**Time and Space Complexity (Big O)**



* **Asymptotically Tight Bound:** θ(g(n)) tightly bounds f(n) asymptotically because it has upper and lower bounds.
* **Asymptotically Nonnegative:**  N exists such that f(n) >= 0 for all n >= 0
* **Asymptotically Positive:** N exists such that f(n) > 0 for all n >= n

**ASCII**

**American Standard Code for Information Interchange**

**Regular ASCII only has a total of 128 characters. Extended ASCII has a total of 256 characters.**

Any ASCII text is UTF-8 text.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ASCII | Decimal | Hexadecimal | Octal | Binary |
| A | 65 | 41 | 101 | 1000001 |
| B | 66 | 42 | 102 | 1000010 |
| C | 67 | 43 | 103 | 1000011 |
| D | 68 | 44 | 104 | 1000100 |
| E | 69 | 45 | 105 | 1000101 |
| F | 70 | 46 | 106 | 1000110 |
| G | 71 | 47 | 107 | 1000111 |
| H | 72 | 48 | 110 | 1001000 |
| I | 73 | 49 | 111 | 1001001 |
| J | 74 | 4A | 112 | 1001010 |
| K | 75 | 4B | 113 | 1001011 |
| L | 76 | 4C | 114 | 1001100 |
| M | 77 | 4D | 115 | 1001101 |
| N | 78 | 4E | 116 | 1001110 |
| O | 79 | 4F | 117 | 1001111 |
| P | 80 | 50 | 120 | 1010000 |
| Q | 81 | 51 | 121 | 1010001 |
| R | 82 | 52 | 122 | 1010010 |
| S | 83 | 53 | 123 | 1010011 |
| T | 84 | 54 | 124 | 1010100 |
| U | 85 | 55 | 125 | 1010101 |
| V | 86 | 56 | 126 | 1010110 |
| W | 87 | 57 | 127 | 1010111 |
| X | 88 | 58 | 130 | 1011000 |
| Y | 89 | 59 | 131 | 1011001 |
| Z | 90 | 5A | 132 | 1011010 |
| ASCII | Decimal | Hexadecimal | Octal | Binary |
| a | 97 | 61 | 141 | 1100001 |
| b | 98 | 62 | 142 | 1100010 |
| c | 99 | 63 | 143 | 1100011 |
| d | 100 | 64 | 144 | 1100100 |
| e | 101 | 65 | 145 | 1100101 |
| f | 102 | 66 | 146 | 1100110 |
| g | 103 | 67 | 147 | 1100111 |
| h | 104 | 68 | 150 | 1101000 |
| i | 105 | 69 | 151 | 1101001 |
| j | 106 | 6A | 152 | 1101010 |
| k | 107 | 6B | 153 | 1101011 |
| l | 108 | 6C | 154 | 1101100 |
| m | 109 | 6D | 155 | 1101101 |
| n | 110 | 6E | 156 | 1101110 |
| o | 111 | 6F | 157 | 1101111 |
| p | 112 | 70 | 160 | 1110000 |
| q | 113 | 71 | 161 | 1110001 |
| r | 114 | 72 | 162 | 1110010 |
| s | 115 | 73 | 163 | 1110011 |
| t | 116 | 74 | 164 | 1110100 |
| u | 117 | 75 | 165 | 1110101 |
| v | 118 | 76 | 166 | 1110110 |
| w | 119 | 77 | 167 | 1110111 |
| x | 120 | 78 | 170 | 1111000 |
| y | 121 | 79 | 171 | 1111001 |
| z | 122 | 7A | 172 | 1111010 |

**UNICODE**

**Universal Coded Character Set**

Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world’s writing systems. The standard is maintained by the Unicode Consortium. It contains 137,994 characters. However, it can contain up to 1,112,064 valid coding points.

Unicode is backwards compatible with ASCII.

Unicode is a 21-bit character encoding that uniquely describes “Code Points”, each code point is represented by a glyph (graphical representation).

* 16 bits are used to identify the code point in a plane.
* 5 bits are used to identify the plane.

**Encoding – Mapping of code points to bit patterns.**

**UTF – Unicode Transformation Format**

Because Unicode characters don't generally fit into one 8-bit byte, there are numerous ways of storing Unicode characters in byte sequences, such as UTF-32 and UTF-8.

* UTF-8 🡺 Dominant in the World Wide Web 🡺 Uses 1 byte for the first 128 code points, up to 4 bytes for other characters. 8-bit Unicode Transformation Format. Code points with lower numerial values, which tend to occur more frequently, are encoded using fewer bytes. It is mostly used by UNIX/Linux and Mac OS X.
* UTF-16 🡺 Uses 2 bytes for each character + 4 bytes for other planes. It is mostly used by Windows Computers.
* UTF-32 🡺 Uses 4 bytes for each character. Obviously takes significant more space.
* UCS-2 🡺 Uses 2 bytes for each character without full support for Unicode, can only represent less than half of all the encoded Unicode characters. “Universal Coded Character Set (UCS)”.
* Mac OS Roman 🡺 Used by old legacy Macintosh Computers and by Java for Mac OS X.
* **1 bit = 0 | 1**
* **1 byte = 8 bits**
* **2 bytes = 16 bits**
* **1 KB = 1,024 bytes**
* **1 MB = 1,024 KB 🡺 (1024 x 1024 bytes)**
* **1 GB = 1,024 MB 🡺 (1024 x 1024 x 1024 bytes)**
* **1 TB = 1,024 GB 🡺 (1024 x 1024 x 1024 x 1024 bytes)**
* **1 PB = 1,024 TB (Petabyte)**
* **1 EB = 1,024 PB (Exabyte)**
* **1 ZB = 1,024 EB (Zettabyte)**
* **1 YB = 1,024 YB (Yottabyte)**

Computers count by base 2:

21 = 2

22 = 2\*2 = 4

23 = 2\*2\*2 = 8

210 = 1,024

220 = 1,048,576

So in computer jargon, the following units are used:

| **Unit** | **Equivalent** |
| --- | --- |
| 1 kilobyte (KB) | 1,024 bytes |
| 1 megabyte (MB) | 1,048,576 bytes |
| 1 gigabyte (GB) | 1,073,741,824 bytes |
| 1 terabyte (TB) | 1,099,511,627,776 bytes |
| 1 petabyte (PB) | 1,125,899,906,842,624 bytes |

**BINARY AND HEXADECIMAL NUMBERS**

|  |  |  |
| --- | --- | --- |
| **Decimal** | **Binary** | **Hexadecimal** |
| 0 | 00000000 | 0 |
| 1 | 00000001 | 1 |
| 2 | 00000010 | 2 |
| 3 | 00000011 | 3 |
| 4 | 00000100 | 4 |
| 5 | 00000101 | 5 |
| 6 | 00000110 | 6 |
| 7 | 00000111 | 7 |
| 8 | 00001000 | 8 |
| 9 | 00001001 | 9 |
| 10 | 00001010 | A |
| 11 | 00001011 | B |
| 12 | 00001100 | C |
| 13 | 00001101 | D |
| 14 | 00001110 | E |
| 15 | 00001111 | F |
| 16 | 00010000 | 10 |
| … | … | … |
| 255 | 11111111 | FF |

**OR**

**0 | 0 = 0**

**0 | 1 = 1**

**1 | 1 = 1**

**AND**

**0 & 0 = 0**

**0 & 1 = 0**

**1 & 1 = 1**

**XOR**

**0 ^ 0 = 0**

**0 ^ 1 = 1**

**1 ^ 1 = 0**

**One’s Complement = Negation**

**~1 = 0**

**~0 = 1**

**Endianness: Refers to the order of bytes within a binary representation of a number.**

**Big-Endian: Places the most significant byte first and the least significant byte last. Dominant ordering in networking protocols.**

**Little-Endian: Places the most significant byte last and the least significant byte first. Dominant ordering for processor architectures.**

TWO’S COMPLIMENT

Two’s Compliment is a mathematical operation on binary numbers and is an example of a radix complement. It is used in computing as a method of signed number representation.

The two’s complement is calculated by inverting the digits and adding one.

010 🡺 110 because 010 + 110 = 1000

Steps:

1. Write the number in binary.
2. Invert the digits: 0 becomes 1 and 1 becomes 0.
3. Add 1.

* Binary: 0 0 0 1 1 1 0 0
* Inversion: 1 1 1 0 0 0 1 1
* Add 1: 1 1 1 0 0 1 0 0

So original number was 28 and new number is -28. A leading zero means positive and a leading one means negative.

Logic shift right 🡺 Inserts a 0 in the sign bit even if it was a negative number and shifts the entire bits to the right. Uses symbol “>>>”.

Arithmetic shift right 🡺 Keeps the sign bit and shifts the bits starting from the one that follows the sign bit. Uses symbol “>>”.

**BIT MANIPULATION**

<http://www.goldsborough.me/bits/c++/low-level/problems/2015/10/11/23-52-02-bit_manipulation/>

1. Set bit: x | (1 << n)
2. Clear bit: x & ~(1 << n)
3. Update bit:
   1. Clear bit 🡺 x & ~(1 << n)
   2. OR bit 🡺 x | (value << n)

<https://catonmat.net/low-level-bit-hacks>