Algorithms Foundations

From Stacks to Binary Heaps

Structure Determines Performance

Table of Contents

1. Data Structures vs Algorithms

A data structure is a container with guarantees. An algorithm is a rule set that assumes those guarantees. Performance emerges from structure, not from implementation tricks.

Concept Anchor: Algorithms assume. Containers guarantee.

Review Questions

- Why does binary search require random access?
- What structural property allows heapify to work efficiently?

2. Stacks and Queues

Stacks enforce Last-In-First-Out discipline. Queues enforce First-In-First-Out discipline. Both operate in constant time when properly implemented.

> **Concept Anchor:** Control discipline shapes traversal behavior.

Review Questions

- Why is DFS naturally paired with a stack?
- Why is BFS naturally paired with a queue?

3. Recursion vs Iteration

Recursion leverages the call stack. Iteration uses explicit control structures. Recursive structures like trees align naturally with recursive algorithms.

> **Concept Anchor:** Recursion is a stack you didn’t write.

Review Questions

- When does recursion become dangerous?
- Can DFS be implemented without recursion?

4. Trees and Traversals

A tree is a connected, acyclic graph with n-1 edges. Tree height governs performance. Balanced trees achieve logarithmic height.

> **Concept Anchor:** Height determines complexity.

Review Questions

- Why must a tree with n nodes have n-1 edges?
- Why does inorder traversal sort only BSTs?

5. Binary Search Trees

BSTs impose ordering: left < root < right. Insertion order determines balance. Skewed BSTs degrade to linear time.

> **Concept Anchor:** A skewed BST is a linked list in disguise.

Review Questions

- What causes BST degeneration?
- Explain the role of inorder successor in deletion.

6. Binary Search

Binary search requires sorted data and random access. It reduces search space by half each step, achieving O(log n).

Concept Anchor: Binary search exploits ordering and indexing.

Review Questions

- Why does binary search fail on linked lists?
- Compare BST search with binary search.

7. Graphs

Graphs generalize trees. Representation choices (list vs matrix) depend on edge growth rate.

> **Concept Anchor:** Sparse vs dense is about growth rate.

Review Questions

- Why is adjacency matrix inefficient for sparse graphs?
- What is the complexity of BFS?

8. Array-Based Trees

Complete trees map perfectly into arrays using index formulas. Sparse trees waste space when represented this way.

> **Concept Anchor:** Completeness enables efficient array mapping.

Review Questions

- What are the parent/child index formulas?
- Why do sparse trees waste space in arrays?

9. Binary Heaps

Binary heaps maintain a complete tree and heap-order property. Insert and remove are O(log n). Heapify is O(n).

Concept Anchor: Heaps guarantee urgency, not total ordering.

Review Questions

- Why must heaps be complete?
- Explain why heapify runs in O(n).

10. Structural Synthesis & Final Review

Throughout this course, we have seen how structure dictates performance. From stacks to heaps, the consistent theme is that constraints enable efficiency.

Concept Anchor: Structure determines performance.

Review Questions

- Why does height govern complexity in trees and heaps?
- Why is local ordering sufficient for priority queues?
- Explain sparse vs dense using asymptotic growth.