td>
<td>( \frac{1}{n} \log_a e )</td>
</tr>
<tr>
<td>( \frac{d}{dx} a^x )</td>
<td>( a^x \ln(a) )</td>
</tr>
</tbody>
</table>

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 $N \geq 0$.

1. 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

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = c, \quad 0 < c < \infty
\]

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

**Little oh**

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

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0
\]

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

\[ f(n) \text{ grows faster than } g(n) \]
(or \( g(n) \) grows slower than \( f(n) \))

if

\[ \lim \left( \frac{f(n)}{g(n)} \right) = \infty, \quad n \to \infty \]

Notation: \( f(n) = \omega(g(n)) \)

pronounced "little omega"

---

**Little omega and Little oh**

if \( g(n) = \omega(f(n)) \)

then \( f(n) = \omega(g(n)) \)

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

\[ \lim \left( \frac{n}{n^2} \right) = 0, \quad n \to \infty, \quad n = o(n^2) \]

\[ \lim \left( \frac{n^2}{n} \right) = \infty, \quad n \to \infty, \quad n^2 = \omega(n) \]

---

**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 \((n+1)/2\)

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

\((n+1)/2 = \Theta(n)\)
- rate of growth of a linear function

Example
Compare \( n^2 \) and \( n^2 + 6n \)

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

\(n^2 + 6n = \Theta(n^2)\)
rate of growth of a quadratic function

Example
Compare \( \log n \) and \( \log n^2 \)

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

\(\log n^2 = \Theta(\log n)\)
- logarithmic rate of growth

Example

\[\Theta(n^3):\]
\[
n^3
\]
\[5n^3 + 4n
\]
\[105n^3 + 4n^2 + 6n
\]

\[\Theta(n^2):\]
\[
n^2
\]
\[5n^2 + 4n + 6
\]
\[n^2 + 5
\]

\[\Theta(\log n):\]
\[
\log n
\]
\[\log n^2
\]
\[\log (n + n^3)
\]
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 \( f(n) = o(g(n)) \)

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 \( g(n) \):
\( f(n) = \Omega(g(n)) \)

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:
\( n+5 = O(n) \)
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
   \[ T_1(N) * T_2(N) = O(f(N) * g(N)) \]

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

- C: constant, we write $O(1)$
- $\log N$: logarithmic
- $\log^2 N$: log-squared
- N: linear
- NlogN: log-squared
- $N^2$: quadratic
- $N^3$: cubic
- $2^N$: exponential
- N!: factorial

Problems

- $N^2 = O(N^2)$: true
- $2N = O(N^2)$: true
- $N = O(N^2)$: true
- $N^2 = O(N)$: false
- $2N = O(N)$: true
- $N = O(N)$: true

Problems

- $N^2 = \Theta (N^2)$: true
- $2N = \Theta (N^2)$: false
- $N = \Theta (N^2)$: false
- $N^2 = \Theta (N)$: false
- $2N = \Theta (N)$: true
- $N = \Theta (N)$: true

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

for loops

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

The running time is $O(n)$.

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.

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

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

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

Consecutive program fragments

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

$O(n)$

```c
sum = 0;
for (i=0; i < n; i++)
    for (j=0; j<2n; j++)
        sum++;
```

$O(n^2)$

The maximum 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

D. If statement

```c
if C
    S1;
else
    S2;
```

The running time is the maximum of the running times of $S1$ and $S2$. 
Example

\[ O(n^3) \]

\[
\begin{align*}
\text{sum} & = 0; \\
\text{for} \ (i=0; \ i<n; \ i++) & \\
\quad \text{for} \ (j=0; \ j<n*n; \ j++) & \\
\quad & \quad \text{sum}++; \\
\end{align*}
\]

Example

\[ O(n^2) \]

\[
\begin{align*}
\text{sum} & = 0; \\
\text{for} \ (i=0; \ i<n; \ i++) & \\
\quad & \quad \text{for} \ (j=0; \ j<i; \ j++) \\
\quad & \quad \quad \text{sum}++; \\
\end{align*}
\]

Example

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

\[
\begin{align*}
\text{for} \ (j = 0; \ j < n*n; \ j++) & \\
\quad & \quad \text{compute_val}(j); \\
\end{align*}
\]

The complexity of \text{compute_val}(x) is given to be \[ O(n \cdot \log n) \]

Example

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

Search in an unordered array of elements

\[ O(n) \]

\[
\begin{align*}
\text{for} \ (i=0; \ i<n; \ i++) & \\
\quad & \quad \text{if} \ (a[i]==x) \\
\quad & \quad \quad \text{return} \ 1; \\
\quad & \quad \text{return} \ -1; \\
\end{align*}
\]
Search in a table nxm

\[ O(n*m) \]

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

Logarithms in Running Time

- Binary search
- Euclid’s algorithm
- Exponentials
- Rules to count operations

Divide-and-Conquer Algorithms

- Subsequently reducing the problem by a factor of two requires \( O(\log N) \) operations
Why log N?

- A complete binary tree with $N$ leaves has $\log N$ levels.
- Each level in the divide-and-conquer algorithm corresponds to an operation.
- Hence, the number of operations is $O(\log N)$.

Example: 8 leaves, 3 levels

Binary Search

Solution 1:
Scan all elements from left to right, each time comparing with $X$.
- Requires $O(N)$ operations.

Solution 2: $O(\log N)$
Find the middle element $A_{\text{mid}}$ in the list and compare it with $X$:
- If they are equal, stop.
- If $X < A_{\text{mid}}$ consider the left part.
- If $X > A_{\text{mid}}$ consider the right part.

Do until the list is reduced to one element.
Euclid's Algorithm
Finding the greatest common divisor (GCD)

GCD of M and N (s.t. M > N)
= GCD of N and M\%N

GCD and Recursion
Recursion:
If M\%N = 0 return N
Else return GCD(N, M\%N)
The answer is the last nonzero remainder.

Euclid's Algorithm
(non-recursive implementation)

long gcd(long m, long n) {
    long rem;
    while (n!=0) {
        rem = m\%n;
        m = n;
        n = rem;
    }
    return m;
}
Why $O(\log N)$?

- $M \% N \leq M/2$
- After 1$^{st}$ iteration, $N$ appears as first argument, the remainder is less than $N/2$
- After 2$^{nd}$ iteration, the remainder appears as first argument and will be reduced by a factor of two
- Hence, $O(\log N)$

Computing $X^N$

$X^N = X^*(X^2)^{N/2}$, $N$ is odd

$X^N = (X^2)^{N/2}$, $N$ is even

Why $O(\log N)$?

If $N$ is odd: two multiplications

The operations are at most $2\log N$: $O(\log N)$

```c
long pow (long x, int n) {
    if (n == 0)
        return 1;
    if (is_Even(n))
        return pow(x*x, n/2);
    else
        return x * pow(x*x, n/2);
}
```
Another recursion for $X^N$

- Another recursive definition that reduces the power just by 1:
  
  $$X^N = X \times X^{N-1}$$

- Here the operations are $N-1$, i.e. $O(N)$, and the algorithm is less efficient than the divide-and-conquer algorithm.

How to count operations

- Single statements (not function calls): constant $O(1) = 1$

- Sequential fragments: the maximum of the operations of each fragment

How to count operations

- Single loop running up to $N$, with single statements in its body: $O(N)$

- Single loop running up to $N$, with the number of operations in the body $O(f(N))$: $O(N \times f(N))$

How to count operations

- Two nested loops each running up to $N$, with single statements: $O(N^2)$

- Divide-and-conquer algorithms with input size $N$: $O(\log N)$

  Or $O(N \times \log N)$ if each step requires additional processing of $N$ elements
Example: What is the probability for two numbers to be relatively prime?

tot=0; rel=0;
for (i=0; i<=n; i++)
    for (j=i+1; j<=n; j++) {
        tot++;
        if (gcd(i,j)==1)
            rel++;
    }
return (rel/tot);

Running time = ?