Algorithm: precise, unambiguous, step by step procedure for carrying out some calculation or more generally for solving some problem

Algorithmics: study of algorithms (design & analysis)

Algorithm Properties: Input, Output, Precision, Determinism, Finiteness, Correctness, Generality

Algorithm Creation Process: understand, design, analyse (possibly back to design), implement

Analysis: correctness, termination, simplicity, generality, time, space

Logarithms: $b^e = x$ iff $\log_b x = e$; $\log xy = \log x + \log y$, $x, y > 0$; $\log \frac{x}{y} = \log x - \log y$, $x, y > 0$; $\log_b x^y = y \log_b x$; $\log_a x = \frac{\log_b x}{\log_b a}$, $a > 0, a \neq 1$; $\log_b x > \log_b y, b > 1, x > y > 0$

Series: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$; $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$; $\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2 = \Theta(n^4)$; $\sum_{i=1}^{n} i^k \approx \frac{n^{k+1}}{k+1} = \Theta(n^{k+1})$; $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} = \Theta(a^n)$; $\sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2 = \Theta(n2^n)$; $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n + 0.57 = \Theta(\lg n)$; $\sum_{i=1}^{n} \lg i = \Theta(n \lg n)$; $\sum_{i=1}^{n} i \lg i = \Theta(n^2 \lg n)$

Theorem: mathematical statement that has been proved true

Lemma: 'small' theorem, usually used in proof of a more important mathematical statement

Corollary: mathematical statement which easily follows from a theorem

Proof: logical argument that a mathematical statement is true

Proof by Construction: mathematical statement about the existence of an object can be proved by constructing the object

Proof by Contradiction: assume that a mathematical statement is false and show that the assumption leads to a contradiction

Polynomial Degree: highest power

Intervals: closed ($[a, b] = x | a \leq x \leq b$), open ($(a, b) = x | a < x < b$), half-open (either side)

Subsequence: consists of only certain terms in the same order as the full sequence

Substring: assume string index start from 1, then for $t[i, j]$ if $i < j$ then substring is from i to j inclusive, if $i = j$ then substring is only i, else then empty string

Boolean Expression: containing boolean variables, operators, parentheses

Normal Forms: conjunctive (clause linked with $\wedge$, inside has $\vee$), disjunctive (opposite)

Upper Bound: u such that $x \leq u$ for all $x \in X$, X: all reals

Lower Bound: l such that $x \geq l$ for all $x \in X$

Supremum: least upper bound

Infimum: greatest lower bound

Graph: consists of set of vertices and edges, edge is unordered (unless directed) pair of vertices, simple if without loops or multiple edges

Degree: number of edges incident on the vertex

Path: alternating sequence of vertices and edges, starting and ending with vertices, simple has no repeated vertices

Diameter: maximum distance between any two vertices

Cycle: path starting and ending at the same vertex with actual length, simple if without repeated vertices

Hamiltonian Cycle: cycle that contains each vertex exactly once

Euler Cycle: cycle with no repeated edges that contains all edges and vertices, exists iff connected and degree of every vertex is even

Complement: of simple graph, denoted as $\bar{G}$, same vertices, edge in $\bar{G}$ iff not in G

Tree: connected and acyclic; connected and has $n - 1$ edges, acyclic and has $n - 1$ edges, level of vertex is simple path length from root, height is max length

Homogeneous Recurrence: characteristic equation ($a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = 0$) linear, homogeneous (combination of $t_i = 0$), constant coefficients, guess $t_n = x^n$, unknown x, so $a_0 x^n + a_1 x^{n-1} + \cdots + a_k x^{n-k} = 0$, factor out x^{n-k}, so $p(x) = a_0 x^k + a_1 x^{k-1} + \cdots + a^k x^0 = 0$, so the general solution is $\sum_{i=1}^{k} c_i r_i^n$, where r is the roots of the equation (if distinct)

HR Example: $T(n) = 2T(n-1), T(1) = 1, T(n) - 2T(n-1) = 0, T(n) = x^n, T(n-1) = x^{n-1}$, solve $x^n - 2x^{n-1} = 0, x = 2, T(n) = c_1 2^n, T(1) = 1 = c_1 2^1, c_1 = 0.5$

HR with Non Distinct Roots: for each non distinct root include it but multiply with n each time ($c_1 r^n + c_2 n r^n + c_3 n^2 r^n + \cdots$)

Asymptotic Upper Bound: $f(n = O(g(n)))$ if there exist $C_1 > 0$ and N_1 such that $f(n) \leq C_1 g(n), n \geq N_1$

Asymptotic Lower Bound: $f(n) = \Omega(g(n))$ if there exist $C_2 > 0$ and N_2 such that $f(n) \geq C_2 g(n), n \geq N_2$

Master Theorem: $T(n) = aT(n/b) + f(n), f(n) = \Theta(n^k)$, $T(n) = \Theta(n^k)$ if $a < b^k$, $T(n) = \Theta(n^k \log n)$ if $a = b^k$, $T(n) = \Theta(n^{\log_b a})$ if $a > b^k$

Smoothness: if the function is non decreasing from N inclusive to infinity, and for every integer $b \geq 2, f(bn) = O(f(n))$

Formal Smoothness: for any integer $b \geq 2, C > 0, N > 0, f(bn) \leq Cf(n), f(n) \leq f(n+1)$, for all $n \geq N$

Binary Tree Traversal: preorder (root, left, right), inorder (left, root, right), postorder (left, right, root)

Binary Search Tree: to insert search through and put in spot where it would be, to delete node with 0 or 1 child simply delete like linked list, else find minimum in right subtree, swap with target, and delete

Heap: if array start from 1 then left child is $2i$, right child is $2i + 1$, to siftdown maxheap get larger child, if child is larger than current move child up, look at next child, to delete swap last in array with target, siftdown swapped, to insert do so at end, then siftup (if current is greater than parent move parent down, look at next parent), to heapify do siftdown from $i = n/2 \to 1$

Heap Complexities: lg n for delete, insert (O only), & siftdown (tree height), n for heapify

Indirect Heap: key, into (key index into heap index), outof (heap index into key index), whenever setting outof[x] = y also set into[y] = x, for increase/decreaseKey just siftup the modified node

Heapsort: heapify (maxheap), largest element is in the first, swap with last in the heap, decrease the heap length, siftdown on root, repeat until heap has only 1 element (which would be minimum)

Disjoint Sets: stored as array of parents, parent[i] is the parent of element i, makeset (constructs new set with one element from parameter, set parent to itself), findset (returns the marked member of the set containing parameter, keep going through parent, while doing so check the node parent, if not root make it root and continue to next parent), union (replace sets containing the parameters with the union, find the sets of both and make one of the sets the parent of the other, use the deepest tree as the parent)