Algorithms Cheat Sheet

A quick reference guide to common algorithms and data structures in computer science, covering time complexity, pseudocode, and applications.

Sorting Algorithms

Comparison Sorts

Bubble Sort	Repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. **Time Complexity:** O(n^2)
Insertion Sort	Builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms. **Time Complexity:** O(n^2)
Selection Sort	Divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. **Time Complexity:** O(n^2)
Merge Sort	A divide and conquer algorithm that divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. **Time Complexity:** O(n log n)
Quick Sort	A divide and conquer algorithm that picks an element as pivot and partitions the given array around the picked pivot. **Time Complexity:** O(n log n) average, O(n^2) worst case
Heap Sort	Heap sort involves building a Heap data structure from the array and then repeatedly extracting the maximum element from the Heap and placing it at the end of the array. **Time Complexity:** O(n log n)

Non-Comparison Sorts

Counting Sort	Works by counting the number of occurrences of each distinct element in the input array. **Time Complexity:** O(n + k), where k is the range of input
Radix Sort	Sorts elements by processing individual digits. It groups elements by the digit in the same position and repeats until all digits have been processed. **Time Complexity:** O(nk), where k is the number of digits
Bucket Sort	Distributes the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. **Time Complexity:** O(n + k) average, O(n^2) worst case

Searching Algorithms

Basic Search Algorithms

Linear Search	Sequentially checks each element of the list until a match is found or the whole list has been searched. **Time Complexity:** O(n)
Binary Search	Searches a sorted array by repeatedly dividing the search interval in half. Requires the input data to be sorted. **Time Complexity:** O(log n)
Jump Search	Like binary search, but jumps ahead by fixed steps. The optimal size of a block to be jumped is (\sqrt{n}). **Time Complexity:** O((\sqrt{n}))
Interpolation Search	An improvement over binary search for uniformly distributed data. It estimates the position of the required value. **Time Complexity:** O(log log n) average, O(n) worst case

Graph Algorithms

Basic Graph Traversal

Breadth-First Search (BFS)	Traverses a graph level by level. Starts at the root node and explores all the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. **Time Complexity:** O(V + E), where V is the number of vertices and E is the number of edges.
Depth-First Search (DFS)	Explores as far as possible along each branch before backtracking. It uses a stack to remember where to go when it reaches a dead end. **Time Complexity:** O(V + E)

Shortest Path Algorithms

Dijkstra's Algorithm	An algorithm for finding the shortest paths between nodes in a graph. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. **Time Complexity:** O(V^2 + E) or O(E log V) with a priority queue
Bellman-Ford Algorithm	Computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. **Time Complexity:** O(V * E)
Floyd-Warshall Algorithm	An algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices. **Time Complexity:** O(V^3)

Minimum Spanning Tree Algorithms

Kruskal's Algorithm	A greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. **Time Complexity:** O(E log E) or O(E log V)
Prim's Algorithm	A greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. **Time Complexity:** O(E + V log V) using Fibonacci heap

Dynamic Programming

Common Dynamic Programming Problems

Fibonacci Sequence

Calculating the nth Fibonacci number using dynamic programming to avoid redundant calculations.

Knapsack Problem

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Longest Common Subsequence (LCS)

Find the longest subsequence common to all sequences in a set of sequences (often just two sequences).

Edit Distance

The minimum number of edits (insertions, deletions, or substitutions) needed to transform one string into another.

Matrix Chain Multiplication

Finding the most efficient way to multiply a given sequence of matrices.