1 Summary: Binary and Logic

- Binary Unsigned Representation: each 1-bit is a power of two, the right-most is for $2^0$:
  \[0110101_2 = 2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 4 + 1 = 53_{10}\]

- Unsigned Range on \(n\) bits is \([0... (2^n - 1)]\). For \(n=4\) we have the range \([0=0000, 1=0001, ..., 15=1111]\)

- Binary Two’s Complement Representation: same as unsigned, except the power of two for the first bit (the sign bit) is with negative sign. When the signed bit is 0 its positive identical with unsigned transformation; when its 1 that power of two is negative:
  \[0100101_2 = +2^5 + 2^2 + 2^0 = 32 + 4 + 1 = 37_{10}\]
  \[1100101_2 = -2^6 + 2^5 + 2^2 + 2^0 = -64 + 32 + 4 + 1 = -64 + 37 = -27_{10}\]

- it is essential to know how many bits to use, to know which bit is the sign bit for negatives:
  7 bits two’s complement: \(-27 = 1100101\)
  10 bits two’s complement: \(-27 = 1111100101\)

- Two’s Complement Range on \(n\) bits is \([-2^{n-1}... + (2^n-1)]\). For \(n = 4\) we have the range \([-8=1000, -7=1001, -6=1010, ..., -1=1111, 0=0000, 1=0001, ..., 7=0111]\)

- 3-Step rule to convert from base 10 to two’s complement negatives
  - write in binary \(27 = 0000011011\)
  - flip the bits \(1111100100\)
  - add one \(1111100101\)

- addition works like in base 10 with carry bit 1+1 = write 0, carry 1; 1+1+1 = write 1 carry 1 etc

- subtraction base2: use addition of the negative value

- base 4 = $2^2$ use every two bits in base 2 to make a bit in base 4:
  \[0100101_2 = 0_2010_2 = 0.(val = 2)._{(val = 1)}.(val = 1)_4 = 211_4 = 37_{10}\]

- base 8 = $2^3$ use every three bits in base 2 to make a bit in base 8:
  \[0100101_2 = 0_20101_2 = 0.(val = 4)._{(val = 5)}._8 = 45_8 = 37_{10}\]

- base 16 = $2^4$ use every four bits in base 2 to make a bit in base 16:
  \[0100101_2 = 010_20101_2 = (val = 2)._{(val = 5)}._{16} = 25_{16} = 37_{10}\]
  \[00110111_2 = 001_21011_2 = (val = 1)._{(val = 11)}._{16} = 1B_{16} = 16 + 11_{10} = 27_{10}\]

- any base, like base=3: break the number into powers of 3, with coefficients \(\{0,1,2\}\)
  \[35_{10} = 27 + 6 + 2 = 3^3 + 0 * 3^2 + 2 * 3^1 + 2 * 3^0 = 1022_3\]
LOGIC TABLE

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A \land B (AND)</th>
<th>A \lor B (OR)</th>
<th>A \oplus B (XOR)</th>
<th>\neg(A \land B) (NAND)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

• DNF (OR between clauses for out=1); CNF (OR between negated clauses for out=0). EXAMPLE:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>output</th>
<th>DNF</th>
<th>CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\neg A \land \neg B \land C</td>
<td>A \lor \neg B \lor C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\neg A \land B \land C</td>
<td>A \lor \neg B \lor C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A \land \neg B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A \land B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A \land \neg B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A \land B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A \land B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A \land B \land \neg C</td>
<td>\neg A \lor B \lor \neg C</td>
</tr>
</tbody>
</table>

\( DNF = (\neg A \land \neg B \land C) \lor (\neg A \land B \land C) \lor (A \land \neg B \land \neg C) \lor (A \land B \land \neg C) \)

\( CNF = (A \lor B \lor C) \land (A \lor \neg B \lor C) \land (\neg A \lor B \lor \neg C) \land (\neg A \lor \neg B \lor \neg C) \)

• BOOLEAN ALGEBRA LAWS (1=true=T; 0=false=F)

<table>
<thead>
<tr>
<th>Commutative laws</th>
<th>( p \land q \equiv q \land p )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( p \lor q \equiv q \lor p )</td>
</tr>
<tr>
<td>Associative laws</td>
<td>( (p \land q) \land r \equiv p \land (q \land r) )</td>
</tr>
<tr>
<td></td>
<td>( (p \lor q) \lor r \equiv p \lor (q \lor r) )</td>
</tr>
<tr>
<td>Distributive laws</td>
<td>( p \land (q \lor r) \equiv (p \land q) \lor (p \land r) )</td>
</tr>
<tr>
<td></td>
<td>( p \lor (q \land r) \equiv (p \lor q) \land (p \lor r) )</td>
</tr>
<tr>
<td>Identity laws</td>
<td>( p \land T \equiv p )</td>
</tr>
<tr>
<td></td>
<td>( p \lor F \equiv p )</td>
</tr>
<tr>
<td>Complement laws</td>
<td>( p \land \neg p \equiv F )</td>
</tr>
<tr>
<td></td>
<td>( p \lor \neg p \equiv T )</td>
</tr>
<tr>
<td>Annihilator laws</td>
<td>( p \land F \equiv F )</td>
</tr>
<tr>
<td></td>
<td>( p \lor T \equiv T )</td>
</tr>
<tr>
<td>Idempotence laws</td>
<td>( p \land p \equiv p )</td>
</tr>
<tr>
<td></td>
<td>( p \lor p \equiv p )</td>
</tr>
<tr>
<td>Absorption laws</td>
<td>( p \land (p \lor q) \equiv p )</td>
</tr>
<tr>
<td></td>
<td>( p \lor (p \land q) \equiv p )</td>
</tr>
<tr>
<td>Double negation law</td>
<td>( \neg(\neg p) \equiv p )</td>
</tr>
<tr>
<td>De Morgan’s laws</td>
<td>( \neg(p \land q) \equiv \neg p \lor \neg q )</td>
</tr>
<tr>
<td></td>
<td>( \neg(p \lor q) \equiv \neg p \land \neg q )</td>
</tr>
</tbody>
</table>
### Logic Gates

<table>
<thead>
<tr>
<th>Name</th>
<th>NOT</th>
<th>AND</th>
<th>NAND</th>
<th>OR</th>
<th>NOR</th>
<th>XOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg. Expr.</td>
<td>$\overline{A}$</td>
<td>$AB$</td>
<td>$\overline{AB}$</td>
<td>$A \lor B$</td>
<td>$\overline{A \lor B}$</td>
<td>$A \oplus B$</td>
<td>$\overline{A \oplus B}$</td>
</tr>
<tr>
<td>Symbol</td>
<td><img src="image" alt="Symbol of NOT" /></td>
<td><img src="image" alt="Symbol of AND" /></td>
<td><img src="image" alt="Symbol of NAND" /></td>
<td><img src="image" alt="Symbol of OR" /></td>
<td><img src="image" alt="Symbol of NOR" /></td>
<td><img src="image" alt="Symbol of XOR" /></td>
<td><img src="image" alt="Symbol of XNOR" /></td>
</tr>
<tr>
<td>Truth Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>X</th>
<th>B</th>
<th>A</th>
<th>X</th>
<th>B</th>
<th>A</th>
<th>X</th>
<th>B</th>
<th>A</th>
<th>X</th>
<th>B</th>
<th>A</th>
<th>X</th>
<th>B</th>
<th>A</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
2 Summary: Mod, Number Theory, RSA

• division $a$ to $b \geq 2$: $r = a \mod b \Leftrightarrow a = bq + r$; with quotient $q$ and remainder $r \in \mathbb{Z}_b = \{0, 1, 2, ..., b-1\}$

• recap mod operations

• fast exponentiation by writing exponent $a$ as a sum of powers of 2. Example
  $3^2 = 9$; $3^4 = 9^2 = 81 = 1 \mod 10$; $3^8 = 1^2 = 1 \mod 10$; $3^{16} = 1^2 = 1 \mod 10$
  $3^{21} \mod 10 = 3^{16} \cdot 3^4 \cdot 3^1 \mod 10 = 1 \cdot 1 \cdot 3 = 3 \mod 10$

• $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \cdots \cdot p_t^{e_t}$ unique decomposition into primes

• $gcd(a, b) =$ all common (intersection) primes (each with min exponent)
  $lcm(a, b) =$ union of primes (each with max exponent)
  $ab =$ all primes together (with sum of exponents)

• $gcd(a, b) \cdot lcm(a, b) = ab$

• $a \mid b$ means ‘‘$a$ divides $b$’’ same as ‘‘$a$ is factor of $b$’’ same as ‘‘$b$ is multiple of $a$’’ same as $b = ak$ for some integer $k$

• $a, b$ have the same remainder mod $n$ if and only if $n$ divides their difference: $a \mod n = b \mod n \Leftrightarrow n \mid a - b$

• if prime $p \mid ab$; then $p \mid a \lor p \mid b$

• $a, b$ are ‘‘coprimes’’ (or relatively prime) if they have no common prime factors; then $gcd(a, b) = 1$

• if $n \mid ab$ and $a, n$ coprimes $gcd(a, n) = 1$; then $n \mid b$

• if $n \mid a$ and $m \mid a$ and $gcd(n, m) = 1$; then $nm \mid a$

• after dividing $a, b$ by their $d = gcd(a, b)$, one gets coprime numbers: $gcd(\frac{a}{d}, \frac{b}{d}) = 1$

• $a$ has multiplicative inverse $b = a^{-1} \mod n$ means $ab \mod n = 1$. That is possible if and only if $gcd(a, n) = 1$

• $a$ inverse mod $n$ (if exists) can be found as $a^{v-1}$ for integer $v$ with property $a^v = 1 \mod n$ ($v =$ order of $a$). Trying powers to obtain the order is inefficient, not practical for large $n$.

• $gcd$-bezout-coefficients $(x, y)$ for $(a, b)$ always exist to give the $gcd(a, b) = ax + by$.

• if $a, b$ coprime, $1 = gcd(a, b) = ax + by$. Then $x, y$ are the two inverses $x = a^{-1} \mod b$ and $y = b^{-1} \mod a$

• Euclid-Extended finds $x, y$ coefficients by transforming the problem $(a, b)$ into problem $(b, r)$ recursively, and then recursively-back computing the coefficients. It is efficient, even for large $a, b$.

• Linear cipher $y = ax + b \mod n$ works if and only if $(a, n)$ are coprime, that is $\exists a^{-1} \mod n$. The decoder is $x = (y - b)a^{-1} \mod n$
• Euler totient $\varphi(n)$ is the size of the set $C_n=\{\text{remainders coprime with } n\}$; in other words $\varphi(n)=\text{number of coprimes smaller than } n$.

Euler’s theorem: $a^{\varphi(n)} \equiv 1 \mod n$ for any $a \in C_n$.

• So we have four ways to find $a^{-1}$, the inverse of $a \mod n$:
  1) Brute force. Try different values $b<n$ until one works ($ab=1 \mod n$)
  2) Best in practice. $x, y = \text{EuclidExtend}(a, n)$. Then $x = a^{-1}$ is the inverse mod $n$.
  3) Find order $v$ for $a$, so $a^v = 1 \mod n$ then $a^{v-1} \mod n$ is the inverse of $a$. Can’t do fast exponentiation ($v$ unknown); still usually faster than method 1)
  4) Best if $\varphi(n)$ known. $\varphi(n)$ acts as an order for $a$ ($a^{\varphi(n)} = 1$), so the inverse is $a^{-1} = a^{\varphi(n)-1}$. Power modulo $n$ is efficient with fast exponentiation.

• For primes $p$, $\varphi(p) = p-1$ so that theorem becomes Fermats theorem $a^{p-1} \equiv 1 \mod p$ when $(a, p)$ coprimes

• Primality Test for $n$. Try for several $a<n$ to see if $a^{n-1} \mod n = 1$.
  if any of the tests(a) gives “NO”, then $n$ certainly not prime
  if all tests(a) gives “YES”, $n$ is likely prime (rare exceptions: Carmichael numbers)

• $n = pq$ (two primes) then $\varphi(n) = (p-1)(q-1)$; so if a coprime with $n$ then $a^{\varphi(n)} = a^{(p-1)(q-1)} = 1 \mod n$ or $a^{(p-1)(q-1)k+1} = a \mod n$ for any $k$

• RSA. if $n = pq$ (two large primes); $e$ and $d = e^{-1}$ are each other inverse mod $(p-1)(q-1)$ means $ed = 1 \mod (p-1)(q-1)$.
Then $a^{ed} = a^{(p-1)(q-1)k+1} = a \mod n$.
  · $n$ is known but the prime factors $p, q$ are not —and hard to find.
  · RSA public key for encryption is $e$. $\text{ENCRYPT}(a) = a^e \mod n$
  · RSA secret key for decryption is $d$. $\text{DECRYPT}(a^e) = (a^e)^d \mod n = a$
  · RSA signature: verify that one has the correct secret key, by receiving $(a, b = a^d)$ and decrypting $b$ with public key $b^e = (a^d)^e \mod n = a$

• Chinese Reminder: if $p, q$ are coprime, any pair of remainders ($a \in Z_p, b \in Z_q$) corresponds uniquely to a remainder $x \in Z_{pq}$ such that $x \mod p = a$ and $x \mod q = b$
### 3 Summary: Sets, Counting, Permutations & Combinations

- **Set builder notation**
  
  \[ A = \{ \text{positive integers smaller than 100 divisible with 7} \} = \{ x | x \in \mathbb{Z}; 0 < x < 100; 7 | x \} \]

- **count A via indexing**
  
  \[ A = \{ x | x \in \mathbb{Z}; 0 < x < 100; 7 | x \} = \{ 7 \cdot i | i \in \mathbb{Z}; 1 \leq i \leq 14 \} \Rightarrow |A| = 14 \]

- **union, intersection, set difference, set symmetric difference**
  
  \[ A \cup B = \{ x | x \in A \ OR \ x \in B \} \]
  
  \[ A \cap B = \{ x | x \in A \ AND \ x \in B \} \]
  
  \[ A - B = \{ x | x \in A \ AND \ x \notin B \} \]
  
  \[ A \Delta B = (A - B) \cup (B - A) = (A \cup B) - (A \cap B) \]

- **Cartesian Product**
  
  \[ A \times B = \{ (x, y) | x \in A; y \in B \} \]

- **Product Rule**: if any element from A can be combined with any element from B, then the number of combinations is \(|A \times B| = |A| \times |B|\)

- **power set of A** is \(P(A)\) = the set of all subsets of A, including A and \(\emptyset\); \(|P(A)| = 2^{|A|}\). Example:
  
  \(A = \{x, y, z\}\). Then \(P(A) = \{\emptyset; \{x\}; \{y\}; \{z\}; \{x, y\}; \{x, z\}; \{y, z\}; \{x, y, z\}\}\)

- **inclusion-exclusion principle**: \(|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|\)

- **Sum (Partition) Rule**: if \(\emptyset = |A \cap B| = |A \cap C| = |B \cap C|\) then \(|A \cup B \cup C| = |A| + |B| + |C|\)

- **Pigeonhole Principle**: if \(n\) items are put into \(k\) boxes, then at least one box contains at least \(\left\lceil \frac{n}{k} \right\rceil\) items

- **Permutations** \(P(n, k)\) ways to choose a sequence of \(k\) items out of \(n\). ORDER MATTERS.
  
  \[ P(n, k) = n \times (n - 1) \times (n - 1) \times \ldots \times (n - k + 1) = \frac{n!}{(n-k)!} \]

- **Combinations** \(C(n, k) = \binom{n}{k}\) ways to choose a set of \(k\) items out of \(n\). ORDER DOESNT MATTER.
  
  \[ \binom{n}{k} = n \times (n - 1) \times (n - 1) \times \ldots \times (n - k + 1) / (1 \times 2 \times \ldots \times k) = \frac{n!}{k!(n-k)!} \]

- **Binomial Theorem**
  
  \[(x + y)^n = \binom{n}{0} x^0 y^n + \binom{n}{1} x^1 y^{n-1} + \ldots \binom{n}{n} x^n y^0 = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \]
\[ x = 1, y = 1 : 2^n = (1 + 1)^n = \sum_{k=0}^{n} \binom{n}{k} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n} \]

\[ x = -1, y = 1 : 0 = (-1 + 1)^n = \sum_{k=0}^{n} \binom{n}{k}(-1)^k = \binom{n}{0} - \binom{n}{1} + \ldots + (-1)^n \]

- Pascal Formula \( \binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \)

- Pascal Triangle applies his formula at every row:

\[ \begin{array}{ccccccc}
 n & \binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \binom{n}{3} & \binom{n}{4} & \binom{n}{5} \\
 1 & 1 & 1 & & & & \\
 2 & 1 & 2 & 1 & & & \\
 3 & 1 & 3 & 3 & 1 & & \\
 4 & 1 & 4 & 6 & 4 & 1 & \\
 5 & 1 & 5 & 10 & 10 & 5 & 1 \\
\end{array} \]

- Balls in Bins: The number of ways to throw indistinguishable \( n \) balls into \( m \) distinguishable bins \( B_1, B_2, \ldots B_m \) is counted by choosing \( m - 1 \) bin-separator locations out of \( n + m - 1 \) spots for balls and separators. For example for \( n = 10, m = 6 \) the throw with bin counts 1-2-4-1-0-2 corresponds to the following choices of \( m - 1 = 5 \) separator locations among \( m + n - 1 = 15 \) balls+separators: ●|●●|●●●●|●||●●

Balls in Bins count is \( \binom{n+m-1}{m-1} \)

- circular permutation: when sitting \( n \) people at a round table the number of ways to sit them (not including rotations) is by keeping one chair-fixed and permute the rest; thus \( (n-1)! \) ways.

- counting with bijective functions: if a bijection (pairing) can be established between set \( A \) and set \( B \) then they have the same size, \( |A| = |B| \). A bijection function \( f \) has two properties
  - injectivity (different arguments produce different values): \( \forall x, y \in A; x \neq y \Rightarrow f(x) \neq f(y) \)
  - surjectivity (all \( B \) are possible function values): \( \forall v \in B, \exists x \in A, f(x) = v \)
4 Summary: Probability

• if \( \Omega \) is a set of outcome/events and \( A \subset \Omega \) then uniform probability \( \Pr[x \in A] = |A|/|\Omega| \). There are 2 options here for each outcome, either \( x \in A \) or \( x \notin A \)

• random variables \( X, Y, Z \) each partition the space \( \Omega \) by a certain criteria (for example \( X = \) object color, \( Y = \) object price, \( Z = \) object shape). They are called “random” because any object pulled at random can have any of the values of \( X \) as color, any value of \( Y \) as price, and any \( Z \) as shape. For short notation we say \( \Pr(X, Y) = \Pr(x = X, y = Y) \) is the probability that a random object has color \( x \) and price \( y \)

• Bayes Theorem \( \Pr(Y|X) \cdot P(X) = \Pr(X,Y) = \Pr(X|Y) \cdot P(Y) \) which means probability to have a particular \( (X = red, Y = 100) \) object is the probability to have \( (X = red) \) times probability to have \( (Y = 100, \text{ given that } X = red) \) or vice versa. The equality is the same as saying \( \Pr(Y|X) = \frac{\Pr(X|Y) \cdot P(Y)}{P(X)} \)

• marginalization of variable \( Y \) over variable \( X \): \( \Pr(Y) = \sum_x \Pr(x = X) \cdot \Pr(X, Y) = \sum_x \Pr(x = X) \cdot \Pr(Y|X) \).

\[
\Pr(Y) = \sum_x \Pr(x = X) \cdot \Pr(X, Y) = \sum_x \Pr(x = X) \cdot \Pr(Y|X).
\]

If \( X \) is binary with only two possible values \((X; \bar{X})\) (for example “pass” vs “fail”) then we have \( \Pr(Y) = \Pr(X) \cdot \Pr(Y|X) + \Pr(\bar{X}) \cdot \Pr(Y|\bar{X}) \)

• with marginalization of \( Y \) over binary random variable \( X \) we can write Bayes as

\[
\Pr(X|Y) = \frac{\Pr(Y|X) \cdot P(X)}{P(Y)} = \frac{\Pr(Y|X) \cdot P(X)}{\Pr(Y)}.
\]

• Independent variables \( X, Y \) means \( \Pr(Y|X) = \Pr(Y) \) which is to say \( Y \) does not depend on \( X \) (price does not depend on color). From Bayes this means also \( \Pr(X|Y) = \Pr(X) \) and \( \Pr(X, Y) = \Pr(X) \cdot \Pr(Y) \)

• expected value (or mean) for a numeric-value random variable is the average of values weighted by probabilities:

\[
\mathbb{E}[X] = \sum_x x \cdot \Pr(x = X) = \sum_x x \cdot \Pr(x)
\]

• expectation ALWAYS distributes over sum, even if variables are not independent. But they have to have numeric values

\[
\mathbb{E}[X + Y + Z] = \mathbb{E}[X] + \mathbb{E}[Y] + \mathbb{E}[Z]
\]

• variance = avg distance-to-mean\(^2\) weighted by probabilities

\[
\text{var}[X] = \sum_x (x - \text{mean})^2 \cdot \Pr(x = X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2 + \mathbb{E}^2[X] - 2X \mathbb{E}[X]] = \mathbb{E}[X^2] + 2\mathbb{E}^2[X] - 2\mathbb{E}[X]^2 = \mathbb{E}[X^2] - \mathbb{E}^2[X]
\]

• variance distributes over sum ONLY IF \( X, Y \) are independent

\( X, Y \) independent \( \Rightarrow \) \( \text{var}(X + Y) = \text{var}(X) + \text{var}(Y) \)

• entropy = randomness of \( X \) (the more random, the higher the entropy)

\[
H[X] = \sum_x \Pr(x) \log \left( \frac{1}{\Pr(x)} \right)
\]

• Markov Chains
5 Summary: Order of Growth For Positive Functions

- in order of growth we care about the general asymptotic (large \( n \)) behaviour of \( f(n) \). For example \( f(n) = 3n^2 - 2\log(n) - 5 \) grows like \( n^2 \)

- \( f = O(g) \) “BIG-O” means \( f \) less or equal with \( g \) asymptotically. Formally \( f(n) \leq C \cdot g(n) \) for \( n \geq n_0 \) and a constant \( C > 0 \). We can pick \( C \) and \( n_0 \) to anything we want but once picked they cant change

  Example: \( f(n) = n \log(n); g(n) = n^2 \Rightarrow f = O(g) \)
  Example: \( f(n) = 100n^2; g(n) = 0.0001n^2 - 200n - 1000 \Rightarrow f = O(g) \)

- \( f = \Omega(g) \) “BIG-OMEGA” means \( f \) bigger or equal with \( g \) asymptotically. Formally \( f(n) \geq D \cdot g(n) \) for \( n \geq n_0 \) and a constant \( D > 0 \). We can pick \( D \) and \( n_0 \) to anything we want but once picked they cant change

  Example: \( f(n) = n^2 \log(n); g(n) = n^2 \Rightarrow f = \Omega(g) \)
  Example: \( f(n) = n^2; g(n) = 100000000n \Rightarrow f = \Omega(g) \)

- \( f = \Theta(g) \) “BIG-THETA” means \( f \) grows equal with \( g \) asymptotically. Formally \( D \cdot g(n) \leq f(n) \leq C \cdot g(n) \) for \( n \geq n_0 \) and a constants \( C > D > 0 \). We can pick \( C > D > 0 \) and \( n_0 \) to anything we want. The inequality can be stated in reverse \( D \cdot f(n) \leq g(n) \leq C \cdot f(n) \) for different constants \( C, D \) — it is logically equivalent.

  Example: \( f(n) = n^2 - 5n + 10; g(n) = 100n^2 + 1000n + 78 \Rightarrow f = \Theta(g) \)
  Example: \( f(n) = n; g(n) = 100000000n \Rightarrow f = \Theta(g) \)
  Example: \( f(n) = \log_a(n); g(n) = \log_b(n)(a > b > 1) \Rightarrow f = \Theta(g) \)

- \( f = o(g) \) “small-o” means \( f \) strictly less than \( g \) asymptotically. Formally \( f(n) < C \cdot g(n) \) for ANY constant \( C > 0 \) and a properly chosen \( n \geq n_C \). Such inequality must be proven for any constant \( C \); equivalently it means \( \lim_{n \to \infty} f(n)/g(n) = 0 \)

  Example: \( f(n) = n \log(n); g(n) = n^2 \Rightarrow f = o(g) \)
  Example: \( f(n) = 100n^{100}; g(n) = 1.001^n \Rightarrow f = o(g) \)
  Example: \( f(n) = 2^n; g(n) = 2.0001^n \Rightarrow f = o(g) \)
  Example: \( f(n) = 100000n; g(n) = n! \Rightarrow f = o(g) \)

- \( f = \omega(g) \) “small-omega” means \( f \) strictly bigger than \( g \) asymptotically. Formally \( f(n) > D \cdot g(n) \) for ANY constant \( D > 0 \) and a properly chosen \( n \geq n_D \). Such inequality must be proven for any constant \( D \); equivalently it means \( \lim_{n \to \infty} f(n)/g(n) = \infty \)

  Example: \( f(n) = n \log(n); g(n) = n \Rightarrow f = \omega(g) \)
  Example: \( f(n) = 0.001n^2; g(n) = 100000n^{1.99} \Rightarrow f = \omega(g) \)

- in general if \( \text{asy} \) stands for asymptotic behaviour we have

\[ \log^5(n) <_{\text{asy}} n <_{\text{asy}} n \cdot \log^5(n) <_{\text{asy}} n^2 <_{\text{asy}} n^3 <_{\text{asy}} \ldots <_{\text{asy}} 1.0001^n <_{\text{asy}} 2^n <_{\text{asy}} 3^n <_{\text{asy}} \ldots <_{\text{asy}} n! <_{\text{asy}} n^n \]
6 Summary: Sequences, Series, Induction

- linear sum of indices $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

- arithmetic progression sequence $x_k = ak + b = (a + b, 2a + b, 3a + b, 4a + b, ...)$
  Characteristic: difference between values (delta) is constant $a$
  Partial sum is $\sum_{k=1}^{n} x_k = \sum_{k=1}^{n} (ak + b) = a \sum_{k=1}^{n} k + n * b = a \frac{n(n+1)}{2} + nb$

- quadratic sum of indices $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$

- quadratic progression sequence $x_k = ak^2 + bk + c = (a + b + c, 4a + 2b + c, 9a + 3b + c, 16a + 4b + c, ...)$
  Characteristic: difference between values (delta) is arithmetic progression $2ak + (a + b)$
  Partial sum is $\sum_{k=1}^{n} x_k = \sum_{k=1}^{n} (ak^2 + bk + c) = a \sum_{k=1}^{n} k^2 + b \sum_{k=1}^{n} k + n * c = a \frac{n(n+1)(2n+1)}{6} + b \frac{n(n+1)}{2} + nc$

- geometric sum of indices with base $r \neq 1$: $\sum_{k=0}^{n} r^k = \frac{r^{n+1}-1}{r-1}$
  - if base $r > 1$: $\frac{r^{n+1}-1}{r-1} = \Theta(r^n)$
  - if base $r < 1$: $\sum_{k=0}^{\infty} r^k = \lim_{n \to \infty} \frac{r^{n+1}-1}{r-1} = \frac{1}{1-r} = \Theta(1)$
  - if base $r = 1$: $\sum_{k=0}^{n} r^k = \sum_{k=0}^{n} 1 = n + 1 = \Theta(n)$

- geometric progression sequence $x_k = ar^k = (a, ar, ar^2, ar^3, ...)$
  Characteristic: ratio between values is constant $r$
  Partial sum for $r \neq 1$ is $\sum_{k=1}^{n} x_k = \sum_{k=1}^{n} ar^k = a \sum_{k=1}^{n} r^k = a \frac{r^{n+1}-1}{r-1}$

- harmonic series $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} \approx \ln(n) + 0.577 = \Theta(\log(n))$

- telescoping series $\sum_{k=1}^{n} \frac{1}{k(k+1)} = \sum_{k=1}^{n} \left(\frac{1}{k} - \frac{1}{k+1}\right) = 1 - \frac{1}{n+1}$
  - $\log(\prod_{k=1}^{n} \frac{1}{k+1}) = \sum_{k=1}^{n} \log\left(\frac{k}{k+1}\right) = \sum_{k=1}^{n} (\log(k) - \log(k+1)) = \log(1) - \log(n+1) = -\log(n+1)$

- Induction formula proofs $S_n = \sum_{k=1}^{n} (...) = f(n)$ For example $S_n = \sum_{k=1}^{n} k = f(n) = \frac{n(n+1)}{2}$
  - Base case $n = 1, 2$ : verify manually that $S_1 = f(1), S_2 = f(2)$, etc

- Induction Step (weak). Statement: $S_n = f(n) \Rightarrow S_{n+1} = f(n+1)$
  - Induction Step proof. Idea: relate $S_{n+1}$ to $S_n$
    $S_{n+1} = S_n + \delta(S_{n+1} - S_n) = f(n) + \delta(S_{n+1} - S_n) = ...(algebra)... = f(n+1)$

- Induction Step (strong). Statement: $\{S_1 = f(1); S_2 = f(2); ..., S_n = f(n)\} \Rightarrow S_{n+1} = f(n+1)$
  - Induction Step proof. Idea: relate $S_{n+1}$ to several previous $S_k (k \leq n)$
    $S_{n+1} = S_n + \delta(S_{n+1} - S_n, S_{n-1}, etc) = f(n) + \delta(S_{n+1} - S_n - S_{n-1} - ...) = ...(algebra)... = f(n+1)$
Induction logic proofs $S_n = \text{predicate}(n)$. For example $S_n = \text{“Any binary three with } n \text{ vertices has depth at least } \lfloor \log(n) \rfloor\text{”}$

- Base case $n = 1, 2, \text{etc} : \text{verify manually the predicates } S(1), S(2), \text{etc. Might need more than 2 base cases}$

- Induction Step (weak). Statement : $S_n \text{ TRUE } \Rightarrow S_{n+1} \text{ TRUE}$
  - Induction Step proof. Idea: relate $S_{n+1}$ to $S_n$
    Start with a configuration appropriate for $S_{n+1}$ (for example a tree with $n + 1$ vertices). Then reduce conveniently the setup to the previous size $n$ (for example ignore a vertex). Apply $S_n$ for this reduced problem, then using the result conveniently deduce $S_{n+1}$

- Induction Step (strong). Statement : $\{S_n, S_{n-1}, \ldots, S_{\text{base}}\} \text{ TRUE } \Rightarrow S_{n+1} \text{ TRUE}$
  - Induction Step proof. Idea: relate $S_{n+1}$ to several previous $S_k, k \leq n$
    Start with a configuration appropriate for $S_{n+1}$ (for example a tree with $n + 1$ vertices). Then reduce conveniently the setup to several prior sizes $k \leq n$ (for example remove an edge to get two smaller trees). Apply $S_k$ for these reduced problems, then using the result conveniently deduce $S_{n+1}$
7 Summary : Algorithms and Recurrences

- Linear Unordered Search : search the list/array until value found or array finished
  - best case : found on first element Θ(1)
  - average case : found about in the middle Θ(n/2) = Θ(n)
  - worst case : not found after checking all values = Θ(n)

- Linear Ordered Search : search the sorted list/array until value found or array finished
  - best case : found on first element Θ(1)
  - average case : found about in the middle, or value checked is already too high so give up Θ(n/2) = Θ(n)
  - worst case : not found after checking all values = Θ(n)

- Binary Search : search the sorted array by checking the middle value and recurse on appropriate half
  - \( T(n) = 1 + T(n/2) \): 1 for the comparison and \( T(n/2) \) for the recursion on one of the halves
  - best case : found on first element Θ(1)
  - worst case : not found after halving all the way to one element = Θ(log(n))
  - average case : found after about half of the recursive calls Θ(log(n))

- Bubble Sort: sort an array by fixing “bubbles” \( A[i], A[i + 1] \) that are in incorrect order, as long as they exist.
  - best case : no bubbles need fixing, so a pass through is necessary to verify, thus Θ(n)
  - worst case : all \( \binom{n}{2} = n(n-1)/2 \) bubbles needs fixing (array is fully backwards) = Θ(n^2)
  - average case : about half of bubbles needs fixing Θ(n^2)

- Selection Sort: find array minimum, output it, remove it, then repeat for remaining elements, etc
  - finding minimum in \( k \) elements is linear Θ(k)
  - for each \( k = n, n-1, n-2, ..., 1 \) we need linear effort, so total that is about \( n + (n-1) + ... + 1 \) = Θ(n^2)
  - \( T(n) = n + T(n-1) \); \( n \) for finding minimum, and \( T(n-1) \) for recursing to a problem of size \( n-1 \)

- Insert Sort: keep previously elements sorted and “insert” the new element in the correct spot by swapping it from the right, until all elements are sorted
  - OUTPUT\( k = [a_1, a_2, ... a_k] \) sorted
  - OUTPUT\( k+\text{next} = [a_1, a_2, ... a_k, v] \) compare \( v \) to the left and swap if necessary, until \( v \) is in the correct spot, i.e. when the element on the left is not bigger than \( v \)
  - repeat for all elements in input order
  - best case: already sorted, no swaps; Θ(n)
  - worst case: backwards sorted, all swaps; Θ(n^2)
  - average case: about half of all swaps; Θ(n^2)

- Merge Sort: split in half, recurse on both sides to sort them, then merge the sorted halves into a sorted output
  - \( T(n) = n + 2T(n/2) \); \( n \) for merging sorted halves, and twice \( T(n/2) \) for recursing to both half-sides
  - best/worst/average case: solving the recurrence gives \( T(n) = Θ(n \log(n)) \)
  - optimal sorting time based on comparisons is Θ(n log(n)), so Merge Sort is optimal in that sense.
Quick Sort: pick a random an element $v$ and put it in its correct sorted position $p$ ($v = A[p]$); rearrange the array so the element left to the pivot $p$ are smaller than $A[p]$, and the ones right to the pivot are larger than $A[p]$
- the rearrangement takes $\Theta(n)$
- recurse on both sides, which now can be sorted independently
- $T(n) = n + T(p - 1) + T(n - p)$; $n$ for rearrangement, $T(p - 1)$ for recurrence on left side; $T(n - p)$ for recurrence on right side
- best/avg case: pivot roughly in the middle; $\Theta(n \log(n))$
- worst case: pivot always at extremes; $\Theta(n^2)$

Solving recurrences of the form $T(n) = aT(n/b) + n^c$ for the order of growth: apply the recurrence iteratively few times to find a pattern for $k$ iterations, then figure out how the pattern looks for the last/biggest $k$

$$T(n) = a^kT\left(\frac{n}{b^k}\right) + n^c$$

$$= a[aT\left(\frac{n}{b}\right) + \left(\frac{n}{b}\right)^c] + n^c = a^2T\left(\frac{n}{b^2}\right) + a\left(\frac{n}{b}\right)^c + n^c$$

$$= a^2[aT\left(\frac{n}{b^2}\right) + \left(\frac{n}{b^2}\right)^c] + a\left(\frac{n}{b}\right)^c + n^c = a^3T\left(\frac{n}{b^3}\right) + a^2\left(\frac{n}{b^2}\right)^c + a\left(\frac{n}{b}\right)^c + n^c$$

$$= a^3[aT\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^3}\right)^c] + a^2\left(\frac{n}{b^2}\right)^c + a\left(\frac{n}{b}\right)^c + n^c = a^4T\left(\frac{n}{b^4}\right) + a^3\left(\frac{n}{b^3}\right)^c + a^2\left(\frac{n}{b^2}\right)^c + a\left(\frac{n}{b}\right)^c + n^c$$

(general pattern after $k$ iterations)

$$= a^kT\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i\left(\frac{n}{b^i}\right)^c$$

(last $k = \log_b(n)$)

$$= a^{\log_b(n)}T(1) + \sum_{i=0}^{\log_b(n)-1} a^i\left(\frac{n}{b}\right)^c$$

$$= n^{\log_b(a)}T(1) + n^c \sum_{i=0}^{\log_b(n)-1} \left(\frac{a}{b}\right)^i$$

- if $a = b^c (c = \log_b(a))$ that becomes $\Theta(n^c + n^c \log_b(n)) = \Theta(n^c \log(n))$

- if $r = a/b^c \neq 1$, applying geometric series formula for base $r$, that becomes $\Theta(n^{\log_b(a)} + n^c \cdot \frac{r^{\log_b(n)} - 1}{r - 1})$

- if $r = a/b^c < 1 \Leftrightarrow a < b^c \Leftrightarrow c > \log_b(a)$, it becomes

$$\Theta(n^{\log_b(a)} + n^c \cdot \frac{1}{1-r}) = \Theta(n^{\log_b(a)} + n^c) = \Theta(n^c)$$

- if $r = a/b^c > 1 \Leftrightarrow a > b^c \Leftrightarrow c < \log_b(a)$ it becomes

$$\Theta(n^{\log_b(a)} + n^c \cdot r^{\log_b(n)}) = \Theta(n^{\log_b(a)} + n^c \cdot n^{\log_b(r)}) = \Theta(n^{\log_b(a)} + n^c \cdot n^{\log_b(a) - \log_b(b^c)}) = \Theta(n^{\log_b(a)} + n^{c + \log_b(a) - c}) = \Theta(n^{\log_b(a)})$$

- Fibonacci base $F_0 = 0; F_1 = 1$ and recurrence $F_{n+1} = F_n + F_{n-1}$, satisfies $F_n = \Theta(\phi^n)$ where $\phi \approx 1.618$ is the positive root of quadratic equation $\phi^2 = \phi + 1$
8 Summary: Graphs

- Graph $G = (V,E)$ has set of vertices $V$ and set of edges $E$; each edge $e = (u,v)$ is a pair of vertices. Graphs are undirected if edges are undirected $(u,v) = (v,u)$ (for all edges) or directed if $((u,v) \neq (v,u))$. Graphs can be represented by adjacency matrix where $A_{uv} = 1$ indicates edge $(u,v)$ exists, or by linked lists per vertex.

- **degree**($u$) = number of edges incident n vertex $u$. Sum of all degrees is twice the number of edges, because each edge $(u,v)$ contributes +1 to both degrees of $u$ and $v$

- **path**($u,v$) = a sequence of edges starting from $u$ ending in $v$. If graph is directed, edges have to be considered in their direction (see picture left for path 1-4-2-5)

- **cycle** is a path that begins and ends at the same vertex $u$ (see picture right for cycle 2-5-4). An undirected cycle that passes through $u$ and $v$ can be thought as two different paths between $u$ and $v$

- **connected component**: a subgraph where there is a path between any two nodes. Graphs can have several connected components, which of course are not connected between them (left picture for undirected components; right for directed components)

- **tree** = connected graph with no cycles. A tree with $V$ vertices must have exactly $|V| - 1$ edges: less would disconnect it; more would form cycles.

- **Spanning tree** = a tree that includes all graph vertices. There can be multiple spanning trees (see two below drawn with red edges)
• BFS= Breadth-First-Search traversal: start at a root-vertex (wave 0), look for immediate neighbors (wave 1); look at their new neighbors (wave 2) etc. Edges used to discover neighbors (blue in the picture below) are called the BFS-edges and form the BFS-tree.

• BFS: the shortest path from \( u \) to \( v \) (by number of edges) can be found running BFS from root \( u \) and check the wave number that finds \( v \). If \( v \) not found, it means that there is no path from \( u \) to \( v \), or \( v \) is unreachable from \( u \).

• DFS= Depth-First-Search traversal: go as deep as possible on the current branch, before coming back to follow other branches. An immediate neighbor to the root is discovered only after the entire subtree of the current neighbor is explored (see picture). Edges used for DFS advance are marked in red and form the DFS tree; the picture below contains two DFS trees because the first one starting in A could not explore the whole graph. Each vertex is marked with a discovery time and a finishing time.