Modular Arithmetic

- cyclic

$\frac{a}{b} = qb + r$

quotient → qRr ← remainder

divisor → $b\overline{)a}$ ← dividen

- clock is mod 12

- the result of finding a congruence mod M, is the same value as the one's digit base M

EX: $10 + 4 \pmod{12} \equiv 2$

$\frac{10+4}{12} \rightarrow 12\overline{)14}$ R2

EX: mod 4

3 0 1 2

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

diagonal of 1 less than modulus

even diagonals

lots of patterns

EX: mod 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

the output is the remainder when divided by the modulus

String Operations

- **Reverse-**
  - Input: a string
  - Output: a string of the same length w/ the characters in opposite order
  - Example: Reverse(1011)=1101
- **Compliment (Two's Compliment)-**
  - Input: Binary String
  - Output: Binary string where all of the origional ones are changed to zeros and the origional zeros are changed to ones
  - Example: Compliment(1011 0001) = 0100 1110
- **Transpose-**
  - **Linear (transpose a string n places)**
    - Input: a string
    - Output: a string of the same length the bit in position zero moves n places to the left, position 1 moves to n+1, position 2 to n+2, and so on. Any bits that move beyond the origional string length are brought around to the right side of the string.
    - Example: Transpose(ABCDEF) 3 places

- **Stringlength-**
  - Input: a string
  - Output: a whole number equal to the number of characters in the string
  - Example: Stringlength(computer) = 8
    Stringlength(compsci) = 7
- **Checksum**
  - Input: a binary string
  - Output: a whole number equal to the sum of bit values or the number of ones in the string

    EX: 1101 1100 = 5
- **MSB/LSB**
  - Most significant / Least significant
    Left most Right most
  - Bit(s) (the number of bits must be stated or known)
    vs.
    Byte
- **Concatenate**
  - Input: two strings
  - Output: a string where the characters of the second are written to the right of the first in the same order as the origional
  - Example: Concatenate 1101 with and 0011 = 11010011
- **Pad**
  - Input: a binary (or hex) string of length n, where n<m
  - Output: a string of sf length m, starting w/ m-n zeros, concatenate w/ the origional string (leftfill w/ zeros until you reach the desires stringlength)

    EX: Pad (1100) to 6 bits = 00 1100
- **Split**
  - **a string into n partitions**
    - Input: a string of length m
    - Output: If m is divisible by n ($m \bmod n, R=0$), then write the first $\frac{m}{n}$ bits as an individual string, the next $\frac{m}{n}$ bits as a string and so on. If not, keep padding the string until m is divisible by n

Time Complexity or Computational Time

Can you find the sum of 1, 2, and 3? NO! not in 1 step

(1+2)+3=
↓
between 2 numbers

$\sum_{n=1}^{x} n = \frac{x(x+1)}{2}$

sigma → "the sum of"

end value
Σ
index and start value

- Constant - when output is always the same
- Measuring the rate of the time an algorithm takes to complete a process relative to the complexity of the input set.

Bigger than exponential growth is factorial

$10! = 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$

read "ten factorial," the product of the natural numbers ≤ a given number

$O(n)$ means linear complexity

$O(n^c)$ polynomial time

$O(n^2)$ Quadratic time

what to know (computational times)

FASTEST
- Constant
- Log
- Roots → particularly square root
- Linear
- Lin·log
- Polynomials → order 2 is Quadratic
- Exponential

SLOWEST Factorial

be able to order

know Big O notation of each → notation put around a function (model of how quickly the actual model goes)

adding 2 #'s vs 100 #'s

becoming more complex

double resolution = double time

x5 more numbers, x5 as long

Search Algorithms

↳ used for finding a data point in a data set

Linear search (series search)

- Examines one element at a time, one after another
  - Find maximum, minimum

O(n) → linear time

Binary search → (fastest out of ones we look at)

1. order the data set
2. Halve the data set and examine which half to keep
3. Halve the "keeper" set and examine which "new" half to keep.
4. Repeat until complete

O(log n) → Log time

Hash Table

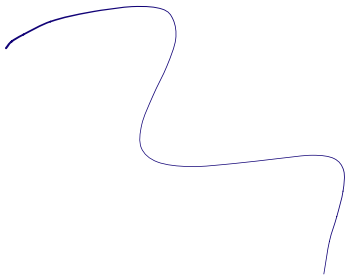
- Data is stored in an indexed list w/ a Hashing Function to associate a data point and an index. Data may then be called by index rather than dealing w/ the full data function.

O(n) → Linear Time

- very scalable
- constant average time

mod 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4								
5								
6								
7								



mod 4

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2