

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 – must terminate.
	• Determinism – each step is unambiguous.
	• Input/Output – operates on given data and produces results.
	• Effectiveness – every operation is basic enough to be performed mechanically.
Analogy	Recipe for a dish, blueprint for a building, or a set of instructions for a robot.

2. Why Study Algorithms?

1. **Performance** – Determines how fast a program runs and how much memory it uses.
 2. **Scalability** – Good algorithms handle large inputs gracefully.
 3. **Correctness** – Formal reasoning helps guarantee that solutions are right.
 4. **Transferable Skills** – Techniques (divide and conquer, greedy, dynamic programming) apply across domains.
-

3. Algorithm Design Paradigms

Paradigm	Core Idea	Typical Problems
Brute Force	Try every possibility.	Small search spaces, verification.
Divide & Conquer	Split problem ? solve sub-problems ? combine.	Merge sort, quicksort, binary search.
Greedy	Make locally optimal choice at each step.	Minimum spanning tree (Kruskal, Prim), Huffman coding.
Dynamic Programming	Break into overlapping sub-problems; store results (memoization).	Shortest paths (Bellman-Ford, Floyd-Warshall), knapsack, edit distance.

Introduction to Algorithms

Paradigm	Core Idea	Typical Problems
Backtracking	Incrementally build solutions, backtrack on conflicts.	N-Queens, Sudoku, subset sum.
Randomized	Use random choices to simplify or speed up.	Quickselect, Monte Carlo methods, randomized quicksort.
Recursive	Function calls itself with smaller input.	Tree traversals, factorial, Fibonacci (with memoization).

4. Analyzing Algorithms

4.1 Time Complexity

- **Worst-case:** maximum running time over all inputs of size n .
- **Average-case:** expected running time assuming a probability distribution on inputs.
- **Big-O notation:** Upper bound, e.g., $O(n^2)$.
- **Common classes**

Class	Typical Example	Growth
$O(1)$	Access array element	Constant
$O(\log n)$	Binary search	Logarithmic
$O(n)$	Linear scan	Linear
$O(n \log n)$	Merge sort	Linearithmic
$O(n^2)$	Bubble sort	Quadratic
$O(2^n)$	Exhaustive subset search	Exponential
$O(n!)$	Permutation generation	Factorial

4.2 Space Complexity

- Measures extra memory used beyond input storage.
- Same notation (O , Θ , Ω).
- Example: recursive quicksort uses $O(\log n)$ stack space on average.

4.3 Asymptotic Bounds

Symbol	Meaning
--------	---------

Introduction to Algorithms

$O(f(n))$ Upper bound ($\exists c \cdot f(n)$ for large n)

$\Omega(f(n))$ Lower bound ($\exists c \cdot f(n)$ for large n)

$\Theta(f(n))$ Tight bound (both O and Ω)

Θ