

Algorithms Made Simple

A practical two-page reference for choosing, comparing, and reasoning about common algorithms.

1. Complexity at a glance

Complexity	What it feels like	Typical example
$O(1)$	Work stays constant	Array lookup by index
$O(\log n)$	Problem shrinks each step	Binary search
$O(n)$	Touch each item once	Linear scan
$O(n \log n)$	Efficient comparison sorting	Merge sort
$O(n^2)$	Compare many pairs	Simple nested loops

Rule of thumb: analyze the dominant growth term and describe both time and extra space.

2. Core patterns

Linear scan

Use when data is unsorted or you must inspect every item.
Track only the best answer or state you need.

Binary search

Use on sorted data or any monotonic yes/no condition.
Compare the middle, then discard half.

Two pointers

Move two indices through a sequence to find pairs,
windows, partitions, or palindromes efficiently.

Hash lookup

Trade extra memory for fast membership, counting,
deduplication, or key-to-value retrieval.

Divide and conquer

Split into smaller independent problems, solve recursively,
then combine their answers.

Dynamic programming

Cache overlapping subproblems when the same state
would otherwise be solved repeatedly.

3. Data structure chooser

Need	Start with	Why
Fast indexed access	Array / list	Direct position-based retrieval
First-in, first-out work	Queue	Processes items in arrival order
Last-in, first-out work	Stack	Supports nesting and backtracking
Fast membership or counting	Hash set / map	Average constant-time lookup
Repeated smallest/largest item	Heap / priority queue	Efficient priority removal
Relationships and routes	Graph	Models nodes and connections
Hierarchical information	Tree	Represents parent-child structure

4. A reliable problem-solving loop

1	Clarify	State the input, required output, constraints, and important edge cases.
2	Model	Choose the data structure that makes the key operation easy.
3	Start simple	Write the correct brute-force idea before optimizing.
4	Find repeated work	Look for sorting, caching, hashing, pruning, or a better traversal.
5	Prove it	Explain the invariant: what remains true after every step?
6	Measure	State time and space complexity, then test tiny and extreme cases.

5. Edge-case checklist

Empty input	One element	Duplicates
Already sorted	Reverse sorted	Missing target
Negative / zero values	Largest allowed input	Disconnected graph

Keep learning

Use this sheet as a decision aid, then deepen the intuition with visual explanations and worked examples in **Algorithms Made Simple** by Choral Software.

choralsoft.com/products/