

Introduction to Algorithms

Introduction to Algorithms – Study Notes

1. What Is an Algorithm?

Aspect	Description
Definition	A finite, well-defined sequence of steps that transforms an input into an output.
Key Properties	• Finiteness – terminates after a limited number of steps.
	• Determinism – each step is precisely specified.
	• Input/Output – may have zero or more inputs; produces at least one output.
	• Effectiveness – each operation is simple enough to be performed by a computer (or a person).
Analogy	Recipe, instruction manual, or a flowchart for solving a problem.

2. Why Study Algorithms?

1. **Performance Guarantees** – Predict how fast a solution runs (time) and how much memory it uses (space).
 2. **Problem Solving** – Provide reusable patterns (e.g., divide-and-conquer, greedy, dynamic programming).
 3. **Scalability** – Choose solutions that remain feasible as data sizes grow.
 4. **Correctness** – Formal reasoning and proofs give confidence that the solution works for all inputs.
-

3. Basic Terminology

Term	Meaning
Instance	A specific input to the problem (e.g., a particular graph).
Complexity	Measure of resources needed (time, space).
Worst-case	Upper bound on resources for any input of size n .
Average-case	Expected resources over a distribution of inputs.
Asymptotic Notation	Describes growth rates: O , Θ , Ω , o , ω .

Introduction to Algorithms

Term	Meaning
Pseudocode	Language-independent description of an algorithm.
Correctness	Proof that the algorithm always produces the intended output.
Deterministic vs. Non-deterministic	Deterministic: single execution path; Non-deterministic: may explore many paths (e.g., backtracking).

4. Algorithm Design Paradigms

Paradigm	Core Idea	Typical Problems
Brute Force	Enumerate all possibilities.	Subset sum, permutation generation.
Divide & Conquer	Split problem ? solve subproblems ? combine.	Merge sort, binary search, FFT.
Greedy	Make locally optimal choice at each step.	Minimum spanning tree (Kruskal, Prim), Huffman coding, activity selection.
Dynamic Programming	Break into overlapping subproblems; store solutions.	Knapsack, longest common subsequence, Floyd-Warshall.
Backtracking	Incrementally build candidates, abandon when impossible.	N-Queens, Sudoku solver.
Randomized	Use random choices to improve expected performance.	QuickSort (random pivot), Monte Carlo algorithms.
Parallel / Distributed	Partition work across multiple processors.	Parallel matrix multiplication, MapReduce.

5. Analyzing Algorithms

5.1 Time Complexity

1. **Count elementary operations** (comparisons, assignments, arithmetic).
2. Express as a function $T(n)$ of input size n .
3. Simplify using asymptotic notation.

Complexity Class	Typical Growth	Example
$O(1)$	Constant	Access array element

Introduction to Algorithms

Complexity Class	Typical Growth	Example
$O(\log n)$	Logarithmic	Binary search