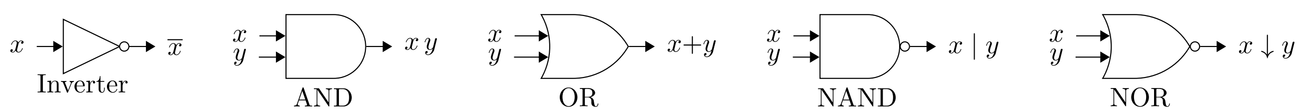
Crib Sheets

This section of the *Guide* provides a "crib sheet" for each of the 13 chapters of the text. Each sheet includes some of the more difficult-to-remember definitions, notation, and formulas from the chapter.

You should check with your instructor to find out whether these will be allowed on the exams in your class. Even if your instructor does not allow their use during the exam, together with the Key Terms and Results at the end of each chapter, you may find these pages useful as you prepare for exams.

Crib Sheet for Chapter 1

Logical and: $p \wedge q$ is true when both p and q are true, is false when at least one of p and q is false.
Logical or (inclusive): $p \vee q$ is true when at least one of p and q is true, is false when both p and q are false.
Exclusive or: $p \oplus q$ is true when exactly one of p and q is true, is false otherwise.
Conditional statement (implication): $p \to q \equiv$ "If p, then q" $\equiv$ "p only if q" $\equiv$ "p is a sufficient condition for q" $\equiv$ "q is a necessary condition for p." $p \to q$ is false when p is true and q is false, is true otherwise. $\neg(p \to q) \equiv p \wedge (\neg q)$. $p \to q$ is equivalent to its **contrapositive** $\neg q \to \neg p$, but not to its **converse** $q \to p$ or its **inverse** $\neg p \to \neg q$.
Biconditional statement: $p \leftrightarrow q$, means $(p \to q) \wedge (q \to p)$, usually read "if and only if" and sometimes written "iff" in English.
De Morgan's laws: $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$; $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$.
The basic logical operations can be represented by **gates**:



They can be combined to make **combinational circuits** to represent any logical expression.
Quantifiers: $\forall x(P(x) \to Q(x)) \equiv$ "for all x, if $P(x)$ then $Q(x)$"; $\exists x(P(x) \wedge Q(x)) \equiv$ "there exists an x such that $P(x)$ and $Q(x)$." Here $P(x)$ and $Q(x)$ are **propositional functions**, and there is always a **domain** or **universe of discourse**, either implicit or explicitly stated, over which the variable ranges.
Negations of quantified propositions: $\neg\forall x P(x) \equiv \exists x \neg P(x)$; $\neg\exists x P(x) \equiv \forall x \neg P(x)$.
Theorem: a proposition that can be proved; **lemma:** a simple theorem used to prove other theorems; **proof:** a demonstration that a proposition is true; **corollary:** a proposition that can be proved as a consequence of a theorem that has just been proved.
A **valid** argument—one using correct rules of inference based on tautologies—will always give correct conclusions if the hypotheses used are correct. Invalid arguments, relying on **fallacies**, such as affirming the conclusion, denying the hypothesis, begging the question, or circular reasoning, can lead to false conclusions.
Some rules of inference: $[p \wedge (p \to q)] \to q$ (**modus ponens**); $[\neg q \wedge (p \to q)] \to \neg p$ (**modus tollens**); $[(p \to q) \wedge (q \to r)] \to (p \to r)$ (**hypothetical syllogism**); $[(p \vee q) \wedge (\neg p)] \to q$ (**disjunctive syllogism**); $\{P(a) \wedge \forall x\,[P(x) \to Q(x)]\} \to Q(a)$ (**universal modus ponens**); $\{\neg Q(a) \wedge \forall x\,[P(x) \to Q(x)]\} \to \neg P(a)$ (**universal modus tollens**); $(\forall x\, P(x)) \to P(c)$ (**universal instantiation**); ($P(c)$ for an arbitrary c) $\to \forall x\, P(x)$ (**universal generalization**); $(\exists x\, P(x)) \to$ ($P(c)$ for some element c) (**existential instantiation**); ($P(c)$ for some element c) $\to \exists x\, P(x)$ (**existential generalization**).
Trivial proof: a proof of $p \to q$ that just shows that q is true without using the hypothesis p.
Vacuous proof: a proof of $p \to q$ that just shows that the hypothesis p is false.
Direct proof: a proof of $p \to q$ that shows that the assumption of the hypothesis p implies the conclusion q.
Proof by contraposition: a proof of $p \to q$ that shows that the assumption of the negation of the conclusion q implies the negation of the hypothesis p (i.e., proof of contrapositive).
Proof by contradiction: a proof of p that shows that the assumption of the negation of p leads to a contradiction.
Proof by cases: a proof of $(p_1 \vee p_2 \vee \cdots \vee p_n) \to q$ that shows that each conditional statement $p_i \to q$ is true.
Statements of the form $p \leftrightarrow q$ require that both $p \to q$ and $q \to p$ be proved. It is sometimes necessary to give two separate proofs (usually a direct proof or a proof by contraposition); other times a string of equivalences can be constructed starting with p and ending with q: $p \leftrightarrow p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_n \leftrightarrow q$.
To give a **constructive proof** of $\exists x P(x)$ is to show how to find an element x that makes $P(x)$ true. **Nonconstructive existence proofs** are also possible, often using proof by contradiction.
One can **disprove** a universally quantified proposition $\forall x P(x)$ simply by giving a **counterexample**, i.e., an object x such that $P(x)$ is false. One cannot *prove* it with an example, however.
Fermat's last theorem: There are no positive integer solutions of $x^n + y^n = z^n$ if $n > 2$.
An integer is **even** if it can be written as $2k$ for some integer k; an integer is **odd** if it can be written as $2k+1$ for some integer k; every number is even or odd but not both. A number is **rational** if it can be written as p/q, with p an integer and q a nonzero integer.

Crib Sheet for Chapter 2

Empty set: the set with no elements, $\{\ \}$, is denoted $\emptyset$; this is not the same as $\{\emptyset\}$, which has one element.
Subset: $A \subseteq B \equiv \forall x(x \in A \rightarrow x \in B)$; **proper subset:** $A \subset B \equiv (A \subseteq B) \wedge (A \neq B)$ (i.e., B has at least one element not in A).
Equality of sets: $A = B \equiv (A \subseteq B \wedge B \subseteq A) \equiv \forall x(x \in A \leftrightarrow x \in B)$.
Power set: $\mathcal{P}(A) = \{ B \mid B \subseteq A \}$ (the set of all subsets of A). A set with n elements has 2^n subsets.
Cardinality: $|S|$ = number of elements in S.
Some specific sets: $\mathbf{R}$ is the set of real numbers, all of which can be represented by finite or infinite decimals; $\mathbf{N} = \{0, 1, 2, 3, \ldots\}$ (natural numbers); $\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (integers); $\mathbf{Z}^+ = \{1, 2, \ldots\}$ (positive integers); $\mathbf{Q} = \{ p/q \mid p, q \in \mathbf{Z} \wedge q \neq 0 \}$ (rational numbers); $\mathbf{Q}^+ = \{ p/q \mid p, q \in \mathbf{Z}^+ \}$ (positive rational numbers).
Set operations: $A \times B = \{ (a, b) \mid a \in A \wedge b \in B \}$ (**Cartesian product**); $\overline{A}$ = the set of elements in the universe not in A (**complement**); $A \cap B = \{ x \mid x \in A \wedge x \in B \}$ (**intersection**); $A \cup B = \{ x \mid x \in A \vee x \in B \}$ (**union**); $A - B = A \cap \overline{B}$ (**difference**); $A \oplus B = (A - B) \cup (B - A)$ (**symmetric difference**).
Inclusion–exclusion (simple case): $|A \cup B| = |A| + |B| - |A \cap B|$.
De Morgan's laws for sets: $\overline{A \cap B} = \overline{A} \cup \overline{B}$; $\overline{A \cup B} = \overline{A} \cap \overline{B}$.
A **function** f from A (the **domain**) to B (the **codomain**) is an assignment of a unique element of B to each element of A. Write $f : A \rightarrow B$. Write $f(a) = b$ if b is assigned to a. **Range** of f is $\{ f(a) \mid a \in A \}$; f is **onto (surjective)** $\equiv$ range$(f) = B$; f is **one-to-one (injective)** $\equiv \forall a_1 \forall a_2 [f(a_1) = f(a_2) \rightarrow a_1 = a_2]$.
If f is one-to-one and onto (**bijective**), then the **inverse** function $f^{-1} : B \rightarrow A$ is defined by $f^{-1}(y) = x \equiv f(x) = y$.
If $f : B \rightarrow C$ and $g : A \rightarrow B$, then the **composition** $f \circ g$ is the function from A to C defined by $f \circ g(x) = f(g(x))$.
Rounding functions: $\lfloor x \rfloor$ = the largest integer less than or equal to x (**floor function**); $\lceil x \rceil$ = the smallest integer greater than or equal to x (**ceiling function**).

Summation notation: $\displaystyle\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n$.

Sum of first n positive integers: $\displaystyle\sum_{j=1}^{n} j = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$.

Sum of squares of first n positive integers: $\displaystyle\sum_{j=1}^{n} j^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$.

Sum of geometric progression: $\displaystyle\sum_{j=0}^{n} ar^j = a + ar + ar^2 + \cdots + ar^n = \frac{ar^{n+1} - a}{r - 1}$ if $r \neq 1$.

Two sets have the **same cardinality** if there is a bijection between them. We say that $|A| \leq |B|$ if there is a one-to-one function from A to B.
A set is **countable** if it is finite or there is a bijection from the positive integers to the set—in other words, if the elements of the set can be listed a_1, a_2, ... Sets of the latter type are called **countably infinite**, and their cardinality is denoted $\aleph_0$. The empty set, the integers, and the rational numbers are countable; **the set of real numbers and the power set of the set of natural numbers are uncountable**. The union of a countable number of countable sets is countable.
The **Schröder-Bernstein theorem** states that if $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$. In other words, if there is a one-to-one function from A to B and there is a one-to-one function from B to A, then there is a one-to-one and onto function from A to B.
Matrix multiplication: The $(i, j)^{\text{th}}$ entry of $\boldsymbol{AB}$ is $\sum_{t=1}^{k} a_{it} b_{tj}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, where $\boldsymbol{A}$ is an $m \times k$ matrix and $\boldsymbol{B}$ is a $k \times n$ matrix. **Identity matrix $\boldsymbol{I}_n$** with 1's on main diagonal and 0's elsewhere is the multiplicative identity.
Cardinality arguments can be used to show that some functions are **uncomputable**.
Matrix addition $(+)$, Boolean meet $(\wedge)$ and join $(\vee)$ are done entry-wise; Boolean matrix product $(\odot)$ is like matrix multiplication using Boolean operations.
Transpose: $\boldsymbol{A}^t$ is the matrix whose $(i, j)^{\text{th}}$ entry is a_{ji} (the $(j, i)^{\text{th}}$ entry of $\boldsymbol{A}$); $\boldsymbol{A}$ is **symmetric** if $\boldsymbol{A}^t = \boldsymbol{A}$.

Crib Sheet for Chapter 3

An **algorithm** is a finite sequence of precise instructions for performing a computation or solving a problem.

Algorithms can be expressed in **pseudocode**.

Most algorithms have the following **properties:** having input, having output, definiteness, correctness, finiteness, effectiveness, generality.

Algorithms that make what seems to be the "best" choice at each step are called **greedy algorithms**. Sometimes they work; sometimes they don't. For example, the greedy change-making algorithm works for American coins, but does not work for some other combinations of denominations.

There are important algorithmic paradigms besides greedy, including **brute force** (examine all possible solutions in order to determine the best solution) and some that will be studied in later chapters (**dynamic programming**, **probabilistic algorithms**, and **divide-and-conquer**).

The **halting problem** is **unsolvable**: There is no algorithm to test whether a given computer program with a given input will ever halt.

Big-*O* notation: "$f(x)$ is $O(g(x))$" means $\exists C\, \exists k\, \forall x(x > k \rightarrow |f(x)| \leq C|g(x)|)$. Big-$O$ of a sum is largest (fastest growing) of the functions in the sum; big-O of a product is the product of the big-O's of the factors. If f is $O(g)$, then g is $\Omega(f)$ ("big-Omega"). If f is both big-O and big-Omega of g, then f is $\Theta(g)$ ("big-Theta").

Little-*o* notation; This was introduced in the exercise set. We say that $f(x)$ is $o(g(x))$ if $\lim_{x\rightarrow\infty} f(x)/g(x) = 0$.

Powers grow faster than logs: $(\log n)^c$ is $O(x^d)$ but not the other way around, where c and d are positive numbers.

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1+f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$ and $(f_1f_2)(x)$ is $O(g_1(x)g_2(x))$.

$\log n!$ is $O(n \log n)$.

Binary search has **time complexity** $O(\log n)$, whereas **linear search** has (**worst case** and **average case**) time complexity $O(n)$; both have **space complexity** $O(1)$ (not counting input). Bubble sort and insertion sort have $O(n^2)$ worst case time complexity.

Matrix multiplication by the standard algorithm has time complexity $O(m_1m_2m_3)$ if the matrices have dimensions $m_1 \times m_2$ and $m_2 \times m_3$. More efficient algorithms can reduce the complexity of multiplying two $n \times n$ matrices from $O(n^3)$ to $O(n^{\sqrt{7}})$.

Important complexity classes include **polynomial** (n^b), **exponential** (b^n for $b > 1$), and **factorial** ($n!$).

A problem that can be solved by an algorithm with polynomial worst-case time-complexity is called **tractable**; otherwise they are called **intractable**.

The class **P** is the class of tractable problems. The class **NP** consists of problems for which it is possible to *check solutions* (as opposed to finding solutions) in polynomial time. Clearly $\mathbf{P} \subseteq \mathbf{NP}$. The **P versus NP problem** asks whether in fact $\mathbf{P} = \mathbf{NP}$; no one knows the answer.

Crib Sheet for Chapter 4

Divisibility: $a \mid b$ means $a \neq 0 \wedge \exists c(ac = b)$ (a is a **divisor** or **factor** of b; b is a **multiple** of a).

Base b representations: $(a_{n-1}a_{n-2}\ldots a_2a_1a_0)_b = a_{n-1}b^{n-1} + \cdots + a_2b^2 + a_1b + a_0$. To convert from base 10 to base b, continually divide by b and record remainders as a_0, a_1, a_2, ... ($b = 8$ is **octal**; $b = 16$ is **hexadecimal**, using A through F for digits 10 through 15). Convert from binary to octal by grouping bits by threes, from the right, to hexadecimal by grouping by fours.

Addition of two **binary numerals** each of n bits ($(a_{n-1}a_{n-2}\ldots a_2a_1a_0)_2$) requires $O(n)$ bit operations. **Multiplication** requires $O(n^2)$ bit operations if done naively, $O(n^{1.585})$ steps by more sophisticated algorithms.

Division "algorithm": $\forall a \forall d > 0 \exists q \exists r(a = dq + r \wedge 0 \le r < d)$; q is the quotient and r is the remainder; we write $a \textbf{ mod } d$ for the remainder. Example: $-18 = 5 \cdot (-4) + 2$, so $-18 \textbf{ mod } 5 = 2$.

Congruent modulo m: $a \equiv b \pmod m$ iff $m \mid a - b$ iff $a \textbf{ mod } m = b \textbf{ mod } m$. One can do arithmetic in $\mathbf{Z}_m = \{0, 1, \ldots, m-1\}$ by working modulo m. There are fast algorithms for computing $b^n \textbf{ mod } m$, based on successive squaring.

Integer $n > 1$ is **prime** iff its only factors are 1 and itself $(2, 3, 5, 7, \ldots)$; otherwise it is **composite** $(4, 6, 8, 9, \ldots)$. There are infinitely many primes, but it is not known whether there are infinitely many twin primes (primes that differ by 2) or whether every even positive integer greater than 2 is the sum of two primes (**Goldbach's conjecture**) or whether there are infinitely many **Mersenne primes** (primes of the form $2^p - 1$).

Naive **test for primeness** (and method for finding **prime factorization**): To find prime factorization of n, successively divide it by all primes less than $\sqrt{n}$ $(2, 3, 5, \ldots)$; if none is found, then n is prime. If a prime factor p is found, then continue the process to find the prime factorization of the remaining factor, namely n/p; this time the trial divisions can start with p. Continue until a prime factor remains. The **prime number theorem** states that there are approximately $n/\ln n$ primes less than or equal to n.

Fundamental theorem of arithmetic: Every integer greater than 1 can be written as a product of one or more primes, and the product is unique except for the order of the factors. (Proof based on fact that if a prime divides a product of integers, then it divides at least one of those integers.)

Euclidean algorithm for greatest common divisor: $\gcd(x, y) = \gcd(y, x \textbf{ mod } y)$ if $y \neq 0$; $\gcd(x, 0) = x$. Using extended Euclidean algorithm or working backwards, one can find **Bézout coefficients** and write $\gcd(a, b) = sa + tb$.

Two integers are **relatively prime** if their greatest common divisor (gcd) is 1. The integers a_1, a_2, ..., a_n are **pairwise relatively prime** iff $\gcd(a_i, a_j) = 1$ whenever $1 \le i < j \le n$.

Chinese remainder theorem: If m_1, m_2, ..., m_n are pairwise relatively prime, then the system $\forall i(x \equiv a_i \pmod{m_i})$ has unique solution modulo $m_1m_2\cdots m_n$. Application: handling very large integers on a computer.

Fermat's little theorem: $a^{p-1} \equiv 1 \pmod p$ if p is prime and does not divide a. The converse is not true; for example $2^{340} \equiv 1 \pmod{341}$, so 341 $(=11 \cdot 31)$ is a **pseudoprime**.

If a and b are positive integers, then there exist integers s and t such that $as+bt = \gcd(a, b)$ (**linear combination**). This theorem allows one to compute the **multiplicative inverse** $\overline{a}$ of a modulo b (i.e., $\overline{a}a \equiv 1 \pmod b$) as long as a and b are relatively prime, which enables one to solve **linear congruences** $ax \equiv c \pmod b$.

A **primitive root** modulo a prime p is an integer r in $\mathbf{Z}_p$ such that every nonzero element of $\mathbf{Z}_p$ is a power of r.

Discrete logarithms: $\log_r a = e$ modulo p if $r^e \textbf{ mod } p = a$ and $1 \le e \le p - 1$.

A common **hashing function:** $h(k) = k \textbf{ mod } m$, where k is the key.

Check digits, for error-correcting codes like UPCs, involve modular arithmetic.

Pseudorandom numbers: can be generated by the **linear congruential method:** $x_{n+1} = (ax_n + c) \textbf{ mod } m$, where x_0 is arbitrarily chosen **seed**. Then $\{x_n/m\}$ will be rather randomly distributed numbers between 0 and 1.

Shift cipher: $f(p) = (p + k) \textbf{ mod } 26$ [A $\leftrightarrow$ 0, B $\leftrightarrow$ 1, ...]. Julius Caesar used $k = 3$. **Affine cipher** uses $f(p) = (ap + b) \textbf{ mod } 26$ with $\gcd(a, 26) = 1$.

RSA public key encryption system: An integer M representing the plaintext is translated into an integer C representing the ciphertext using the function $C = M^e \textbf{ mod } b$, where n is a public number that is the product of two large (maybe 100-digit or so) primes, and e is a public number relatively prime to $(p-1)(q-1)$; the primes p and q are kept secret. Decryption is accomplished via $M = C^d \textbf{ mod } n$, where d is an inverse of e modulo $(p-1)(q-1)$. It is infeasible to compute d without knowing p and q, which are infeasible to compute from n. Similar methods can be used for **key exchange protocols** and **digital signatures**.

Crib Sheet for Chapter 5

The well-ordering property: Every nonempty set of nonnegative integers has a least element.

Principle of mathematical induction: Let $P(n)$ be a propositional function in which the domain (universe of discourse) is the set of positive integers. Then if one can show that $P(1)$ is true (**basis step** or **base case**) and that for every positive integer k the conditional statement $P(k) \to P(k+1)$ is true (**inductive step**), then one has proved $\forall n P(n)$. The hypothesis $P(k)$ in a proof of the inductive step is called the **inductive hypothesis**. More generally, the induction can start at any integer, and there can be several base cases.

Strong induction: Let $P(n)$ be a propositional function in which the domain (universe of discourse) is the set of positive integers. Then if one can show that $P(1)$ is true (**basis step** or **base case**) and that for every positive integer k the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \to P(k+1)$ is true (**inductive step**), then one has proved $\forall n P(n)$. The hypothesis $\forall j \le k\, P(j)$ in a proof of the inductive step is called the (strong) **inductive hypothesis**. Again, the induction can start at any integer, and there can be several base cases.

Sometimes **inductive loading** is needed, where we must prove by mathematical induction or strong induction something stronger than the desired statement so as to have a powerful enough inductive hypothesis (this concept was introduced in the exercises).

Inductive or **recursive** definition of a function f with the set of nonnegative integers as its domain: specification of $f(0)$, together with, for each $n > 0$, a rule for finding $f(n)$ from values of $f(k)$ for $k < n$. Example: $0! = 1$ and $(n+1)! = (n+1) \cdot n!$ (**factorial function**).

Inductive or **recursive** definition of a set S: a rule specifying one or more particular elements of S, together with a rule for obtaining more elements of S from those already in it. It is understood that S consists precisely of those elements that can be obtained by applying these two rules.

Structural induction can be used to prove facts about recursively defined objects.

Fibonacci numbers: $f_0,\ f_1,\ f_2,\ \ldots$: $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for all $n \ge 2$.

Lamé's theorem: The number of divisions used by the Euclidean algorithm to find $\gcd(a, b)$ is $O(\log b)$.

An algorithm is **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input. It is **iterative** if it is based on the repeated use of operations in a loop.

There is an efficient recursive algorithm for computing **modular powers** (b^n **mod** m), based on computing $b^{\lfloor n/2 \rfloor}$ **mod** m.

Merge sort is an efficient recursive algorithm for sorting a list: break the list into two parts, recursively sort each half, and merge them together in order. It has $O(n \log n)$ time complexity in all cases.

A program segment S is **partially correct** with respect to **initial assertion** p and **final assertion** q, written $p\{S\}q$, if whenever p is true for the input values of S and S terminates, q is true for the output values of S.

A **loop invariant** for **while** *condition* S is an assertion p that remains true each time S is executed in the loop; i.e., $(p \wedge \textit{condition})\{S\}p$. If p is true before the program segment is executed, then p and $\neg\textit{condition}$ are true after it terminates (if it does). In symbols, $p\{\textbf{while } \textit{condition } S\}(\neg\textit{condition} \wedge p)$.

Crib Sheet for Chapter 6

Sum rule: Given t mutually exclusive tasks, if task i can be done in n_i ways, then the number of ways to do exactly one of the tasks is $n_1 + n_2 + \cdots + n_t$.
Size of union of disjoint sets: $|A_1 \cup A_2 \cup \cdots \cup A_n| = |A_1| + |A_2| + \cdots + |A_n|$.
Two-set case of inclusion–exclusion: $|A \cup B| = |A| + |B| - |A \cap B|$.
Product rule: If a task consists of successively performing t tasks, and if task i can be done in n_i ways (after previous tasks have been completed), then the number of ways to do the task is $n_1 \cdot n_2 \cdots n_t$.
A set with n elements has 2^n subsets (equivalently, there are 2^n bit strings of length n).
Tree diagrams can be used to organize counting problems.
Pigeonhole principle: If more than k objects are placed in k boxes, then some box will have more than 1 object.
Generalized version: If N objects are placed in k boxes, then some box will have at least $\lceil N/k \rceil$ objects.
Ramsey number: $R(m,n)$ is the smallest number of people there must be at a party so that there exist either m mutual friends or n mutual enemies (assuming each pair of people are either friends or enemies). $R(3,3) = 6$.
r-permutation of set with n objects: *ordered* arrangement of r of the objects from the set (no repetitions allowed); there are $P(n,r) = n!/(n-r)!$ such permutations.
r-combination of set with n objects: *unordered* selection (i.e., subset) of r of the objects from the set (no repetitions allowed); there are $C(n,r) = n!/[r!(n-r)!]$ such combinations. Alternative notation is $\binom{n}{r}$, called **binomial coefficient**.
Pascal's identity: $C(n,k-1) + C(n,k) = C(n+1,k)$ if $n \ge k \ge 1$; allows construction of **Pascal's triangle** of binomial coefficients, using $C(n,0) = C(n,n) = 1$ along the sides.
Combinatorial identities often have combinatorial proofs: $C(n,r) = C(n,n-r)$; $(a+b)^n = \sum_{k=0}^{n} C(n,k)a^{n-k}b^k$ (**binomial theorem**), with corollary $\sum_{k=0}^{n} C(n,k) = 2^n$; $C(m+n,r) = \sum_{k=0}^{r} C(m,r-k)C(n,k)$ (**Vandermonde's identity**).
Number of r-**permutations** of an n-set **with repetitions allowed** is n^r; number of r-**combinations** of an n-set **with repetitions allowed** is $C(n+r-1,r)$. This latter value is the same as the **number of solutions in nonnegative integers** to $x_1 + x_2 + \cdots + x_n = r$.
Permutations with indistinguishable objects: Number of n-permutations of an n-set with n_i indistinguishable objects of type t for $1 \le t \le k$ is $n!/(n_1!n_2! \cdots n_k!)$. This also gives the number of ways to distribute n distinguishable objects into k distinguishable boxes so that box t gets n_t objects.
For distributing distinguishable object into distinguishable boxes, use product rule (or the formula $n!/(n_1!n_2! \cdots n_k!)$ if the number in each box is specified). For distributing indistinguishable object into distinguishable boxes, use formula for the number of combinations with repetitions allowed. For distributing distinguishable object into indistinguishable boxes, there is no good closed formula; **Stirling numbers of the second kind** are involved. Distributing indistinguishable object into indistinguishable boxes involves **partitions of positive integers**, and there is no good closed formula.
There are good **algorithms** for finding the lexicographically "next" permutation or combination and thereby **for generating all permutations or combinations**.

Crib Sheet for Chapter 7

If all outcomes are equally likely in a **sample space** S with n outcomes, then the **probability of an event** E is $p(E) = |E|/n$; more generally, if $p(s_i)$ is probability of i^{th} outcome s_i, then $p(E) = \sum_{s_i \in E} p(s_i)$.

Probability distributions satisfy these conditions: $0 \le p(s) \le 1$ for each $s \in S$, and $\sum_{s \in S} p(s) = 1$.

For **complementary** event, $p(\overline{E}) = 1 - p(E)$; for **union** of two events (either one or both happen), $p(E \cup F) = p(E) + p(F) - p(E \cap F)$; for **independent events**, $p(E \cap F) = p(E)p(F)$.

The **conditional probability** of E given F (probability that E will happen after it is known that F happened) is $p(E|F) = p(E \cap F)/p(F)$.

Bernoulli trials: If only two outcomes are **success** and **failure**, with $p(\text{success}) = p$ and $p(\text{failure}) = q = 1 - p$, then the **binomial distribution** applies, with probability of exactly k success in n trials being $b(k; n, p) = C(n,k)p^k q^{n-k}$.

Bayes' theorem: If E and F are events such that $p(E) \ne 0$ and $p(F) \ne 0$, then

$$p(F \mid E) = \frac{p(E \mid F)p(F)}{p(E \mid F)p(F) + p(E \mid \overline{F})p(\overline{F})}.$$

A **random variable** assigns a number to each outcome. **Expected value** (expectation) of random variable X is $E(X) = \sum_{i=1}^{n} p(s_i)X(s_i)$; alternatively, $E(X) = \sum_{r \in X(S)} p(X = r)r$; expected number of successes for n Bernoulli trials is pn.

Variance of random variable X is $V(X) = \sum_{i=1}^{n} p(s_i)(X(s_i) - E(X))^2$; variance can also be computed as $V(X) = E(X^2) - E(X)^2$; square root of variance is **standard deviation** $\sigma(X)$; variance of number of successes for n Bernoulli trials is npq.

X_1 and X_2 are **independent** if $p(X_1 = r_1 \text{ and } X_2 = r_2) = p(X_1 = r_1) \cdot p(X_2 = r_2)$. In this case $E(XY) = E(X)E(Y)$.

Expectation is linear even when the variables are not independent. This means that the expectation of a sum is sum of expectations, and $E(aX + b) = aE(X) + b$.

Variance of a sum is the sum of the variances ($V(X_1 + X_2) = V(X_1) + V(X_2)$) *when the variables are independent.*

The random variable X that gives the number of flips needed before a coin lands tails, when the probability of tails is p and the flips are independent, has the **geometric distribution**: $p(X = k) = (1-p)^{k-1}p$ for $k = 1, 2, \ldots$; $E(X) = 1/p$.

Chebyshev's inequality: $p(|X(s) - E(X)| \ge r) \le V(X)/r^2$.

A **probabilistic algorithm** is an algorithm that might give the incorrect answer but only with small probability. For example, there are good probabilistic tests as to whether or not a natural number is prime.

The **probabilistic method** is a proof technique that shows the existence of an object with a given property by showing that there is a nonzero probability of choosing such an object if choices are made at random. For example, there is a probabilistic proof that the Ramsey number $R(k,k)$ is at least $2^{k/2}$.

Crib Sheet for Chapter 8

A **recurrence relation** for a sequence a_0, a_1, a_2, ..., is a formula expressing each a_n in terms of previous terms (for all $n > n_0$); **initial conditions** specify a_0 through a_{n_0}; a **solution** to such a system is an explicit formula for a_n in terms of n that satisfies the recurrence relation and initial conditions.

The **Fibonacci numbers** are defined recursively by $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$; continues $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, $f_6 = 8$, $f_7 = 13$, $f_8 = 21$, ...; explicit formula is that f_n equals nearest integer to $((1+\sqrt{5})/2)^n/\sqrt{5}$.

A recurrence relation is **linear** of degree k if it is of the form $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + f(n)$; if all the c_i's are constants, then it has **constant coefficients**; if $f(n)$ is identically 0, then it is **homogeneous**. Such a recurrence relation and k initial conditions completely determine the sequence.

Recurrence relations of degree 1 can often be solved by **iteration**. Given a_n expressed in terms of a_{n-1}, rewrite a_{n-1} in this equation using the same recurrence relation with $n-1$ in place of n. This expresses a_n in terms of a_{n-2}. Then rewrite a_{n-2} in terms of a_{n-3}, again using the recurrence relation. Continue in this way, noting the pattern that evolves, until finally you have a_n written explicitly in terms of a_1 (or a_0), probably as a series. This gives the explicit solution (preferably with the series summed in closed form).

To **solve** linear homogeneous recurrence relation with constant coefficients: (1) write down the **characteristic equation** $r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_k = 0$ and find all its roots, with multiplicities; (2) each distinct root (**characteristic root**) r gives rise to a solution $a_n = r^n$; if a root is repeated, occurring s times, then there are solutions $a_n = n^i r^n$ for $i = 0, 1, \ldots, s-1$; (3) take arbitrary linear combination of all solutions so obtained, with coefficients α_1, α_2, ..., α_k; (4) plug in the k initial conditions to solve for the α's.

To **solve** linear nonhomogeneous recurrence relation with constant coefficients: (1) solve the **associated homogeneous recurrence relation** (with the $f(n)$ term omitted) to obtain a general solution $a_n^{(h)}$ with some yet-to-be-calculated constants α_i; (2) obtain a **particular solution** $a_n^{(p)}$ of the nonhomogeneous recurrence relation using the method of undetermined coefficients (the form to use depends on $f(n)$ and on the solution of the associated homogeneous recurrence relation); (3) write down the general solution: $a_n = a_n^{(h)} + a_n^{(p)}$; (4) plug in the k initial conditions to solve for the α's.

Divide-and-conquer relation: $f(n) = af(n/b) + g(n)$. If f is an increasing function satisfying this relation whenever n is a power of b, where $g(n) = cn^d$, then $f(n) = O(n^d)$ if $a < b^d$, $f(n) = O(n^d \log n)$ if $a = b^d$, and $f(n) = O(n^{\log_b a})$ if $a > b^d$ (**master theorem**).

Divide-and-conquer algorithms work by dividing a problem into simpler non-overlapping subproblems; **dynamic programming algorithms** work by dividing a problem into simpler overlapping subproblems. Matrix multiplication can be done in $O(n^{\log 7}) \approx O(n^{2.8})$ steps, rather than the naive $O(n^3)$, using a divide-and-conquer algorithm. Talks in a lecture hall can be scheduled using dynamic programming to maximize total attendance.

Generating functions are expressions of the form $f(x) = \sum_{k=0}^{\infty} a_k x^k$, associated with an infinite sequence $\{a_k\}$. They can be used to solve recurrence relations, prove combinatorial identities, and solve counting problems. To model a combinatorial situation (such as counting how many ways there are to distribute cookies), let a_k be the quantity of interest (the answer when there are k cookies); choices are modeled by adding (corresponding to "or" situations—the person can get 1, 2, or 3 cookies) or multiplying (corresponding to "and" situations—Tom, Dick, and Jane must each receive cookies) polynomials in x. The combinatorics is then replaced by the algebra of multiplying out the polynomials or obtaining a **closed form** expression. The most important generating function in applications is $1/(1-ax)^n = \sum_{k=0}^{\infty} C(n+k-1,k) a_k x^k$. **Partial fractions** must sometimes be used when solving recurrence relations using generating functions.

Inclusion–exclusion principle: (for $n=3$) $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$; (general case) $|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|$. Can be applied to counting number of objects among N objects with none of a collection of properties: $N(P_1' P_2' \ldots P_n') = N - \sum_i N(P_i) + \sum_{i<j} N(P_i P_j) - \sum_{i<j<k} N(P_i P_j P_k) + \cdots + (-1)^n N(P_1 P_2 \ldots P_n)$. Specific application: counting the number of primes up to n found by using **sieve of Eratosthenes**.

Number of onto functions from an m-set to an n-set is $n^m - C(n,1)(n-1)^m + C(n,2)(n-2)^m - \cdots + (-1)^{n-1} C(n, n-1) 1^m$.

A **derangement** is a permutation leaving no object in its original position; there are $D_n = n![1 - 1/1! + 1/2! - 1/3! + \cdots + (-1)^n/n!]$ derangements of n objects; as $n \to \infty$, $D_n/n!$ quickly approaches $1/e \approx 0.368$.

Crib Sheet for Chapter 9

(Binary) relation R from A to B: subset of $A \times B$. Write aRb for $(a,b) \in R$; relation **on** A is relation from A to A; graph of a function from A to B is a relation such that $\forall a \in A \ \exists$ exactly one pair (a,b) in the relation. Relation R on a set A is **reflexive** if aRa for all $a \in A$; **irreflexive** if aRa for no $a \in A$; **symmetric** if aRb implies bRa for all $a,b \in A$; **asymmetric** if aRb implies that b is not related to a, for all $a,b \in A$; **antisymmetric** if $aRb \wedge bRa$ implies $a = b$ for all $a,b \in A$; **transitive** if $aRb \wedge bRc$ implies aRc for all $a,b,c \in A$.

Inverse relation to R is given by $bR^{-1}a$ if and only if aRb.

If R is a relation from A to B, and S is a relation from B to C, then the **composite** is the relation $S \circ R$ from A to C in which a is related to c if and only if there exists a $b \in B$ such that aRb and bSc. $R^n = R \circ R \circ \cdots \circ R$.

n-ary relation on domains A_1, A_2, ..., A_n is a subset of $A_1 \times A_2 \times \cdots \times A_n$; data bases using the relational data model are just sets of n-ary relations in which each n-tuple is called a **record** made up of **fields** (n is the **degree**); a domain is a **primary key** if the corresponding field uniquely determines the record.

The **projection** $P_{i_1,i_2,\ldots,i_m}$ maps an n-tuple to the m-tuple formed by deleting all fields not in the list $i_1, i_2, \ldots, i_m$.

The **join** J_p takes a relation R of degree m and a relation S of degree n and produces a relation of degree $m+n-p$ by finding all tuples $(a_1, a_2, \ldots, a_{m+n-p})$ such that $(a_1, a_2, \ldots, a_m) \in R \wedge (a_{m-p+1}, a_{m-p+2}, \ldots, a_{m-p+n}) \in S$.

A relation R on $A = \{a_1, a_2, \ldots, a_n\}$ can be represented by an $n \times n$ matrix $\boldsymbol{M}_R$ whose $(i,j)^{\text{th}}$ entry is 1 if $a_i R a_j$ and is 0 otherwise. Reflexivity, symmetry, antisymmetry, transitivity can easily be read off the matrix. Boolean products of the matrices give the matrix for the composite ($\boldsymbol{M}_{S \circ R} = \boldsymbol{M}_R \odot \boldsymbol{M}_S$).

Digraph for R on A: a vertex for each element of A and an arrow (**arc, edge**) from a to b whenever aRb (**loop** at a when aRa). Reflexivity, symmetry, antisymmetry, transitivity can easily be read off the digraph.

Closure of relation R on set A with respect to property P: smallest relation on A containing R and having property P; **reflexive closure**—add all pairs (a,a) if not already in R ($R \cup \Delta_A$, where $\Delta_A = \{\,(a,a) \mid a \in A\,\}$); **symmetric closure**—add pair (b,a) whenever (a,b) is in R, if (b,a) is not already in R ($R \cup R^{-1}$).

Transitive closure of R equals R^* (**connectivity relation** for R), defined as $\bigcup_{n=1}^{\infty} R^n$; can be computed efficiently at the matrix level using **Warshall's algorithm**, and visually at the digraph level by considering **paths**.

A relation R on set A is an **equivalence relation** if R is reflexive, symmetric, and transitive; usually equivalence relations can be recognized by their definition's being of the form "two elements are related if and only if they have the same [something]." For each $a \in A$ the set of elements in A related to (i.e., **equivalent to**) a is the **equivalence class** of a, denoted $[a]$. (Canonical example: congruence modulo m.) The set of equivalence classes partitions A into pairwise disjoint nonempty sets; conversely, every partition of A induces an equivalence relation by declaring two elements to be related if they are in the same set of the partition.

A relation R on set A is a **partial order** if R is reflexive, antisymmetric, and transitive; **total** or **linear** order if in addition every pair of elements are **comparable** (either aRb or bRa). (A,R) is called a **partially ordered set** or **poset**, and R is denoted $\preceq$ ($\prec$ means $\preceq$ but not equal); canonical example is $\subseteq$ on $P(S)$.

Partial orders on A_i induce **lexicographic order** on $A_1 \times A_2 \times \cdots \times A_n$ given by $(a_1, a_2, \ldots, a_n) \prec (b_1, b_2, \ldots, b_n)$ if for some $i \geq 0$, $a_1 = b_1$, ..., $a_i = b_i$, and $a_{i+1} \prec b_{i+1}$. A partial order on A induces **lexicographic order** on strings A^* given by $a_1 a_2 \ldots a_n \prec b_1 b_2 \ldots b_m$ if $(a_1, a_2, \ldots, a_t) \prec (b_1, b_2, \ldots, b_t)$ where $t = \min(m,n)$ or if $(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n)$ and $n < m$.

Hasse diagram represents poset on finite set by placing x above y whenever $y \prec x$, and also in this case drawing line from x to y if there is no z such that $y \prec z \prec x$; then $a \prec b$ if and only if there is an upward path from a to b in the Hasse diagram.

m in a poset is **maximal** if there is no x with $m \prec x$ (**minimal** in the dual situation); m is a **greatest element** if $x \preceq m$ for all x (dually for **least element**). An **upper bound** for a subset B is an element u such that $b \preceq u$ for all $b \in B$ (dually for **lower bound**), and u is a **least** upper bound if it is an upper bound such that $u \preceq v$ for every upper bound v (dually for **greatest** lower bound). A poset is a **lattice** if l.u.b.'s and g.l.b.'s always exist.

Every finite poset has a minimal element that can be found by moving down the Hasse diagram. Iterating this observation gives an algorithm for obtaining a total order **compatible** with the partial order (**topological sorting**)—keep peeling off a minimal element.

Crib Sheet for Chapter 10

A **simple graph** $G = (V, E)$ consists of nonempty set of **vertices** (singular: **vertex**) and set of unordered pairs of distinct vertices called **edges**. A **multigraph** allows more than one edge joining same pair of vertices (**multiple** or **parallel edges**)—E is just a set with an endpoint function f taking each edge e to its two *distinct* endpoints. A **pseudograph** is like a multigraph but endpoints $f(e)$ need not be distinct, allowing for **loops**. A **directed graph** (**digraph**) is just like a simple graph except that edges are directed (each e is an *ordered pair*, and loops are allowed). **Directed multigraph** is just like multigraph (parallel edges allowed) except that edges are directed (loops allowed). Given directed graph, we can ignore order and look at **underlying undirected graph**.

Graphs can be used to model relationships: of many kinds, such as acquaintances, food webs, telephone calls, road systems, the Internet, tournaments, organizational structure.

Vertices joined by an edge are **adjacent** and the edge is **incident** to them. **Degree of a vertex** ($\deg(v)$) is the number of incident edges, with loops counted double; **isolated vertex** has degree 0, **pendant vertex** has degree 1; **regular graph** has all degrees equal. In digraph $\deg^-(v)$ is number of edges leading into v (**in-degree**; v is **terminal** vertex) and $\deg^+(v)$ is number of edges leading out of v (**out-degree**; v is **initial** vertex).

Handshaking theorem: Undirected: $2e = \sum_{v\in V} \deg(v)$; corollary—number of vertices of odd degree is even. Directed: $e = \sum_{v\in V} \deg^-(v) = \sum_{v\in V} \deg^+(v)$.

Bipartite graph: vertex set can be partitioned into two nonempty sets with no edges joining vertices in same set. The **complete** graph K_n has n vertices and an edge joining every pair ($n(n-1)/2$ edges in all); **complete bipartite** graph $K_{m,n}$ has $m+n$ vertices in parts of sizes m and n and an edge joining every pair of vertices in different parts (mn edges in all). The **cycle** C_n has n vertices and n edges, joined in a circle; the **wheel** W_n is C_n with one more vertex joined to these n vertices. The **cube** Q_n has all n-bit binary strings for vertices, with an edge between every pair of vertices differing in only one bit position.

(V, E) is **subgraph** of (W, F) if $V \subseteq W$ and $E \subseteq F$; **union** of two simple graphs is formed by taking union of corresponding vertex sets and corresponding edge sets.

Graph can be represented by **adjacency matrix** (m in position (i, j) denotes m parallel edges from i to j), **adjacency lists**, **incidence matrix** (1 is position (i, j) says that vertex i is incident to edge j).

Graphs are **isomorphic** if there is a bijection between vertex sets that preserves all adjacencies and nonadjacencies; to show graphs are *not* isomorphic, find **invariant** on which they differ (e.g., degree sequence, existence of cycles).

A **path** of length n from u to v is a sequence of n edges leading successively from u to v; is a **circuit** if $n > 0$ and $u = v$, is **simple** if no edge occurs more than once. A graph is **connected** if every pair of vertices is joined by a path; digraph is **strongly connected** if every pair of vertices is joined by a path in each direction, **weakly connected** if underlying undirected graph is connected. **Components** are maximal connected subgraphs. Removal of **cut edge** (**bridge**) or **cut vertex** (**articulation point**) creates more components.

Vertex connectivity of a graph G: $\kappa(G) =$ size of a smallest **vertex cut** (set of edges whose removal disconnects G); G is **k-connected** if $\kappa(G) \ge k$; **edge connectivity**: $\lambda(G) =$ size of a smallest **edge cut**.

$(i, j)^{\text{th}}$ entry of $\boldsymbol{A}^r$, where $\boldsymbol{A}$ is adjacency matrix, counts numbers of paths of length r from i to j.

An **Euler circuit [path]** is simple circuit [path] containing all edges. Connected graph has an Euler circuit [path] if and only if the vertex degrees are all even [the vertex degrees are all even except for at most two vertices]. Splicing algorithm or Fleury's algorithm finds them efficiently.

A **Hamilton path** is path containing all vertices exactly once; Hamilton path together with edge back to starting vertex is **Hamilton circuit**. No good necessary and sufficient conditions for existence of these, or algorithms for finding them, are known. Q_n has Hamilton circuit for all $n \ge 2$ (**Gray code**).

Weighted graphs: have **lengths** assigned to edges; one can find shortest path from u to v (minimum sum of weights of edges in the path) using **Dijkstra's algorithm**.

A **planar graph** is a graph having a **planar representation** (drawing in plane without edges crossing); **Kuratowski's theorem:** graph is planar if and only if it has no subgraph **homeomorphic** to (formed by performing **elementary subdivisions** on edges of) K_5 or $K_{3,3}$.

Euler's formula: Given planar representation with v vertices, e edges, c components, r regions, $v-e+r = c+1$; corollary: in planar graph with at least 3 vertices, $e \le 3v - 6$.

A graph is **colored** by assigning colors to vertices with adjacent vertices getting distinct colors; minimum number of colors required is **chromatic number**. Every planar graph can be colored with four colors. There are also applications of **edge colorings** (adjacent edges must get different colors).

Crib Sheet for Chapter 11

A **tree** is a connected undirected graph with no simple circuits; characterized by having unique simple path between every pair of vertices, and by being connected and satisfying $e = v - 1$. A **forest** is an undirected graph with no simple circuits—each component is a tree, and $e = v -$ number of components. Every tree has at least two vertices of degree 1.

A **rooted tree** is a tree with one vertex specified as root; can be viewed as directed graph away from root. If uv is a directed edge, then u is **parent** and v is **child**; **ancestor, descendant, sibling** defined genealogically. Vertices without children are **leaves**; others are **internal**. Draw trees with root at the top, so that vertices occur at **levels**, with root at level 0; **height** is maximum level number. A tree is **balanced** if all leaves occur only at bottom or next-to-bottom level, **complete** if only at bottom level. The **subtree rooted at** a is the tree involving a and all its descendants.

An ***m*-ary tree** is a rooted tree in which every vertex has at most m children (**binary** tree when $m = 2$); a **full** m-ary tree has exactly m children at each internal vertex. Full m-ary tree with i internal vertices and l leaves has $n = i + l = mi + 1$ vertices. An m-ary tree with height h satisfies $l \le m^h$, so $h \ge \lceil \log_m l \rceil$ (equality in latter inequality if tree is balanced).

An **ordered rooted tree** has an order among the children of each vertex, drawn left-to-right; in ordered binary tree, each child is a **right child** or **left child**, and subtree rooted at right [left] child is called **right [left] subtree**.

A **binary search tree (BST)** is binary tree with a key at each vertex so that at each vertex, all keys in left subtree are less and all keys in right subtree are greater than key at the vertex; $O(\log n)$ algorithm for insertion and search in BST.

Decision trees provide lower bounds on number of questions an algorithm needs to ask to accomplish its task for all inputs (e.g., coin-weighing, searching).

Binary trees can be used to encode **prefix codes**, binary codes for symbols so that no code word is the beginning of another code word. **Huffman codes** are efficient prefix codes for data compression.

Game trees can be used to find optimal strategies for two-person games, using the **minmax** principle. Value of a leaf is payoff to first player. Value of an internal vertex at an even level (square) is maximum of values of its children; value of an internal vertex at an odd level (circle) is minimum of values of its children.

Universal address system: root is labeled 0; children of root are labeled $1, 2, \ldots$; children of vertex labeled x are labeled $x.1, x.2, \ldots$. Addresses are ordered using preorder traversal.

Preorder visits root, then subtrees (recursively) in preorder; **postorder** visits subtrees (recursively) in postorder, then root; **inorder** visits first subtree (recursively) in inorder, then root, then remaining subtrees (recursively) in inorder.

The **expression tree** for a calculation has constants at leaves, operations at internal vertices (evaluated by applying operation to values of its children).

Prefix (Polish) notation corresponds to preorder traversal of expression tree (operator precedes operands); **postfix (reverse Polish) notation** corresponds to postorder traversal of expression tree (operands precede operator); both of these allow unambiguous expressions without using parentheses. **Infix** form is normal notation but requires full parenthesization (corresponds to inorder traversal of expression tree).

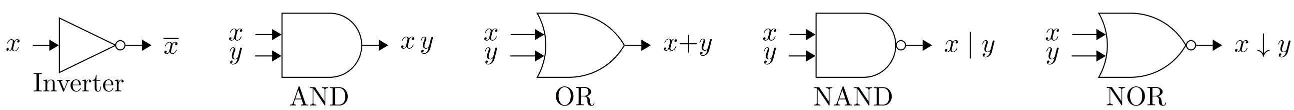
Naive sorting routines like **bubble sort** require $O(n^2)$ steps in worst case; the best that can be done with comparison-based sorting is $O(n \log n)$ (a huge improvement), using something like **merge sort**.

A **spanning tree** is a tree containing all vertices of a connected given graph; can be found by **depth-first** search (recursively search the unvisited neighbors) or **breadth-first** search (fan out). Edges not in depth-first search spanning tree (**back edges**) join vertices to ancestors or descendants. Edges not in breadth-first search spanning tree (**cross edges**) join vertices at same level or level differing by one. Depth-first search can be modified to implement **backtracking** algorithms for exhaustive consideration of all cases of a problem (like graph coloring).

Minimum spanning trees can be found in weighted graphs using greedy **Prim** or **Kruskal** algorithms (choose least costly edge at each stage that doesn't get you into trouble).

Crib Sheet for Chapter 12

Boolean operations: sum, product, and complementation defined on $\{0,1\}$ by $0+0=0$, $0+1=1+0=1+1=1$, $1\cdot 1=1$, $0\cdot 1=1\cdot 0=0\cdot 0=0$ (also write product using concatenation, without the dot), $\overline{0}=1$, $\overline{1}=0$; also *XOR* defined by $1\oplus 1=0\oplus 0=0$ and $1\oplus 0=0\oplus 1=1$, *NAND* defined by $1\mid 1=0$ and $1\mid 0=0\mid 1=0\mid 0=1$, *NOR* defined by $0\downarrow 0=1$ and $1\downarrow 0=0\downarrow 1=1\downarrow 1=0$. A **Boolean algebra** is an abstraction of this, with operations $\vee$, $\wedge$, and $\overline{}$, which also applies to other situations (e.g., sets).
Boolean operations obey same identities (commutative, associative, idempotent, distributive, De Morgan, etc.) with $\cup$ replaced by $+$, $\cap$ replaced by $\cdot$, U replaced by 1, and $\emptyset$ replaced by 0.
Boolean variables are variables taking on only values 0 and 1; **Boolean expression** is expression made up from Boolean constants (0 and 1), variables and operations combined in usual ways with parentheses where desired to override the natural precedence that products are evaluated before sums.
Dual of Boolean expression: interchange 0 and 1, $+$ and $\cdot$; dual of an identity is an identity.
Boolean functions are functions from n-tuples of variables to $\{0,1\}$. They can be represented using Boolean expressions, and in particular, in **disjunctive normal form** as **sums of products** or in **conjunctive normal form** as **products of sums**. In sum of products form, each product is a **minterm** $y_1y_2\cdots y_n$, where each y_i is a **literal**, either x_i or $\overline{x}_i$. Two expressions are called **equivalent** if they compute the same function.
Set of operators is **functionally complete** if every Boolean function can be represented using them. Examples are $\{+,\cdot,\overline{}\}$, $\{+,\overline{}\}$, $\{\cdot,\overline{}\}$, $\{\mid\}$, and $\{\downarrow\}$.
The standard Boolean operations can be represented by **gates**:



They can be combined to make **combinational circuits** to represent any Boolean function, such as **(full) adders** (take two bits and a carry and produce a sum bit and a carry) or **half-adders** (same, without the carry as input).
Minimization of circuits: given Boolean function in sum of products form, find an expression as simple as possible to represent it (i.e., use as few literals and operations as possible, meaning using few gates to produce the circuit). Geometric method (**Karnaugh maps** or **K-maps**) and tabular method (**Quine-McCluskey procedure**) organize this task efficiently for small n. Typical K-maps with blocks circled:

	y	$\overline{y}$
x	1	
$\overline{x}$	1	

	$y\,z$	$y\,\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}\,z$
x	1		1	1
$\overline{x}$	1		1	

	$y\,z$	$y\,\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}\,z$
$w\,x$	1	1	1	1
$w\,\overline{x}$	1			1
$\overline{w}\,\overline{x}$	1	1		
$\overline{w}\,x$	1			

Typical Quine–McCluskey calculation:

			Step 1		Step 2	
	Term	String	Term	String	Term	String
1	$w\,x\,y\,z$	1111	$(1,2)\,w\,x\,y$	$111-\ \Leftarrow$	$(3,4,5,8)\,\overline{y}\,z$	$--01$
2	$w\,x\,y\,\overline{z}$	1110	$(1,3)\,w\,x\,z$	$11-1\ \Leftarrow$		
3	$w\,x\,\overline{y}\,z$	1101	$(3,4)\,w\,\overline{y}\,z$	$1-01$		
4	$w\,\overline{x}\,\overline{y}\,z$	1001	$(3,5)\,x\,\overline{y}\,z$	-101		
5	$\overline{w}\,x\,\overline{y}\,z$	0101	$(4,6)\,w\,\overline{x}\,\overline{y}$	$100-\ \Leftarrow$		
6	$w\,\overline{x}\,\overline{y}\,\overline{z}$	1000	$(4,8)\,\overline{x}\,\overline{y}\,z$	-001		
7	$\overline{w}\,\overline{x}\,y\,\overline{z}$	0010 $\Leftarrow$	$(5,8)\,\overline{w}\,\overline{y}\,z$	$0-01$		
8	$\overline{w}\,\overline{x}\,\overline{y}\,z$	0001				

	1	2	3	4	5	6	7	8
$\overline{y}\,z$			X	X	X			X
$w\,x\,y$	X	X						
$w\,x\,z$	X		X					
$w\,\overline{x}\,\overline{y}$				X		X		
$\overline{w}\,\overline{x}\,y\,\overline{z}$							X	

$\overline{y}\,z + w\,x\,y + w\,\overline{x}\,\overline{y} + \overline{w}\,\overline{x}\,y\,\overline{z}$ covers all minterms

Crib Sheet for Chapter 13

A **vocabulary** or **alphabet** V is just a finite nonempty set of symbols; strings of symbols from V, including the empty string λ, are **words**; the set of *all* words is denoted V^*; a **language** is any subset of V^*.

If L_1 and L_2 are languages, then so are the **union** $L_1 \cup L_2$, **intersection** $L_1 \cap L_2$, **concatenation** $L_1L_2 = \{\, uv \mid u \in L_1 \wedge v \in L_2 \,\}$ ($L^2 = LL$, etc.), **Kleene closure** $L^* = \{\lambda\} \cup L \cup L^2 \cup L^3 \cdots$, **complement** $\overline{L} = V^* - L$.

Phrase-structure grammar: $G = (V, T, S, P)$, where V is a vocabulary, $T \subseteq V$ is the set of **terminal** symbols (the **nonterminal symbols** are $N = V - T$), $S \in V$ is the **start symbol**, P is a set of **productions**, which are rules of the form $w_1 \to w_2$, where $w_1, w_2 \in V^*$. (The convention is to use capital letters for the nonterminals.)

Derivations: If a string u can be transformed to a string v by applying some production (i.e., replacing a substring w_1 in u by w_2 where $w_1 \to w_2$ is a production), then write $u \Rightarrow v$ and say v is **directly derivable** from u; if $u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n$, then write $u_1 \overset{*}{\Rightarrow} u_n$ and say u_n is **derivable** from u_1.

The **language generated by** G is the set of strings *of terminal symbols* derivable from S (the start symbol).

Types of grammars: type 0—no restrictions; **type 1 (context-sensitive)**—productions are all of the form $lAr \to lwr$, where A is a nonterminal symbol, l and r are strings of zero or more terminal or nonterminal symbols, and w is a nonempty string of terminal or nonterminal symbols, or of the form $S \to \lambda$ as long as S does not appear on the right-hand side of any other production; **type 2 (context-free)**—left side of each production must be single nonterminal symbol; **type 3 (regular)**—only allowed productions are $A \to bB$, $A \to b$, and $S \to \lambda$, where A, B, and S are nonterminals, b is terminal, and S is start symbol; each type is included in previous type. Productions in type 2 grammars can also be represented in **Backus–Naur form**; nonterminals have angled brackets surrounding them; a vertical bar means "or"; arrows are replaced by ::=; e.g., $\langle A\rangle ::= \langle A\rangle c\langle A\rangle\langle B\rangle \mid 3 \mid \langle B\rangle\langle C\rangle$.

Derivation (or **parse**) **tree:** shows transformation of start symbol into string of terminals (the leaves, read from left to right), invoking a production at each internal vertex.

Palindrome: string that reads the same forward as backward, i.e., $w = w^R$ (string whose first half is the reverse of its last half, either of the form uu^R or uxu^R for x a symbol). Set of palindromes is context-free but not regular.

Finite state machine with output on the transitions (Mealy machine): $M = (S, I, O, f, g, s_0)$, where S is set of **states**, I and O are **input and output alphabets**, $s_0 \in S$ is **start state**, $f : S \times I \to S$ and $g : S \times I \to O$ are **transition and output functions**; can be represented by **state table** or **state diagram** with i, o labeling edge (s, t) if $f(s, i) = t$ and $g(s, i) = o$. "Moves" between states as it reads an input string, producing an output string of same length; can be used for language recognition (output 1 if input read so far is in language, 0 if not).

Finite state machine with output on the states (Moore machines): same as Mealy machine except $g : S \to O$ assigns an output to every state rather than to every transition; output is one symbol longer than input.

Finite state machine with no output (deterministic automaton): $M = (S, I, f, s_0, F)$—same as Mealy machine except that there is no output function but rather a set of **final states** $F \subseteq S$. A string w is **recognized** or **accepted** by M if M ends up in a final state on input w; the **language recognized** or **accepted** by M, written $L(M)$, is the set of all strings accepted by M. (Plural of "automaton" is "automata.")

Nondeterministic automaton: same as deterministic one except that the transition function f sends a state and input symbol to a *set of states*; think of the machine as choosing which state to go into next from among the possibilities provided by f. A string is accepted if some sequence of choices leads to a final state at the end of the input; $L(M)$ defined as before. For automata of either type, we say that M_1 and M_2 are equivalent if $L(M_1) = L(M_2)$. **Theorem:** Given a nondeterministic finite automaton, there is an equivalent deterministic one.

Regular expressions over a set I are built up from symbols for the elements of I, a symbol for the empty set, and a symbol for the empty string by the operations of concatenation, union, and Kleene closure. The expressions represent the corresponding sets of strings. **Regular sets (languages)** are sets represented by regular expressions. Regular languages are **closed under** intersection, union, concatenation, Kleene closure, complement; **context-free languages are closed under** union, concatenation, Kleene closure.

Theorem: A set is regular iff it is generated by some regular grammar iff it is accepted by some finite automaton.

Pumping lemma: If z is a string in $L(M)$ of length longer than the number of states in M, then we can write $z = uvw$, with $v \neq \lambda$ so that $uv^iw \in L(M)$ for all i. This allows us to prove, for example, that $\{\, 0^n1^n \mid n = 1, 2, 3, \ldots \,\}$ is not regular.

A **Turing machine** is specified by a set of 5-tuples (s, x, s', x', d): if in state s scanning symbol x on tape, it writes x' on tape, enters state s', and moves tape head R or L according to d. TM's can recognize all computable (Type 0) languages, can compute all computable functions (Church–Turing thesis).