Introduction to Information Security

Christian J. Rudder

November 2024

Contents

This page is left intentionally blank.

The following five sections are from Concise Work Section Modules [37].
Available at: `https://github.com/Concise-Works/sect-modules`

— 1 —

Prerquesites

1.1 Asymptotic Notation

Asymptotic analysis is a method for describing the limiting behavior of functions as inputs grow infinitely.

Definition 1.1: Asymptotic

Let $f(n)$ and $g(n)$ be two functions. As n grows, if $f(n)$ grows closer to $g(n)$ never reaching, we say that "$f(n)$ is **asymptotic** to $g(n)$."

We call the point where $f(n)$ starts behaving similarly to $g(n)$ the **threshold** n_0. After this point n_0, $f(n)$ follows the same general path as $g(n)$.

Definition 1.2: Big-O: (Upper Bound)

Let f and g be functions. $f(n)$ our function of interest, and $g(n)$ our function of comparison.

Then we say $f(n) = O(g(n))$, "$f(n)$ **is big-O of** $g(n)$," if $f(n)$ grows no faster than $g(n)$, up to a constant factor. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq f(n) \leq c \cdot g(n)$$

Represented as the ratio $\dfrac{f(n)}{g(n)} \leq c$ for all $n \geq n_0$. Analytically we write,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

Meaning, as we chase infinity, our numerator grows slower than the denominator, bounded, never reaching infinity.

Examples:

(i.) $3n^2 + 2n + 1 = O(n^2)$

(ii.) $n^{100} = O(2^n)$

(iii.) $\log n = O(\sqrt{n})$

Proof 1.1: $\log n = O(\sqrt{n})$

We setup our ratio:

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}}$$

Since $\log n$ and $\sqrt{n}$ grow infinitely without bound, they are of indeterminate form $\frac{\infty}{\infty}$. We apply L'Hopital's Rule, which states that taking derivatives of the numerator and denominator will yield an evaluateable limit:

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{d}{dn}\log n}{\frac{d}{dn}\sqrt{n}}$$

Yielding derivatives, $\log n = \frac{1}{n}$ and $\sqrt{n} = \frac{1}{2\sqrt{n}}$. We substitute these back into our limit:

$$\lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Our limit approaches 0, as we have a constant factor in the numerator, and a growing denominator. Thus, $\log n = O(\sqrt{n})$, as $0 < \infty$. ∎

Definition 1.3: Big-Ω: (Lower Bound)

The symbol Ω reads "Omega." Let f and g be functions. Then $f(n) = \Omega(g(n))$ if $f(n)$ grows no slower than $g(n)$, up to a constant factor. I.e., lower bounded by $g(n)$. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq c \cdot g(n) \leq f(n)$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

Meaning, as we chase infinity, our numerator grows faster than the denominator, approaching 0 asymptotically.

Examples: $n! = \Omega(2^n)$; $\dfrac{n}{100} = \Omega(n)$; $n^{3/2} = \Omega(\sqrt{n})$; $\sqrt{n} = \Omega(\log n)$

Definition 1.4: Big Θ: (Tight Bound)

The symbol Θ reads "Theta." Let f and g be functions. Then $f(n) = \Theta(g(n))$ if $f(n)$ grows at the same rate as $g(n)$, up to a constant factor. I.e., $f(n)$ is both upper and lower bounded by $g(n)$. Let n_0 be our asymptotic threshold, and $c_1 > 0, c_2 > 0$ be some constants. Then, for all $n \geq n_0$,

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

Meaning, as we chase infinity, our numerator grows at the same rate as the denominator.

Examples: $n^2 = \Theta(n^2)$; $2n^3 + 2n = \Theta(n^3)$; $\log n + \sqrt{n} = \Theta(\sqrt{n})$.

Definition 1.5: Little o: (Strict Upper Bound)

The symbol o reads "little-o." Let f and g be functions. Then $f(n) = o(g(n))$ if $f(n)$ grows strictly slower than $g(n)$, meaning $f(n)$ becomes insignificant compared to $g(n)$ as n grows large. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq f(n) < c \cdot g(n)$$

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

Meaning, as we chase infinity, the ratio of $f(n)$ to $g(n)$ shrinks to zero.

Examples: $n = o(n^2)$; $\log n = o(n)$; $n^{0.5} = o(n)$.

Definition 1.6: Little ω: (Strict Lower Bound)

The symbol ω reads "little-omega." Let f and g be functions. Then $f(n) = \omega(g(n))$ if $f(n)$ grows strictly faster than $g(n)$, meaning $g(n)$ becomes insignificant compared to $f(n)$ as n grows large. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq c \cdot g(n) < f(n)$$

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

Meaning, as we chase infinity, the ratio of $g(n)$ to $f(n)$ shrinks to zero.

Examples: $n^2 = \omega(n)$; $n = \omega(\log n)$.

Definition 1.7: Asymptotic Equality ($\sim$)

The symbol $\sim$ reads "asymptotic equality." Let f and g be functions. Then $f(n) \sim g(n)$ if, as $n \to \infty$, the ratio of $f(n)$ to $g(n)$ approaches 1. I.e., the two functions grow at the same rate asymptotically. Formally,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

Meaning, as n grows large, the two functions become approximately equal.

Examples: $n + 100 \sim n, \quad \log(n^2) \sim 2\log n$.

Tip: To review:

- **Big-O:** $f(n) < g(n)$ (Upper Bound); $f(n)$ grows no faster than $g(n)$.
- **Big-Ω:** $f(n) > g(n)$ (Lower Bound); $f(n)$ grows no slower than $g(n)$.
- **Big-Θ:** $f(n) = g(n)$ (Tight Bound); $f(n)$ grows at the same rate as $g(n)$.
- **Little-o:** $f(n) < g(n)$ (Strict Upper Bound); $f(n)$ grows strictly slower than $g(n)$.
- **Little-ω:** $f(n) > g(n)$ (Strict Lower Bound); $f(n)$ grows strictly faster than $g(n)$.
- **Asymptotic Equality:** $f(n) \sim g(n)$; $f(n)$ grows at the same rate as $g(n)$.

Theorem 1.1: Types of Asymptotic Behavior

The following are common relationships between different types of functions and their asymptotic growth rates:

- **Polynomials.** Let $f(n) = a_0 + a_1 n + \cdots + a_d n^d$ with $a_d > 0$. Then, $\underline{f(n) \text{ is } \Theta(n^d).}$
  E.e., $3n^2 + 2n + 1$ is $\Theta(n^2)$.
- **Logarithms.** $\underline{\Theta(\log_a n) \text{ is } \Theta(\log_b n)}$ for any constants $a, b > 0$. That is, logarithmic functions in different bases have the same growth rate.
  E.g., $\log_2 n$ is $\Theta(\log_3 n)$.
- **Logarithms and Polynomials.** For every $d > 0$, $\underline{\log n \text{ is } O(n^d).}$ This indicates that logarithms grow slower than any polynomial.
  E.g., $\log n$ is $O(n^2)$.
- **Exponentials and Polynomials.** For every $r > 1$ and every $d > 0$, $\underline{n^d \text{ is } O(r^n).}$ This means that exponentials grow faster than any polynomial.
  E.e., n^2 is $O(2^n)$.

1.2 Evaluating Algorithms

When analyzing algorithms, we are interested in two primary factors: time and space complexity.

Definition 2.1: Time Complexity

The **time complexity** of an algorithm is the amount of time it takes to run as a function of the input size. We use asymptotic notation to describe the time complexity of an algorithm.

Definition 2.2: Space Complexity

The **space complexity** of an algorithm is the amount of memory it uses to store inputs and subsequent variables during the algorithm's execution. We use asymptotic notation to describe the space complexity of an algorithm.

Below is an example of a function and its time and space complexity analysis.

Function 2.1: Arithmetic Series - `Fun1`(A)

Computes a result based on a length-n array of integers:

Input: A length-n array of integers.
Output: An integer p computed from the array elements.

```
Function Fun1(A):
    $p \leftarrow 0$;
    for $i \leftarrow 1$ to $n - 1$ do
        for $j \leftarrow i + 1$ to $n$ do
            $p \leftarrow p + A[i] \cdot A[j]$;
```

Time Complexity: For $f(n) :=$ `Fun1`(A), $f(n) = \frac{n^2}{2} = O(n^2)$. This is because the function has a nested loop structure, where the inner for-loop runs $n - i$ times, and the outer for-loop runs $n - 1$ times. Thus, the total number of iterations is $\sum_{i=1}^{n-1} n - i = \frac{n^2}{2}$.

Space Complexity: We yield $O(n)$ for storing an array of length n. The variable p is $O(1)$ (constant), as it is a single integer. Hence, $f(n) = n + 1 = O(n)$.

Additional Example: Let $f(n, m) = n^2m + m^3 + nm^3$. Then, $f(n, m) = O(n^2m + m^3)$. This is because both n and m must be accounted for. Our largest n term is n^2m, and our largest m term is m^3 both dominate the expression. Thus, $f(n, m) = O(n^2m + m^3)$.

1.3 Computers & Number Base Systems

Definition 3.1: Turing Machine

A **Turing Machine** is a theoretical computational model used to describe the capabilities of a general-purpose **computer**. It consists of an infinite tape (memory) and a read/write head processing symbols on the tape, one at a time, according to a set of predefined rules. The machine moves left or right, reading or writing symbols, and changing states based on what it reads.

The machine **halts** once it reaches a final state or continues indefinitely. Serving as a flexible, **higher-order function** (a function which receives functions).

Definition 3.2: Von Neumann Architecture

Modern computers operate on a model known as the **Von Neumann architecture**, which consists of three primary components:

1. **Memory**: Stores data and instructions as sequences of bits.
2. **Arithmetic and Logic Unit (ALU)**: Executes operations such as addition, subtraction, multiplication, and division on numbers stored in memory.
3. **Control Unit**: Directs the execution of instructions and manages the flow of data between memory and the ALU.

Where numbers are stored in memory cells, each cell holding an integer value represented in a fixed base, typically $B = 2$, meaning **binary**. Where each digit is less than the base B. We represent integers in memory as:

$$a = \sum_{i=0}^{k-1} a_i B^i$$

where a_i represents the individual digits, and B is the base. For large integers, computations may require manipulating several memory cells to store the full number.

Example: Consider the integer $a = 13$, and let us represent it in base $B = 2$ (binary). We can express this number as a sum of powers of 2, corresponding to the binary representation of 13:

$$a = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13$$

In binary, this is represented as the sequence of digits: $a = (1101)_2$. Here, the coefficients $a_3 = 1$, $a_2 = 1$, $a_1 = 0$, and $a_0 = 1$ correspond to the binary digits of 13.
Similarly, if we want to represent $a = 45$ in base $B = 10$ **(decimal)**, we write:

$$a = 4 \cdot 10^1 + 5 \cdot 10^0 = 40 + 5 = 45$$

In this case, the coefficients $a_1 = 4$ and $a_0 = 5$ correspond to the decimal digits of 45.

Definition 3.3: Hexadecimal

Hexadecimal base $B = 16$, using digits 0-9 and the letters A-F, where $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, and $F = 15$. Hexadecimal is commonly used in computing due to its compact representation of binary data. For example, a **byte** (8 bits) can be represented as two hexadecimal digits, simplifying the display of binary data.

Theorem 3.1: Base $2 \leftrightarrow 16$ Conversion

Let bases $B := 2$ (binary) and $H := 16$ (hexadecimal). At a high-level:

Binary to Hexadecimal:

1. Group B digits in sets of 4, right to left. **Pad** leftmost group with 0's if necessary for a full group.
2. Compute each group, replacing the result with their H digit.
3. Finally, combine each H group.

Hexadecimal to Binary:

1. Convert each H digit into a 4 bit B group.
2. Finally, combine all B groups.

Additionally we may also trim any leading 0's.

Example:

- Binary to Hexadecimal:

$$101101111010_2 \quad \Rightarrow \quad \text{Group as ([1011] [0111] [1010])} \quad \Rightarrow \quad B7A_{16}$$

- Hexadecimal to Binary:

$$3F5_{16} \quad \Rightarrow \quad [0011]\ [1111]\ [0101]_2 \Rightarrow \quad 1111110101_2$$

The following definition is for completeness: applications of such a base are currently uncommon.

Definition 3.4: Unary

Unary, base $B = 1$. A system where each number is represented by a sequence of B symbols. Where the number n is represented by n symbols. Often used in theoretical computer science to prove the existence of computable functions.

Example: The number 5 in unary is represented by 5 symbols: $5 = \texttt{IIIII}$ or $2 = \texttt{II}$. There is no concept of 0, the absence of symbols represents 0.

Definition 3.5: Most & Least Significant Bit

In a binary number, the **most significant bit (MSB)** is the leftmost bit. The **least significant bit (LSB)** is the rightmost bit.

Example: Consider this byte (8 bits), $[1111\ 1110]_2$, the MSB = 1 and the LSB = 0.

Theorem 3.2: Adding Binary

We may use the add and carry method alike decimal addition: **Binary Addition Rules:**

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ (add 1 to the next digit (left))

We call the last step, **carry**, as we carry our overflow to the next digit.

Example: Adding $0010\ 0011\ 0100_2$ and 0100_2:

$$\begin{array}{r} 1\,1000\,\overset{1}{0}{\color{red}1}00 \\ +\,0\,0001\,0{\color{red}1}00 \\ \hline 1\,1001\,1000 \end{array}$$

Where $[1\ 1000\ 0100]_2 + [0\ 0001\ 0100]_2 = [1\ 1001\ 1000]_2$.

Theorem 3.3: Signed Binary Numbers - Two's Complement

In a **two's complement system**, an n-bit signed (positive or negative) binary number can represent values in the range $[-2^{n-1}, 2^{n-1}-1]$. Then by most significant bit (MSB):

- If MSB is 0, the number is positive;
- If MSB is 1, the number is negative.

Conversion to Two's Complement :

1. Take an unsigned binary number and invert all bits, turning 0's to 1's and 1's to 0's.
2. Finally add 1 to the least significant bit.

Example: Converting -5 into a 4-bit two's complement:

$$\begin{aligned} 5 \to 0101 &\quad \text{(binary for 5)} \\ 1010 &\quad \text{(inverted)} \\ 1011 &\quad \text{(add 1)} \end{aligned}$$

Thus, -5 is represented as 1011 in 4-bits under two's complement.

1.4 Computing Large Numbers

In this section we discuss algorithms for computing large numbers, but first we define algorithmically, addition, subtraction, multiplication, division, and modula.

Definition 4.1: Wordsize

Our machine has a fixed **wordsize**, which is how much each memory cell can hold. Systems like 32-bit or 64-bit can hold 2^{32} ($\approx$ 4.3 billion) or 2^{64} ($\approx$ 18.4 quintillion) bits respectively.

We say the ALU can performs arithmetic operations at $O(1)$ time, within wordsize. Operations beyond this size we deem **large numbers**.

The game we play in the following algorithms is to compute large integers without exceeding wordsize. Moving forward, we assume our machine is a typical 64-bit system.

Function 4.1: Length of digits - $\|a\|$

We will use the notation $\|a\|$ to denote the number of digits in the integer a. For example, $\|123\| = 3$ and $\|0\| = 1$.

Definition 4.2: Computer Integer Division

Our ALU only returns the quotient after division. We denote the quotient as $\lfloor a/b \rfloor : a, b \in \mathbb{Z}$.

Our first hurdle is long division as , which will set up long addition and subtraction for success.
Scenario - Grade School Long Division: Goes as follows, take $\frac{a}{b}$. Find how times b fits into a evenly, q times. Then $a - bq$ is our remainder r.
Examples: let $a = \{12, 5, 17, 40, 89\}$, $b = \{4, 2, 3, 9, 10\}$ respectively, and base $B = 10$,

$$\text{(1.)}\quad \begin{array}{r} 3 \\ 4\overline{)12} \\ 12 \\ \hline 0 \end{array} \qquad \text{(2.)}\quad \begin{array}{r} 2 \\ 2\overline{)5} \\ 4 \\ \hline 1 \end{array} \qquad \text{(3.)}\quad \begin{array}{r} 5 \\ 3\overline{)17} \\ 15 \\ \hline 2 \end{array} \qquad \text{(4.)}\quad \begin{array}{r} 4 \\ 9\overline{)40} \\ 36 \\ \hline 4 \end{array} \qquad \text{(5.)}\quad \begin{array}{r} 8 \\ 10\overline{)89} \\ 80 \\ \hline 9 \end{array}$$

Take (3.), $a = 17$, $b = 3$: 3 fits into 17 five times, which is 15. 17 take away 15 is 2, our remainder. We create an algorithm to compute this process.

Key Observation: Consider the following powers of 2 of form $x = 2^n + s$, where $x, n, s \in \mathbb{Z}$:

$$3 = 2 + 1 = 0000\ 0011_2 \tag{1}$$
$$6 = 4 + 2 = 0000\ 0110_2 \tag{2}$$
$$12 = 8 + 4 = 0000\ 1100_2 \tag{3}$$
$$24 = 16 + 8 = 0001\ 1000_2 \tag{4}$$
$$48 = 32 + 16 = 0011\ 0000_2 \tag{5}$$
$$96 = 64 + 32 = 0110\ 0000_2 \tag{6}$$
$$192 = 128 + 64 = 1100\ 0000_2 \tag{7}$$

Notice that as we increase the power of 2, the number of bits shift left towards a higher-order bit. Now, instead of calculating powers of 2, we shift bits left or right, to yield instantaneous results.

Theorem 4.1: Binary Bit Shifting (Powers of 2)

Let x be a binary unsigned integer. Where "$\ll$" and "$\gg$" are left and right bit shifts:
Left Shift by k bits: $x \ll k := x \cdot 2^k$
Right Shift by k bits: $x \gg k = \lfloor x/2^k \rfloor$
Remainder: bits pushed out after right shift(s).

Example: Observe, $16 = 10000$ (4 zeros), $8 = 1000$ (3 zeros), we shift by 4 and 3 respectively:

- Instead of $3 \cdot 16$ in base 10, we can $3 \ll 4 = 48$, as $3 \cdot 2^4 = 48$.
- Conversely, Instead of $48/16$ in base 10, $48 \gg 4 = 3$, as $\lfloor 48/2^4 \rfloor = 3$.
- Catching the remainder: say we have $37/8$ base 10, then,

$$37 = 100101_2 \quad \text{and} \quad 8 = 1000_2 \text{ then } 37 \gg 3 = 4 \text{ remainder } 5,$$

as $[100101] \gg 3 = [000100]101$, where 101_2 is our remainder 5_{10}.

Function 4.2: Division with Remainder in Binary (Outline) - *QuoRem()*

For binary integers, let dividend $a = (a_{k-1} \cdots a_0)_2$ and divisor $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned, with $k \geq 1$, $\ell \geq 1$, ensuring $0 \leq b \leq a$, and $b_{\ell-1} \neq 0$, ensuring $b > 0$.

We compute q and r such that, $a = bq + r$ and $0 \leq r < b$. Assume $k \geq \ell$; otherwise, $a < b$. We set $q \leftarrow 0$ and $r \leftarrow a$. Then quotient $q = (q_{m-1} \cdots q_0)_2$ where $m := k - \ell + 1$.

Input: a, b (binary integers)
Output: q, r (quotient and remainder in binary)

```
1 Function QuoRem($a, b$):
2     $r \leftarrow a$;
3     $q \leftarrow \{0_{m-1} \cdots 0\}$;
4     for $i \leftarrow \|a\| - \|b\| - 1$ down to $0$ do
5         $q_i \leftarrow \left\lfloor \frac{r}{b \ll i} \right\rfloor$;
6         $r \leftarrow r - (q_i \cdot (b \ll i))$;
```

Time Complexity: $O(\|a\|(\|a\| - \|b\|))$. In short, line 5 we preform division on $\|a\|$ bits of decreasing size. Though **not totally necessary**, For more detail visit https://shoup.net/ntb/ntb-v2.pdf on page 60. General n cases can be found in Theorem (4.2).

Example Let $a = 47_{10} = 101111_2$ and $b = 5_{10} = 101_2$, we run *QuoRem(a, b)*. We summarize the above example as, "How many times does 101_2 fit into 101111_2?"

1. Does $5 \ll 3$ fit into 101000_2? It fits! $q = 1000_2$, $r = 101111_2 - 101000_2$.
2. Does $5 \ll 2$ fit into 0111_2? no fits! $q = 1000_2$, $r = 0111_2$.
3. Does $5 \ll 1$ fit into 0111_2? no fits! $q = 1000_2$, $r = 0111_2$.
4. Does $5 \ll 0$ fit into 0111_2? It fits! $q = 1001_2$, $r = 0111_2 - 0101$.
5. Return $q = 1001_2 = 9_{10}$, $r = 0010 = 2_{10}$

Long addition: We craft an algorithm for grade school long addition, which goes as follows:

$$\begin{array}{r} \overset{1}{2}5\,3\overset{1}{0}8 \\ +\,39\,406 \\ \hline 64\,714 \end{array}$$

Where adding, $25,308 + 39,406 = 64,714$. We create an algorithm to compute this in the following function.

Here the function *QuoRem* (4.2) to return (quotient, remainder) preforms in $O(1)$ as bits are small enough for word size.

Function 4.3: Addition of Binary Integers - *Add()*

Let $a = (a_{k-1} \cdots a_0)_2$ and $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned binary integers, where $k \geq \ell \geq 1$. We compute $c := a + b$ where the result $c = (c_k c_{k-1} \cdots c_0)_2$ is of length $k+1$, assuming $k \geq \ell$. If $k < \ell$, swap a and b. This algorithm computes the binary representation of $a + b$.

Input: a, b (binary integers)
Output: $c = (c_k \cdots c_0)_2$ (sum of $a + b$)

```
Function Add(a, b):
    $carry \leftarrow 0$;
    for $i \leftarrow 0$ to $\ell - 1$ do
        $tmp \leftarrow a_i + b_i + carry$;
        $(carry, c_i) \leftarrow$ QuoRem($tmp, 2$);
    for $i \leftarrow \ell$ to $k - 1$ do
        $tmp \leftarrow a_i + carry$;
        $(carry, c_i) \leftarrow$ QuoRem($tmp, 2$);
    $c_k \leftarrow carry$;
    return $c = (c_k \cdots c_0)_2$;
```

Note: $0 \leq carry \leq 1$ and $0 \leq tmp \leq 3$.
Time Complexity: $O(\max(\|a\|, \|b\|))$, as we iterate at most the length of the largest input.
Space Complexity: $O(\|a\| + \|b\|)$, though $c = k + 1$, constants are negligible as $k, \ell \to \infty$.

For subtracting, $5,308 - 3,406 = 1,904$, where we borrow 10 from the 5 to make 13:

$$\begin{array}{r} \overset{4}{\cancel{5}}\,\overset{10}{3}08 \\ -\,3\ 406 \\ \hline 1\ 904 \end{array}$$

Function 4.4: Subtraction of Binary Integers - *Subtract()*

Let $a = (a_{k-1} \cdots a_0)_2$ and $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned binary integers, where $k \geq \ell \geq 1$ and $a \geq b$. We compute $c := a - b$ where the result $c = (c_{k-1} \cdots c_0)_2$ is of length k, assuming $a \geq b$. If $a < b$, swap a and b and set a negative flag to indicate the result is negative. This algorithm computes the binary representation of $a - b$.

Input: a, b (binary integers)
Output: $c = (c_{k-1} \cdots c_0)_2$ (difference of $a - b$)

```
Function Subtract(a, b):
    $borrow \leftarrow 0$;
    for $i \leftarrow 0$ to $\ell - 1$ do
        $tmp \leftarrow a_i - b_i - borrow$;
        if $tmp < 0$ then
            $borrow \leftarrow 1$;
            $c_i \leftarrow tmp + 2$;
        else
            $borrow \leftarrow 0$;
            $c_i \leftarrow tmp$;
    for $i \leftarrow \ell$ to $k - 1$ do
        $tmp \leftarrow a_i - borrow$;
        if $tmp < 0$ then
            $borrow \leftarrow 1$;
            $c_i \leftarrow tmp + 2$;
        else
            $borrow \leftarrow 0$;
            $c_i \leftarrow tmp$;
    return $c = (c_{k-1} \cdots c_0)_2$;
```

Note: $0 \leq borrow \leq 1$. Subtraction may produce a borrow when $a_i < b_i$.
Time Complexity: $O(\max(\|a\|, \|b\|))$, iterating at most the length of the largest input.
Space Complexity: $O(\|a\| + \|b\|)$, as the length of c is at most k, with constants negligible as $k, \ell \to \infty$.

For multiplication, $24 \cdot 16 = 384$:

$$\begin{array}{r} \overset{2}{2}4 \\ \times\ 16 \\ \hline 144 \\ +240 \\ \hline 384 \end{array}$$

Where $6 \cdot 4 = 24$, we write the 4 and carry the 2. Then $6 \cdot 2 = 12$ plus the carried 2 is 14. Then we multiply the next digit, 1, we add a 0 below our 144, and repeat the process. Every new 10s place we add a 0. Then we add our two products to get 384.

We create an algorithm to compute this process in the following function:

Function 4.5: Multiplication of Base-B Integers - *Mul()*

Let $a = (a_{k-1} \cdots a_0)_B$ and $b = (b_{\ell-1} \cdots b_0)_B$ be unsigned integers, where $k \geq 1$ and $\ell \geq 1$. The product $c := a \cdot b$ is of the form $(c_{k+\ell-1} \cdots c_0)_B$, and may be computed in time $O(k\ell)$ as follows:

Input: a, b (base-B integers)
Output: $c = (c_{k+\ell-1} \cdots c_0)_B$ (product of $a \cdot b$)

```
Function Mul(a, b):
    for $i \leftarrow 0$ to $k + \ell - 1$ do
        $c_i \leftarrow 0$;
    for $i \leftarrow 0$ to $k - 1$ do
        $carry \leftarrow 0$;
        for $j \leftarrow 0$ to $\ell - 1$ do
            $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$;
            $(carry, c_{i+j}) \leftarrow$ QuoRem($tmp, B$);
        $c_{i+\ell} \leftarrow carry$;
    return $c = (c_{k+\ell-1} \cdots c_0)_B$;
```

Note: At every step, the value of *carry* lies between 0 and $B - 1$, and the value of *tmp* lies between 0 and $B^2 - 1$.
Time Complexity: $O(\|a\| \cdot \|b\|)$, since the algorithm involves $k \cdot \ell$ multiplications.
Space Complexity: $O(\|a\| + \|b\|)$, since we store the digits of a, b, and c.

Function 4.6: Decimal to Binary Conversion - *DecToBin()*

This function converts a decimal number n into its binary equivalent by repeatedly dividing the decimal number by 2 and recording the remainders.

Input: n (a decimal number)
Output: b (binary representation of n)

```
Function DecToBin(n):
    $b \leftarrow$ empty string;
    while $n > 0$ do
        $r \leftarrow n \bmod 2$;
        $n \leftarrow \lfloor \frac{n}{2} \rfloor$;
        $b \leftarrow r + b$;
    return $b$;
```

Time Complexity: $O(\log n)$, as the number of iterations is proportional to the number of bits in n.
Space Complexity: $O(n)$, storing our input n.

Example: Converting 89 to binary given the above function:

$$\begin{aligned}
89_{10} \div 2 &= 44 \quad \text{rem } 1, \longleftarrow \ \textbf{LSB} \\
44_{10} \div 2 &= 22 \quad \text{rem } 0, \\
22_{10} \div 2 &= 11 \quad \text{rem } 0, \\
11_{10} \div 2 &= 5 \quad \text{rem } 1, \\
5_{10} \div 2 &= 2 \quad \text{rem } 1, \\
2_{10} \div 2 &= 1 \quad \text{rem } 0, \\
1_{10} \div 2 &= 0 \quad \text{rem } 1. \longleftarrow \ \textbf{MSB}
\end{aligned}$$

Thus, $89_{10} = 1011001_2$.

Theorem 4.2: Time Complexity of Basic Arithmetic Operations

We generalize the time complexity to a and b as n-bit integers.

(i) **Addition & Subtraction:** $a \pm b$ in time $O(n)$.

(ii) **Multiplication:** $a \cdot b$ in time $O(n^2)$.

(iii) **Quotient Remainder** quotient $q := \lfloor \frac{a}{b} \rfloor : b \neq 0, a > b$; and remainder $r := a \mod b$ has time $O(n^2)$.

1.5 Computational Efficiency

Theorem 5.1: Binary Length of a Number - $\|a\|$

The binary length of an integer a_{10} in binary representation, is given by:

$$\|a\| := \begin{cases} \lfloor \log_2 |a| \rfloor + 1 & \text{if } a \neq 0, \\ 1 & \text{if } a = 0, \end{cases}$$

as $\lfloor \log_2 |a| \rfloor + 1$ correlates to the highest power of 2 required to represent a.

Example: Think about base 10 first. Let there be a 9 digit number $d = 684,301,739$. To reach 9 digits takes 10^8; The exponent plus 1 yields $\|d\|$. Hence, $\lfloor \log_{10} d \rfloor + 1$ is $\|d\|$.

Now, let there be a 7 digit binary number $b = 1001000$, which expanded is:

$$(1 \cdot 2^6) + (0 \cdot 2^5) + (0 \cdot 2^4) + (1 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) = 72,$$

Taking 6 powers of 2 to reach 72, we add 1 to get $\|b\| = 7$. Hence, $\|b\| = \lfloor \log_2 b \rfloor + 1$. Additionally, if $a = 0_2$ then $\|a\| = 1$. as $a^0 = 1$.

Theorem 5.2: Splitting Higher and Lower Bits

Let a be a binary number with n bits. We can split a into two numbers A_1 and A_0 with $n/2$ bits each, representing the first and second halves respectively. Where:

$$A_1 := \frac{a}{2^{\lceil n/2 \rceil}} \quad \text{and} \quad A_0 := a \bmod 2^{\lceil n/2 \rceil}$$

Example: Let's start with base 10. To achieve $A_1 = 7455$ and $A_0 = 62,010$, for $a = 745,562,010$. we take the length $\|a\| := \lfloor \log_{10}(745,562,010) \rfloor + 1 = 9$, as $10^8 \leq 745,562,010 < 10^9$. Then:

$$A_1 = \frac{745,562,010}{10^{\lceil 9/2 \rceil}} = 7455, \quad \text{and} \quad A_0 = 745,562,010 \bmod 10^{\lceil 9/2 \rceil} = 62,010$$

as $10^5 \leq 62,010 < 10^6$. Likewise to finding the remainder in base 2, we can use the same bit shifting technique for base 10 (4.1). We see,

$$[745,562,010]_{10} \text{ right shift by } 5, \ [000,007,455]_{10} \ 62,010.$$

Hence, 62,010 is pushed out, and our remainder. Then, we can apply the same technique to base 2. Let $a = 1111\ 1111\ 1001\ 1001_2$. We have $\|a\| := 16$, then:

$$A_1 = \frac{1111\ 1111\ 1001\ 1001_2}{2^{\lceil 16/2 \rceil}} = 1111\ 1111_2, \text{ and } A_0 = 1111\ 1111\ 1001\ 1001_2 \bmod 2^{\lceil 16/2 \rceil} = 1001\ 1001_2$$

Scenario - *Divide and Conquer Multiplication*: We are to compute,

$$A_1 2^{\lceil n/2 \rceil} + A_0 =: a \quad \times \quad b := B_1 2^{\lceil n/2 \rceil} + B_0.$$

Then we have,

$$\begin{aligned} a \cdot b &= (A_1 2^{\lceil n/2 \rceil} + A_0)(B_1 2^{\lceil n/2 \rceil} + B_0) \\ &= (A_1 2^{\lceil n/2 \rceil})(B_1 2^{\lceil n/2 \rceil}) + (A_1 2^{\lceil n/2 \rceil})B_0 + (B_1 2^{\lceil n/2 \rceil})A_0 + A_0 B_0 \\ &= (A_1 B_1) 2^n + (A_1 B_0 + B_1 A_0) 2^{\lceil n/2 \rceil} + A_0 B_0. \end{aligned}$$

We need to compute 4 products, (A_1B_1), $(A_1B_0), (B_1A_0)$, and (A_0B_0). We now attempt to solve them independently:

Function 5.1: Multiplication of n-bit Integers - *Multiply()*

Let a and b be n-bit integers of base 2. This algorithm recursively computes the product of a and b using a straightforward divide-and-conquer approach, without using Karatsuba's optimization.

Input: n, a, b (where a and b are n-bit integers)
Output: The product $a \times b$

```
Function Multiply($n, a, b$):
    if $n < 2$ then
        return the result of grade-school multiplication for $a \times b$;
    else
        $A_1 \leftarrow a \div 2^{n/2}$;  $A_0 \leftarrow a \bmod 2^{n/2}$;
        $B_1 \leftarrow b \div 2^{n/2}$;  $B_0 \leftarrow b \bmod 2^{n/2}$;
        $p_1 \leftarrow Multiply(n/2, A_1, B_1)$;
        $p_2 \leftarrow Multiply(n/2, A_1, B_0)$;
        $p_3 \leftarrow Multiply(n/2, A_0, B_1)$;
        $p_4 \leftarrow Multiply(n/2, A_0, B_0)$;
        return $p_1 \cdot 2^n + (p_2 + p_3) \cdot 2^{n/2} + p_4$;
```

Time Complexity: $O(n^2)$, as in our master method $T(n) = 4T(n/2) + O(n)$, Theorem (**??**).
Space Complexity: $O(n)$, storing $n + n$ bits for a and b, while we track $O(\log_2 n)$ depth in the recursion stack.

We appear to make no improvement, however there's a small trick to reduce the number of multiplications. We continue on the next page.

Observe our full term, $c := (A_1B_1)2^n + (A_1B_0 + B_1A_0)2^{\lceil n/2 \rceil} + A_0B_0$. Say we computed another term,

$$z := (A_1 + A_0)(B_1 + B_0) = (A_1B_1) + (A_1B_0) + (B_1A_0) + (A_0B_0).$$

Notice how z also contains (A_1B_1) and (A_0B_0), which are also in c. Say $m = (A_1B_0) + (B_1A_0)$. Let $x := (A_1B_1)$ and $y := (A_0B_0)$ then $z - x - y = m$. This reduces the number of multiplications to 3, as we only compute (A_1B_1), (A_0B_0) once, and then z.

We employ the above strategy, which is **Karatsuba's multiplication algorithm**:

Function 5.2: Karatsuba's Multiplication Algorithm - *KMul()*

Let a and b be n-bit integers of base 2. This algorithm recursively computes the product of a and b using a divide-and-conquer approach.

Input: n, a, b (where a and b are n-bit integers)
Output: The product $a \times b$

```
Function Multiply($n, a, b$):
    if $n < 2$ then
        return the result of grade-school multiplication for $a \times b$;
    else
        $A_1 \leftarrow a \div 2^{n/2}$;  $A_0 \leftarrow a \bmod 2^{n/2}$;
        $B_1 \leftarrow b \div 2^{n/2}$;  $B_0 \leftarrow b \bmod 2^{n/2}$;

        $x \leftarrow Multiply(n/2, A_1, B_1)$;
        $y \leftarrow Multiply(n/2, A_0, B_0)$;
        $z \leftarrow Multiply(n/2, A_1 + A_0, B_1 + B_0)$;
        return $x \cdot 2^n + (z - x - y) \cdot 2^{n/2} + y$;
```

Time Complexity: $O(n^{\log_2 3}) \approx O(n^{1.585})$, as in our master method $T(n) = 3T(n/2) + O(n)$, Theorem (**??**).
Space Complexity: $O(n)$.

— 2 —

Networking Fundamentals

2.1 The Internet

Terminology and concepts of the internet, which will be used throughout this text.

Definition 1.1: Protocol

A **protocol** is a set of rules which govern the exchange of data between devices. Protocols define the format, timing, sequencing, and error control of data transmission [31].

Definition 1.2: Internet

The **Internet** is a global network of distributed system communicating over an **Internet Protocol** (IP) [12]. Documents served over the internet are referred to as **webpages** or **websites**.

Definition 1.3: HTTP & HTML

HTTP (HyperText Transfer Protocol), the protocol which transfer data over the internet, distributing **HTML** (HyperText Markup Language) documents. Such documents include **hyperlinks** to other websites, images, and other media [18].

Definition 1.4: RFC (Request for Comments)

RFC (Request for Comments) is a publication from the **Internet Engineering Task Force** (IETF) and the **Internet Society** (ISOC). This body governs the specifications for the internet and its protocols [35].

Definition 1.5: DNS and IP Addresses

An **Internet Protocol address** (IP address) is a unique identifier for a device on a network. The **Domain Name System** (DNS) maps domain names to IP addresses [1].

Definition 1.6: Web Browser

A **web browser** is a software application for accessing the **World Wide Web** (WWW) [36].

Definition 1.7: URL (Uniform Resource Locator)

A **URL** (Uniform Resource Locator) references each webpage, specifying protocol, domain, and path [38]. E.g., `http://www.example.com/path/to/resource`.

- **Protocol**: `http`
- **Domain**: `www.example.com`
- **Path**: `/path/to/resource`

Definition 1.8: Client-Server Model

Most of the internet operates on a **client-server model**, where an agent device–the **client**–requests data from another agent–the **server**–which serves an appropriate response. Clients are not servers and vice versa, as they receive and interpret data differently [9].

Definition 1.9: HTTP Methods

When a client makes a request to a server, they must specify their intent, categorized by **HTTP methods** [17]:

- **GET**: Retrieve data from the server.
- **POST**: Send data to the server.
- **PUT**: Update data on the server.
- **DELETE**: Remove data from the server.

Definition 1.10: HTTP Headers

HTTP headers are key-value pairs sent between the client and server to provide **metadata** about the request or response. **Metadata** is data about the transmitted data, telling the receiver how the incoming data should be interpreted [17].

Tim Berners-Lee and his team at CERN developed the first web server and browser in 1989 [39].

HTTP Version	**Description**
HTTP/0.9 (1991)	Only supports GET method (retrieving HTML alone).
HTTP/1.0 (1996)	RFC#1945, adding support for metadata in HTTP headers, status codes, and POST and HEAD methods [4].
HTTP/1.1 (1997)	Defined in RFC#2068 and later updated by RFC#2616, introduced persistent connections, chunked transfer encoding, and additional cache control mechanisms [16][17].
HTTP/2 (2015)	RFC#7540, improving performance by enabling request and response multiplexing, header compression, and prioritization [3].
HTTP/3 (2022)	Builds upon HTTP/2's features and uses the QUIC transport protocol to reduce latency and improve security. [5]

Table 2.1: Evolution of HTTP Versions

Note: In short, **Persistent Connections** allow multiple requests and responses to be sent over a single connection, reducing latency and improving performance [17]. **Chunked Transfer Encoding** allows the server to send data in chunks, enabling the client to start processing data before the entire response is received [17]. **Multiplexing**, is the ability to send multiple requests and responses over a single connection, reducing latency and improving performance [15]. **QUIC** will be discussed alter on with other transfer protocols in a later section.

2.2 Data Transmission

This section details how internet traffic is transmitted between devices.

Definition 2.1: ISO Model

The **ISO model** (International Organization for Standardization) is a conceptual framework for transmitted data between devices. It is divided into seven layers of function[8]. Published in 1984 by the International Organization for Standardization (ISO) [23].

Definition 2.2: TCP/IP Model

The **TCP/IP model** (Transmission Control Protocol/Internet Protocol) is a concise representation of the ISO model used in practical settings [40].

Definition 2.3: ISO Layers

1. **Physical**: Converts data into physical signals (e.g., electrical, optical, or radio waves) for transmission across the network medium (e.g., cables, fiber optics, or wireless channels).
2. **Data Link**: local delivery of directly connected devices within **Local Area Networks** (LAN) using **Media Access Control** (MAC) addresses for addressing.
3. **Network**: Handles addressing, routing, in external networks from source to destination.
4. **Transport**: Ensures end-to-end delivery, via a message delivery protocol.
5. **Session**: Initiates and terminates network connections, ensuring efficient resource usage.
6. **Presentation**: To translate, compress, and encrypt data (e.g., Operating Systems).
7. **Application**: User facing services such as, HTTP , FTP, DNS, SMTP, etc.

[25][33]

Note: Many of the above layers are closely related, if not identical. In practice, layers 5-6 are integrated into layer 7, and layers 1-2 are often combined into a single layer in the TCP/IP model.

Definition 2.4: TCP/IP Layers

1. **Network Interface**: Physical and data link layers from ISO.
2. **Internet**: Attaches IP addresses to data packets for routing across the internet.
3. **Transport**: Defines the delivery protocol, segmenting data into packets.
4. **Application**: The Session, Presentation, and Application layers from ISO.

[33]

Despite the numbering of the layers, the user interacts with the application layer, which communicates down the chain of layers to the physical layer, where the data is transmitted over the network medium. The receiving device then interprets the data, moving back up the chain to the application layer.

To illustrate the contrast between the ISO and TCP/IP models, consider the diagram:

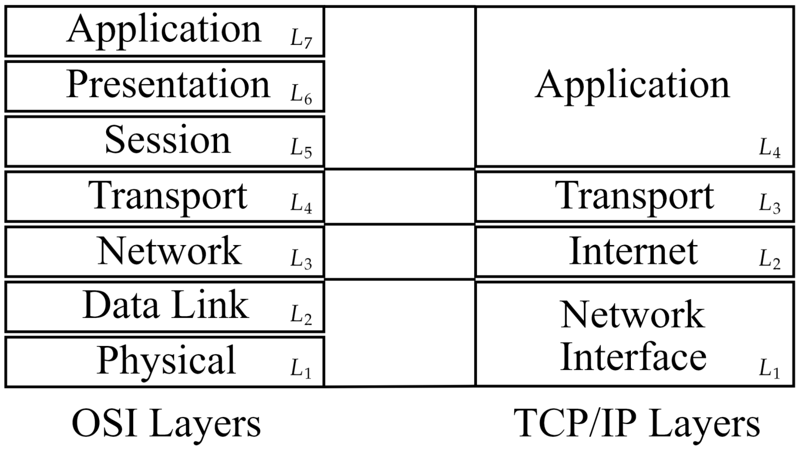


Figure 2.1: ISO vs TCP/IP Model

To illustrate two devices communicating over the internet, consider the diagram:

2.3 Routing Networks

When IP addresses began

Definition 3.1: Routing

Routing is the process of selecting the best path across networks. Data is segmented into packets, each with a destination address. **Routers** are devices which forward this data through the network.

Routers have a **routing table** which maps to other reachable networks. When a packet arrives, the router checks against its routing table to find the best path. [13]

Definition 3.2: Hop-by-Hop & End-to-End Routing

- **Hop-by-Hop Routing**: When a packet of data is forwarded from one router to the next, a forward decision is called a **hop**.
- **End-to-End Routing**: The process of sending data from source to destination without intermediate hops.

It is often rare to see end-to-end routing in modern networks, as data is often forwarded through multiple routers. A target destination may be unreachable from a given router. [20]

Definition 3.3: Router Advertising

When routers inform each other of their existence and the networks they can reach [28].

Definition 3.4: Routing Protocols

- **IP** (Internet Protocol): The primary protocol for routing data across the internet.
- **BGP** (Border Gateway Protocol): The protocol for routing data between **Autonomous Systems** (AS). an AS is a collection of IP networks and routers under the control of a single entity (e.g., an **ISP** (Internet Service Provider)). These may only connect with each other if they have a mutual agreement. ASes identify themselves to external networks using a unique **Autonomous System Number** (ASN). These are unique 16 bit numbers between 1-65534 or 32 bit numbers between 131072-4294967294 (e.g., `AS12345`) [11].
- **OSPF** (Open Shortest Path First): A link-state routing protocol used within an AS. Link-state protocols are a set of algorithms which determine the best path, based on the topology of a network graph [24]. It is also an **IGP** (Interior Gateway Protocol), meaning it operates within a single AS. It does so by sending out **LSAs** (Link State Advertisements) to other routers in the AS. Then routers in the system build a **LSADB** (Link State Advertisement Database) of the network topology. Then a shortest path algorithm is run to determine the best path to each network [6].
- **RIP** (Routing Information Protocol): RIP employs hop count as a routing metric, with a maximum allowable hop count of 15 (network size limitation). It operates as an **IGP** within a single AS, periodically broadcasting the entire routing table to neighboring routers every 30 seconds, which can lead to slower convergence and higher bandwidth usage compared to other protocols. RIP is largely deprecated [22].

Definition 3.5: IP Addressing

IP addresses are unique identifiers for devices on a network. There are two versions of IP addresses, **IPv4** and **IPv6**. IPv4 A 32-bit address (2^{32} addresses), employed since 1983, quickly exhausted all available addresses by the 2010s [26]. IPv6 is a 128-bit address (2^{128} addresses), introduced in 1998 , in an attempt to address this shortage [14][21]. For example,

- **IPv4**: a decimal octet "$x.x.x.x$": $x \in [0, 255]$ (e.g., `192.168.1.1`).
- **IPv6**: a hexadecimal segment "$y : y : y : y : y : y : y : y$": $y \in [0, FFFF]$ (e.g., `2001:0db8:85a3:0000:0000:8a2e:0370:7334`).

Definition 3.6: Subnetting

Instead of a large monolith network of routers, networks can be divided into smaller networks called **subnets**. I.e., Instead of passing data to every device on a network, routers forward data to a representative device on each subnet. [10]

Definition 3.7: Subnet Masking

A **subnet mask** defines which part of an IP address identifies the **network** and which part identifies the **host**.

Definition 3.8: Classful Network

In the beginning, the first octet of an IPv4 address determined the network class—only allowing for 256 networks. The RFC#791 published in 1981 introduced **Classful Networks** [30]. It uses the first three bits of the first octet's binary representation as a subnet mask to determine a class ranging from A-E. Th

Class	**Binary Prefix**	**Range (Decimal)**	**Purpose**	**Details**
A	`0xx`	1.0.0.0 to 126.0.0.0	Unicast (large networks)	For large organizations; 8 bits for the network, 24 for hosts.
B	`10x`	128.0.0.0 to 191.255.0.0	Unicast (medium networks)	For medium-sized networks; 16 bits for the network, 16 for hosts.
C	`110`	192.0.0.0 to 223.255.255.0	Unicast (small networks)	For small networks; 24 bits for the network, 8 for hosts.
D	`1110`	224.0.0.0 to 239.255.255.255	Multicast	Reserved for multicast addressing; not for general use.
E	`1111`	240.0.0.0 to 255.255.255.255	Experimental and future use	Reserved for research and development; not assigned for standard use.

Table 2.2: Overview of IPv4 Address Classes

Definition 3.9: Fixed Length Subnet Masking (FLSM)

Fixed Length Subnet Masking (FLSM) is a technique which divides a network into equal-sized subnets. This may lead to inefficient use of IP addresses. [2]

Definition 3.10: Variable Length Subnet Masking (VLSM)

Variable Length Subnet Masking (VLSM) is a technique which allows for the creation of subnets with different sizes. As some ASes may require more IP addresses than others, VLSM allows for more efficient use of IP addresses. [2]

Definition 3.11: Classless Inter-Domain Routing (CIDR)

Classless Inter-Domain Routing (CIDR), introduced in 1993 through RFC#1518 and RFC#1519 to address IPv4 exhaustion. **CIDR replaced classful subnetting** with VLSM. CIDR notation is written as `IP Address/Prefix Length` (e.g., `192.168.1.0/24`), where:

- `IP Address`: Represents the starting address of the network.
- `Prefix Length`: The number of bits used for the **network portion** of the address.

For example:

`255.0.0.0/`**`8`**; `255.255.0.0/`**`16`**; `255.255.255.0/`**`24`**; `255.255.255.192/`**`26`**;

[19]

Definition 3.12: Route Aggregation

CIDR introduced **Route Aggregation** also known as **Supernetting**, or **Route Summarization**, is the process of combining multiple routes into a single route advertisement. **Example**: Consider an organization assigned the following contiguous IP address blocks:

`192.168.1.0/24`; `192.168.2.0/24`; `192.168.3.0/24`; `192.168.4.0/24`;

Each block holding 256 IP addresses with a subnet mask of 255.255.255.0, requiring four routing table entries. However, these networks share a common prefix: the first 22 bits (`192.168.0.0/22`), which aggregates to: **`192.168.0.0/22`** [19].

Definition 3.13: IP Address Components

Network Address:

- Identifies the specific network segment to which a device is connected.
- Determined by setting all bits in the host portion to 0.
- Example: For the IP address `192.168.1.10` with a subnet mask of `255.255.255.0` (/24), the network address is `192.168.1.0`.

Host Address:

- Uniquely identifies a device within a network segment.
- The bits in the IP address designated for hosts, specified by the subnet mask.
- Example: In the IP address `192.168.1.10` with a /24 subnet mask, the host portion is the last octet (10).

Broadcast Address:

- Used to communicate with all devices on a specific network segment simultaneously.
- Determined by setting all bits in the host portion to 1.
- Example: For the network `192.168.1.0/24`, the broadcast address is `192.168.1.255`.

[34] [32] [27]

Consider the IP address `192.168.2.12/26` and its binary `11000000.10101000.00000010.00001100`:

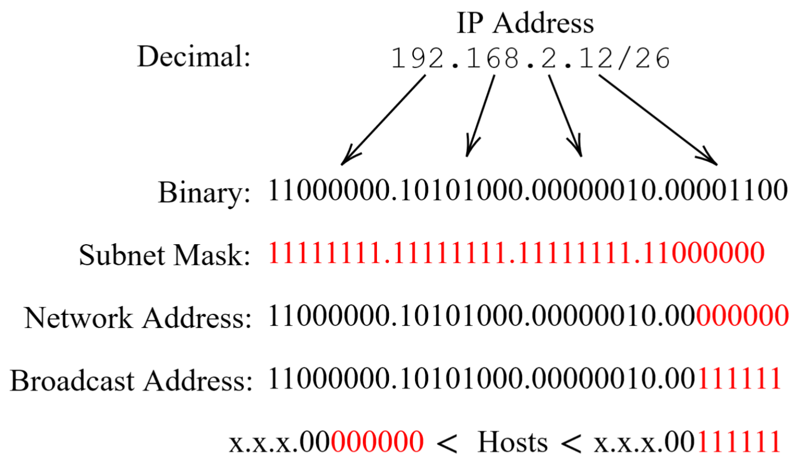

IP Address
Decimal: 192.168.2.12/26
Binary: 11000000.10101000.00000010.00001100
Subnet Mask: 11111111.11111111.11111111.11000000
Network Address: 11000000.10101000.00000010.00000000
Broadcast Address: 11000000.10101000.00000010.00111111
x.x.x.00000000 < Hosts < x.x.x.00111111

Figure 2.2: Binary Subnetting: Red indicating parts of the IP address identified for each component.

Definition 3.14: Common Types of Area Networks (ANs)

PAN (Personal Area Network): A network for personal devices, such as smartphones, smartwatches, or earbuds, with a short range (typically a few meters) using technologies like Bluetooth or infrared.

LAN (Local Area Network): Connects devices within a small area, such as a home, office, or school, enabling high-speed communication and resource sharing.

WLAN (Wireless Local Area Network): A wireless version of LAN that uses Wi-Fi to connect devices within a localized area like a home or office.

CAN (Campus Area Network): A network that connects multiple LANs across a limited geographical area, such as a university campus or corporate facility, for resource sharing and communication.

MAN (Metropolitan Area Network): Covers a larger area than a LAN, typically a city or metropolitan region, connecting multiple LANs or CANs via high-speed infrastructure like fiber optics.

WAN (Wide Area Network): Connects LANs, MANs, or other networks over large geographical areas, such as countries or continents. The internet is the largest WAN example.

SAN (Storage Area Network): A high-speed network that provides access to storage devices for data centers, ensuring fast and reliable storage management.

EPN (Enterprise Private Network): A private network created by organizations to connect their various locations securely, often including VPNs for remote access.

VPN (Virtual Private Network): Creates a secure, encrypted connection over public networks, like the internet, to allow users to access private networks remotely.

HAN (Home Area Network): A network within a home environment, connecting personal devices like computers, printers, and smart home gadgets.

GAN (Global Area Network): A large-scale network that connects multiple WANs and supports worldwide communication, with the internet as its most prominent example.

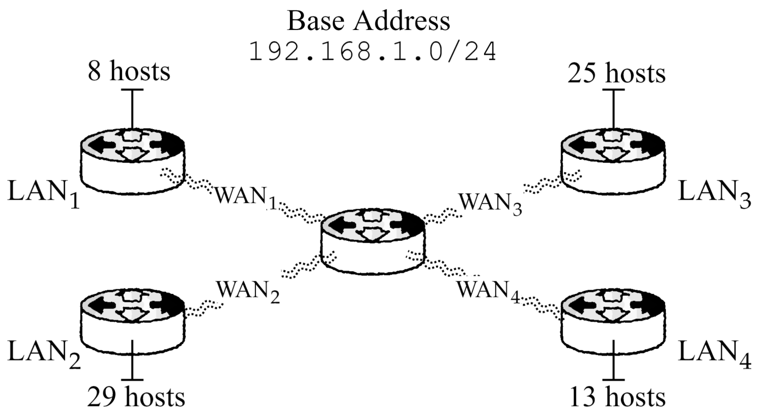
[29]

Example 3.1: Subnetting a Network via VLSM

Consider the FLSM below, which all have 62 hosts per network:

Network Address	Hosts	Broadcast Address
192.168.100.0	.1 – .62	.63
192.168.100.64	.65 – .126	.127
192.168.100.128	.129 – .190	.191
192.168.100.192	.193 – .254	.255

Below illustrates this network, where a router of base address `192.168.1.0/24`, connects four LANs:



Subnetting: Process each LAN from largest to smallest, Select the nearest block size, identify the network, host, and broadcast addresses. Since there is a subnet mask of /24, blocks $[128, 64, 32, 16, 8, 4, 2, 1]$ are available. This is the case as $2^8 = 256$. If a LAN has 129 hosts, that LAN will occupy all 254 addresses (as x.x.x.0 and x.x.x.255 are reserved).

1. **LAN$_2$**: 29 hosts ⇒ Block size 32. Network: `x.0`, Broadcast: `x.31`, Hosts: `x.1-x.30`.
2. **LAN$_3$**: 25 hosts ⇒ Block size 32. Network: `x.32`, Broadcast: `x.63`, Hosts: `x.33-x.62`.
3. **LAN$_4$**: 13 hosts ⇒ Block size 16. Network: `x.64`, Broadcast: `x.79`, Hosts: `x.65-x.78`.
4. **LAN$_1$**: 8 hosts ⇒ Block size 16. Network: `x.80`, Broadcast: `x.95`, Hosts: `x.81-x.94`.

8 hosts need occupy 16, as a block size of 8 only has 6 usable addresses. The computed subnet now only occupies addresses `x.0-x.95`, leaving room for additional LANs [7].

■

Bibliography

[1] DoD standard Internet Protocol. RFC 760, January 1980.

[2] Rahul Awati. Variable length subnet mask (vlsm). TechTarget, October 2021. Last updated October 2021, accessed November 2024.

[3] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). Request for Comments, May 2015. Category: Standards Track.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. Request for Comments, May 1996. Category: Informational.

[5] M. Bishop. HTTP/3. Request for Comments, June 2022. Category: Standards Track.

[6] CertBros. Ospf explained | step by step. YouTube, n.d. Accessed November 2024.

[7] East Charmer. Easy vlsm subnetting | step by step vlsm. https://www.youtube.com/watch?v=IgthYZ9N1vs&ab_channel=EastCharmer, n.d. Accessed November 2024.

[8] Chrystal R. China and Michael Goodwin. What is the osi model? IBM Think, June 2024. Published: 11 June 2024.

[9] Cloudflare Learning Center. What do client side and server side mean?, n.d. Accessed: 2024-11-27.

[10] Cloudflare Learning Center. What is a subnet? | how subnetting works. Cloudflare, n.d. Accessed November 2024.

[11] Cloudflare Learning Center. What is an autonomous system? | what are asns? Cloudflare, n.d. Accessed November 2024.

[12] Cloudflare Learning Center. What is internet protocol (ip)?, n.d. Accessed: 2024-11-27.

[13] Cloudflare Learning Center. What is routing? | ip routing. Cloudflare, n.d. Accessed November 2024.

[14] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. Technical Report RFC 8200, Internet Engineering Task Force (IETF), July 2017. Internet Standard 86, obsoletes RFC 2460.

[15] Editorial Team. Multiplexing. *Network Encyclopedia*, October 2023. Last edited October 5, 2023.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments, January 1997. Category: Standards Track.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments, June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, and 7235. Errata exist.

[18] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.

[19] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (cidr): An address assignment and aggregation strategy. Technical Report RFC 1519, Internet Engineering Task Force (IETF), September 1993. Accessed November 2024.

[20] Simon Heimlicher, Pavan Nuggehalli, and Martin May. End-to-end vs. hop-by-hop transport. In *Proceedings of the ACM SIGMETRICS*. Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland; Centre for Electronics Design and Technology, Indian Institute of Science, Bangalore, India, 2007. Accessed November 2024.

[21] IBM Documentation. Ipv4 and ipv6 address formats. IBM Documentation, January 2022. Last updated 2022-01-18, accessed November 2024.

[22] Javatpoint. Rip protocol. Javatpoint, n.d. Accessed November 2024.

[23] Vijay Kanade. What is the osi model? definition, layers, and importance. Spiceworks, November 2022. Accessed November 2024.

[24] Jim Kurose. Computer networks and the internet: Routing algorithms and link state routing. YouTube, n.d. Video presentation for Computer Networking: A Top-Down Approach (8th edition), J.F. Kurose and K.W. Ross, Pearson, 2020. Taught at University of Massachusetts Amherst.

[25] Samson Leonard. Lecture 1: The osi model. SlidePlayer, 2014. Modified over 9 years ago. References: TCP/IP Protocol Suite, 4th Edition (Chapter 2).

[26] Kwun-Hung Li and Kin-Yeung Wong. Empirical analysis of ipv4 and ipv6 networks through dual-stack sites. *Information*, 12(6), 2021.

[27] B. Manning and P. Perkins. Variable length subnet table for ipv4. Technical Report RFC 1878, Internet Engineering Task Force (IETF), December 1995. Informational.

[28] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor discovery for ip version 6 (ipv6). Technical Report RFC 4861, Internet Engineering Task Force (IETF), September 2007. Standards Track, Obsoletes RFC 2461.

[29] Steve Petryschuk. Types of networks: Lan, wan, man, and more. https://www.auvik.com/franklyit/blog/types-of-networks/?utm_source=chatgpt.com, n.d. Accessed November 2024.

[30] J. Postel. Internet protocol - darpa internet program protocol specification. Technical Report RFC 791, Internet Engineering Task Force (IETF), September 1981. Internet Standard 5, obsoletes RFC 760, updated by RFCs 1349, 2474, and 6864.

[31] J. Postel. Internet protocol: Darpa internet program protocol specification. Technical Report RFC 791, Defense Advanced Research Projects Agency (DARPA), Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA, September 1981. Obsoleted by RFC 1349, RFC 2474, RFC 6864.

[32] J. Postel. Broadcasting internet datagrams in the presence of subnets. Technical Report RFC 922, Internet Engineering Task Force (IETF), October 1984. Standards Track.

[33] Ammar Rayes and Samer Salam. *The Internet in IoT*, pages 35–62. Springer International Publishing, Cham, 2022.

[34] C. Sunshine and J. Postel. Broadcasting internet datagrams. Technical Report RFC 919, Internet Engineering Task Force (IETF), October 1984. Standards Track.

[35] Martin Thomson and Francesca Palombini. The rfc series and rfc editor. RFC 9280, June 2022.

[36] University of Oklahoma. History of the world wide web, n.d. Accessed: 2024-11-27.

[37] Concise Works. Sect modules. https://github.com/Concise-Works/sect-modules, n.d. Accessed November 2024.

[38] World Wide Web Consortium (W3C). Html working group draft: Href and url standards, n.d. Accessed: 2024-11-27.

[39] World Wide Web Consortium (W3C). Hypertext transfer protocol (http) as implemented in w3, n.d. Accessed: 2024-11-27.

[40] Kinza Yasar, Mary E. Shacklett, and Amy Novotny. What is tcp/ip? TechTarget, September 2024. Last updated September 2024.