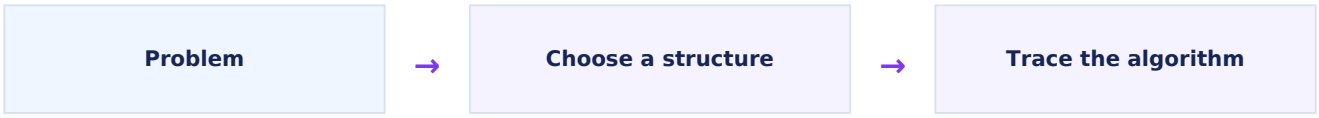
LEARNHUB / TECHNICAL COURSE LIBRARY

Data Structures and Algorithms

Practical examples

Learn how common data structures organize information and how algorithms transform it, with an emphasis on choosing a suitable approach and explaining its cost.



Level: Beginner · Category: Computer Science

Original, beginner-friendly study content designed to accompany the course syllabus and its linked video lesson.

Contents

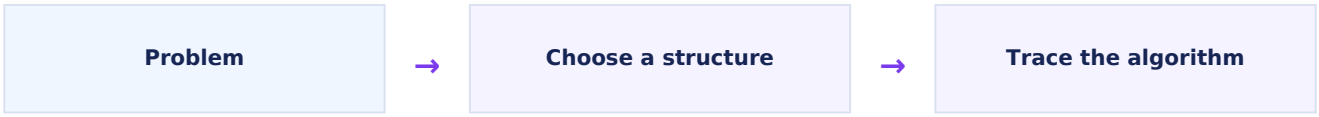
- How to use this resource
- Learning map
- Chapter 01 · Complexity and arrays
- Chapter 02 · Linked lists, stacks, and queues
- Chapter 03 · Trees, heaps, and ordered search
- Chapter 04 · Graphs and traversal
- Chapter 05 · Searching, sorting, and strategy
- Summary and next steps

Course purpose: Learn how common data structures organize information and how algorithms transform it, with an emphasis on choosing a suitable approach and explaining its cost.

How to use this resource

Read the chapter titles first, then study the explanations and examples. Pause at each practice prompt before reading its suggested answer. Write down questions that remain and revisit them during revision.

Learning map



Chapter 01 · Complexity and arrays

Problem context

Compare scanning an unsorted list with looking up an index in an array. State the assumption behind each cost.

Concept to apply

An algorithm is a finite sequence of steps for solving a problem. Big-O notation describes how resource use grows as input size grows, while an array stores indexed elements in contiguous logical order.

Worked example

A one-pass scan for a target visits at most each array item, so its work grows linearly with the number of items.

Walk-through

Identify input size, count the dominant work, and distinguish time from extra memory. Indexed array access is fast, while inserting in the middle usually shifts later elements. Big-O describes growth, not wall-clock seconds.

Check your result

> Index access is constant-time under the usual array model. A value search in an unsorted array is linear because the target may be last or absent.

Chapter 02 · Linked lists, stacks, and queues

Problem context

Choose a queue or stack for processing print jobs in arrival order, then describe an operation that removes the next item.

Concept to apply

A linked list connects nodes through references. A stack removes the most recently added item; a queue removes the earliest added item.

Worked example

A browser's undo history behaves like a stack: a new action is pushed and undo pops the latest action.

Walk-through

Pick a structure by its operations. Linked lists support local node changes when a reference is known but require traversal for position lookup. Stacks and queues make ordering rules explicit and are useful in parsing, scheduling, and breadth-first search.

Check your result

> Use a queue, typically with enqueue and dequeue operations. A stack would reverse the expected processing order.

Chapter 03 · Trees, heaps, and ordered search

Problem context

Insert keys 8, 4, and 12 into a binary search tree and describe where each value belongs.

Concept to apply

A tree is a connected hierarchy of nodes with a root and child relationships. A binary search tree orders keys so smaller keys lie on one side and larger keys on the other.

Worked example

A priority queue schedules the next most urgent task by storing tasks in a min-heap keyed by deadline.

Walk-through

Search cost in a balanced binary search tree grows with its height; an unbalanced tree can degrade toward a list. A heap maintains a parent-child priority rule and supports efficient access to the highest- or lowest-priority item.

Check your result

> 8 becomes the root; 4 is placed in the left subtree because it is smaller; 12 is placed in the right subtree because it is larger.

Chapter 04 · Graphs and traversal

Problem context

For an unweighted social network, choose a traversal to find all people within two connections of a starting person.

Concept to apply

A graph consists of vertices and edges. Edges may be directed or undirected and may carry weights such as distance or cost.

Worked example

A route map models intersections as vertices and roads as weighted edges; Dijkstra's algorithm can find a shortest path when edge costs are nonnegative.

Walk-through

Represent sparse graphs with adjacency lists and dense graphs with matrices when their access pattern warrants it. Depth-first search explores deeply before backtracking; breadth-first search explores neighbors in layers and finds shortest paths in unweighted graphs.

Check your result

> Breadth-first search visits nodes by increasing number of edges from the start. Track visited vertices to avoid revisiting cycles.

Chapter 05 · Searching, sorting, and strategy

Problem context

List the prerequisite that makes binary search correct, then choose a sort for a large general dataset.

Concept to apply

Searching locates an item; sorting orders items by a comparison rule. Algorithm analysis compares correctness, time growth, space use, and assumptions.

Worked example

A directory sorted by last name can use binary search to narrow the location of an entry.

Walk-through

Binary search requires sorted data and halves its remaining search interval each step. Merge sort uses divide-and-conquer for predictable O(n log n) time. Use a simple sort for tiny or nearly sorted data when its trade-off is appropriate.

Check your result

> The data must be ordered by the same comparison used in the search. Merge sort or another O(n log n) algorithm is a reasonable general choice; constraints matter.

Summary and next steps

Data Structures and Algorithms brings together 5 connected topics. Review each definition, explain one example without looking, and then try the practice prompt again.

Key ideas to carry forward

Complexity and arrays: An algorithm is a finite sequence of steps for solving a problem. Big-O notation describes how resource use grows as input size grows, while an array stores indexed elements in contiguous logical order.

Linked lists, stacks, and queues: A linked list connects nodes through references. A stack removes the most recently added item; a queue removes the earliest added item.

Trees, heaps, and ordered search: A tree is a connected hierarchy of nodes with a root and child relationships. A binary search tree orders keys so smaller keys lie on one side and larger keys on the other.

Graphs and traversal: A graph consists of vertices and edges. Edges may be directed or undirected and may carry weights such as distance or cost.

Searching, sorting, and strategy: Searching locates an item; sorting orders items by a comparison rule. Algorithm analysis compares correctness, time growth, space use, and assumptions.

A short review routine

1. Recall the main idea from each chapter before rereading. 2. Work one example by hand. 3. Explain a common mistake and how you would catch it. 4. Revisit a difficult topic after a short break.

Course outcomes

- Compare time and space cost using Big-O notation
- Choose among arrays, linked lists, stacks, and queues
- Explain tree and graph representations
- Trace searching and sorting algorithms

This practical examples is an original LearnHub learning aid. Use the referenced ideas for study, then consult the course video and the rest of the library for additional practice.