Data Structures & Algorithms Notes

Comprehensive Reference Guide

This document provides an overview of fundamental data structures and algorithms, including definitions, properties, and example use cases. Suitable for students and interview preparation.

Table of Contents

1. Arrays

Definition: An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together.

Properties:

- Fixed size
- O(1) access by index
- O(n) search (if unsorted)

Example:

int arr[5] = {1, 2, 3, 4, 5};

Use Cases:

- Storing lists of items
- Implementing other data structures (heaps, hash tables)

2. Linked Lists

Definition: A linked list is a linear data structure where each element is a separate object, called a node, which contains data and a reference to the next node.

Types:
- Singly Linked List
- Doubly Linked List
- Circular Linked List

Properties:
- Dynamic size
- O(1) insertion/deletion at head
- O(n) search

Example Node (C++):

```
struct Node {
    int data;
    Node* next;
};
```

Use Cases:
- Implementing stacks and queues
- Memory management

3. Stacks

Definition: A stack is a linear data structure that follows the Last In First Out (LIFO) principle.

Operations:

- push(x): Add x to the top
- pop(): Remove and return the top
- peek(): Return the top without removing

Properties:

- O(1) push/pop

Use Cases:

- Function call management
- Undo operations
- Expression evaluation

4. Queues

Definition: A queue is a linear data structure that follows the First In First Out (FIFO) principle.

Operations:

- enqueue(x): Add x to the rear
- dequeue(): Remove and return the front

Types:

- Circular Queue
- Priority Queue
- Deque (Double-ended queue)

Use Cases:

- Scheduling
- Buffer management

5. Trees

Definition: A tree is a hierarchical data structure consisting of nodes, with a single root node and potentially many levels of additional nodes.

Types:

- Binary Tree
- Binary Search Tree (BST)
- AVL Tree
- Heap

Properties:

- Recursive structure
- Used for hierarchical data

Use Cases:

- File systems
- Databases (B-trees)
- Expression parsing

6. Graphs

Definition: A graph is a collection of nodes (vertices) and edges connecting pairs of nodes.

Types:

- Directed/Undirected
- Weighted/Unweighted

Representations:

- Adjacency Matrix
- Adjacency List

Use Cases:

- Social networks
- Routing algorithms
- Network analysis

7. Sorting Algorithms

Common Algorithms:

- Bubble Sort: O(n^2)
- Selection Sort: O(n^2)
- Insertion Sort: O(n^2)
- Merge Sort: O(n log n)
- Quick Sort: O(n log n) average
- Heap Sort: O(n log n)

Use Cases:

- Data organization
- Searching optimization

8. Searching Algorithms

Common Algorithms:

- Linear Search: O(n)
- Binary Search: O(log n) (sorted arrays)

Use Cases:

- Data retrieval
- Database queries

9. Hash Tables

Definition: A hash table is a data structure that implements an associative array, a structure that can map keys to values using a hash function.

Properties:

- O(1) average-case lookup, insert, delete
- Collisions handled by chaining or open addressing

Use Cases:

- Caches
- Symbol tables
- Sets and maps

10. Recursion

Definition: Recursion is a method of solving problems where a function calls itself as a subroutine.

Properties:

- Base case and recursive case
- Used in divide and conquer algorithms

Examples:

- Factorial
- Fibonacci sequence
- Tree traversals

11. Dynamic Programming

Definition: Dynamic programming is a technique for solving problems by breaking them down into simpler subproblems and storing the results of subproblems to avoid redundant computation.

Properties:

- Overlapping subproblems
- Optimal substructure

Examples:

- Fibonacci sequence
- Knapsack problem
- Longest common subsequence