Algorithms:

Algorithms	Time Complexity
Euclid’s Algorithm	$O(log_{min(a,b)})$
Middle-school procedure	$O(\sqrt{n})$
Maximum Element	$O(n)$
Matrix Multiplication	$O(n^3)$
Conventional Matrix Multiplication	$O(n^3)$
Strassen’s Matrix Multiplication	$O(n^{log_2(7)}) \approx O(n^{2.8})$
Counting binary digits	$O(log(n))$
Counting bits	$O(log(n))$
The Tower of Hanoi Puzzle	2^n
String Matching	$O(n \times m)$
Closest-Pair by Brute-force	$O(n^2)$
Convex-hull Problem by Brute-force	$O(n^2)$
Convex-hull Problem by devide and conquer	$O(nlog^2(n))$
Traveling Salesman Problem	$O(n!)$
The Assignment Problem	$O(n!)$
Knapsack Problem	2^n
BFS	$O(v)$ or $O(b^d)$
DFS	$O(b^m)$

Algorithms	Time Complexity
Decrease by constant factor	$O(log(n))$
Multiplication à la russe	$O(log(n))$
Exponentiation by Squaring	$O(log(n))$
Lumoto Partition	$O(n)$ or $O(r - l)$
Pre-Order	$O(n)$
In-Order	$O(n)$
Pre-Order	$O(n)$
First Multiplication of Large Integers	$O(n^2)$
Second Multiplication of Large Integers	$O(n^{1.585})$

Sorting Algorithms	Best Case	Average Case	Worst Case	Stable	In-place
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	❌	✓
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	✓	✓
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	✓	✓
Merge Sort	$O(nlog(n))$	$O(nlog(n))$	$O(nlog(n))$	✓	❌
Quick Sort	$O(nlog(n))$	$O(nlog(n))$	$O(n^2)$	❌	✓
Topology Sort	$O(V^2 + E)$	$O(V^2 + E)$	$O(V^2 + E)$	–	–
Median Selection	$O(n)$	$O(n)$	$O(n^2)$	–	–
Quick Select	$O(n)$	$O(n)$	$O(n^2)$	–	–
Binary Search	$O(1)$	$O(log(n))$	$O(log(n))$	–	–

Sorting Algorithms	Best Case	Average Case	Worst Case	Stable	In-place
Sequential Search	$O(1)$	$O(n)$	$O(n)$	–	–

Note:

- Topology Sort:
  - If it represented by a matrix it's complexity will be $O(V^2 + E)$
  - if it represented by a list it's complexity will be $O(V + E)$

In The Exam Choose $O(N^2)$ or $O(V^2 + E)$

Sorting key

- A specially chosen piece of information used to guide sorting. E.g., (sort student records by names.)

Two properties of sorting algorithms:

- **Stability**: A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input.
- **In place**: A sorting algorithm is in place if it does not require extra memory, except, possibly, for a few memory units.

String matching:

- searching for a given word/pattern in a text.

DEFINITION: A set of points in the plane is called convex if, for any two points p and q in the set, the entire line segment with the endpoints at p and q belongs to the set.

Graph:

- A graph is a collection of points called vertices, some of which are connected by line segments called edges.

- A graph $G = < V, E >$ is defined by a pair of two sets: a finite set V of items called vertices and a set E of vertex pairs called edges.

- Undirected and directed graphs (digraphs).

- What's the maximum number of edges in an undirected graph with $|V|$ vertices? $\sum v - 1 = \frac{v \times (v-1)}{2}$

- Complete, dense, and sparse graphs

    - A graph with every pair of its vertices connected by an edge is called complete, $K_{|V|}$

- **Examples of Graph Algorithms**:

    - Graph traversal algorithms
    - Shortest-path algorithms
    - Topological sorting

- **Graph Representation**:

    - Adjacency matrix
        - $n \times n$ boolean matrix if $|V|$ is n.
        - The element on the i_{th} row and j_{th} column is 1 if there's an edge from the i_{th} vertex to the j_{th} vertex; otherwise 0.
        - The adjacency matrix of an undirected graph is symmetric.
    - Adjacency linked lists
        - A collection of linked lists, one for each vertex, that contain all the vertices adjacent to the list's vertex.

Paths

- A path from vertex u to v of a graph G is defined as a sequence of adjacent (connected by an edge) vertices that starts with u and ends with v.

- Simple paths: All edges of a path are distinct.

- Path lengths: the number of edges, or the number of vertices – 1.

Connected graphs

- A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.

Connected component

- The maximum connected sub-graph of a given graph.

Cycle

- A simple path of a positive length that starts and ends at the same vertex.

Acyclic graph

- A graph without cycles
- DAG (Directed Acyclic Graph)

Cyclic graph

- A graph that contains at least one cycle

- The graphs are classified into various categories such as directed, non-directed, connected, non-connected, simple and multi-graph (multiple edges).
- A vertex in a graph can be connected to any number of other vertices using edges.
- An edge can be bidirected or directed.
- An edge can be weighted.
- A graph can have loops and self-loops.

Trees

- A tree (or free tree) is a connected acyclic graph.

Properties of trees:

- For every two vertices in a tree, there always exists exactly one simple path from one of these vertices to the other. Why?
- Rooted trees: The above property makes it possible to select an arbitrary vertex in a free tree and consider it as the root of the so, called rooted tree.
- $|E|$ =$|V| - 1$

Rooted Trees

- Depth of a vertex
    - The length of the simple path from the root to the vertex.
- Height of a tree
    - The length of the longest simple path from the root to a leaf.

Ordered trees

- An ordered tree is a rooted tree in which all the children of each vertex are ordered.

Binary trees

- A binary tree is an ordered tree in which every vertex has no more than two children, and each child is designated is either a left child or a right child of its parent.

Binary search trees

- Each vertex is assigned a number.
- A number assigned to each parental vertex is larger than all the numbers in its left subtree and smaller than all the numbers in its right subtree.

Tree search terminology

- **Root** : no parent.
- **Leaf**: no child.
- **Height**: distance from root to leaf (maximum Successor number of edges in path from root)
- **Level (depth)**: number of edge in the path from the root to that node.
- **Branch**: Internal node, neither the root nor the leaf.

- **Successor nodes of a node**: its children
- **Predecessor node of a node**: its parent
- **Path**: Sequence of nodes along the edges of a tree
- **Frontier**: The set of all leaf nodes available for expansion at any given point (nodes in memory)

The differences between Tree and Graph

Basis For Comparison	Tree	Graph
Path	Only one between two vertices	more than one path is allowed
Root node	It has exactly one root node	Graph doesn't have a root node
Loops	no loops are permitted	Graph can have loops
Complexity	Less Complex	More Complex
Traversal technique	Pre-order, In-order, Post-order	BFS, DFS
Number of edges	$n-1$ (n is the number of node)	not defined
Model type	Hierarchical	Network

Uniformed search strategies

- The uninformed search (also called blind search or unguided search).
- Blind search means that the strategies have no additional information about states beyond that provided in the problem definition.
- All search strategies are distinguished by the order in which nodes are expanded.
- Examples:
  - Breadth-first search
  - Uniform-cost search

    - Depth-first search
    - Depth-limited search
    - Iterative deepening depth-
    - first search
    - Bidirectional search

Breadth First Search (BFS) (Shortest path first)

- Breadth-first search is a simple strategy in which the root node is expanded first, then all the SEARCH successors of the root node are expanded next, then their successors, and so on.

- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

- Time Complexity

    - assume (worst case) that there is 1 goal leaf at the RHS so BFS will expand all nodes
    - = $1 + b + b^2 + ... + b^d$
    - = $O(b^d)$

- Space Complexity

    - how many nodes can be in the queue (worst-case)?
    - at depth d there are bd unexpanded nodes in the $Q = O(b^d)$

- **complete, optimal**

Depth First Search (DFS) (Longest path first)

- Depth-first search always expands the deepest node in the current frontier of the search tree.

- Time Complexity

    - Assume (worst case) that there is 1 goal leaf at the RHS so DFS will expand all nodes (m is maximum depth of any node)
    - = $1 + b + b^2 + ... + b^m$
    - = $O(b^m)$

- Space Complexity

    - How many nodes can be in the queue (worst-case)?
        - at depth $m < d$ we have $b - 1$ nodes at depth d we have b nodes total = $(m-1) \times (b-1) + b = O(bm)$

- **Infinite, not complete, not optimal**

Comparing DFS and BFS:

- Same worst-case time Complexity, but

    - In the worst-case BFS is always better than DFS

    - Sometime, on the average DFS is better if:

        - many goals, no loops and no infinite paths

- BFS is much worse memory-wise

    - DFS is linear space

    - BFS may store the whole search space.

- In general

    - BFS is better if goal is not deep, if infinite paths, if many loops, if small search space

    - DFS is better if many goals, not many loops,

    - DFS is much better in terms of memory

Tree Traverse:

- Pre-Order: Process all nodes of a tree by processing the root, then recursively processing all subtrees.
- In-Order: Process all nodes of a tree by recursively processing the left subtree, then processing the root, and finally the right subtree.
- Post-Order: Process all nodes of a tree by recursively processing all subtrees, then finally processing the root.

Arrays

- A sequence of n items of the same data type that are stored contiguously in computer memory and made accessible by specifying a value of the array's index.

Linked List

- A sequence of zero or more nodes, each containing two kinds of information: some data and one or more links called pointers to other nodes of the linked list.
    - Singly linked list (next pointer)
    - Doubly linked list (next + previous pointers)

Stacks

- A stack of plates
    - insertion/deletion can be done only at the top.
    - LIFO
    - Two operations (push and pop)

Queues

- A queue of customers waiting for services
    - Insertion/enqueue from the rear and deletion/dequeue from the front.
    - FIFO
    - Two operations (enqueue and dequeue)

Priority queues (implemented using heaps)

- A data structure for maintaining a set of elements, each associated with a key/priority, with the following operations:
    - Finding the element with the highest priority
    - Deleting the element with the highest priority
    - Inserting a new element
    - Scheduling jobs on a shared computer

Comp	Array	Linked Lists
Length	fixed length	dynamic length
Memory	contiguous memory locations	arbitrary memory locations
Access	direct access	access by following links

Comparing Orders of Growth (Limit)

- $\lim_{n=\infty} \frac{t(n)}{g(n)}$:
    - 0 implies that $t(n)$ has smaller order of growth than $g(n)$
    - c implies that $t(n)$ has the same order of growth than $g(n)$
    - ∞ implies that $t(n)$ has larger order of growth than $g(n)$
- $\lim_{n=\infty} \frac{t(n)}{g(n)}$ = $\lim_{n=\infty} \frac{t'(n)}{g'(n)}$
- $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

Useful summation formulas and rules

- $\sum_{i=1}^{n} 1$ = $1 + 1 + ... + 1$ = n
- $\sum_{i=u}^{v} 1$ = $1 + 1 + ... + 1$ = $v - u + 1$

- $\sum_{i=1}^{n} i = 1 + 2 + ... + n = \frac{n \times (n+1)}{2} \approx \frac{1}{2}n^2$
- $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + ... + n^2 = \frac{n \times (n+1) \times (2n+1)}{6} \approx \frac{1}{3}n^3$
- $\sum_{i=1}^{n} lg(i) \approx n \times lg(n)$

What is Brute Force?

- A straightforward approach to solving a problem, usually based on problem statement and definitions of the concepts involved
- “**Force**” comes from using computer power not **intellectual** power
- In short, “**brute force**” means “**Just do it!**”
- Examples:
    - Consecutive Integer Checking for ***gcd(m, n)***
    - Definition based **matrix-multiplication**
- It is the only general approach that always works
- Seldom gives efficient solution, but one can easily improve the brute force version.
- Usually can solve small sized instances of a problem

Exhaustive Search

- Many Brute Force Algorithms use Exhaustive Search

- A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.

- Example:

    - Brute force The Closest Pair
    - Traveling Salesman Problem (TSP)
    - Knapsack Problem
    - Assignment Problem

- Approach:

    1. Enumerate and evaluate all solutions, and
    2. Choose solution that meets some criteria (e.g. smallest)

- Frequently the obvious solution But, **slow**

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances

- In many cases, exhaustive search or its variation is the only known way to get exact solution

---

Measuring problem-solving performance

- Completeness: Is the algorithm guaranteed to find a solution when there is one?

- Optimality: Does the strategy find the optimal solution?

- Time complexity: How long does it take to find a solution?

- Space complexity: How much memory is needed to perform the search?

Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance

- Can be implemented either top- down or bottom-up

- Also referred to as inductive or incremental approach

- Examples:

    - Decrease by a constant (usually by 1):

        - insertion sort
        - topological sorting

- Decrease by a constant factor (usually by half)
    - binary search and bisection method
    - exponentiation by squaring
    - multiplication à la russe
- Variable-size decrease
    - Euclid’s algorithm for greatest common divisor
    - Partition-based algorithm for selection problem

Divide-and-Conquer

1. Divide instance of problem into two or more smaller instances
2. Solve smaller instances recursively
3. Obtain solution to original (larger) instance by combining these solutions

Examples:

- Sorting: mergesort and quicksort
- Binary tree traversals
- Binary search
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms

Notes:

- Rate of Growth or Growth rate —> (Slower or Faster)
- Order of Growth —> (Smaller or Larger)
- Backtracking is like DFS
- Strassen’s Matrix Multiplication —> (7 multiplications - 18 additions)
- Conventional Matrix Multiplication —> (8 multiplications - 4 additions)