# Limitations of Algorithmic Power

- Decision Trees
- P, NP & NP-Complete Problems
- Backtracking
- Branch-and-Bound
- Approximation Algorithms

# Outline

## A Decision Trees

Lower bounds via decision tree model for comparison-based algorithms

## B P, NP & NP-Complete

Complexity classes, polynomial vs exponential, Cook's theorem

## C Backtracking

N-Queens problem, Subset-Sum problem, systematic state-space search

## D Branch-and-Bound

Best-first search, bounding functions, Knapsack problem

## E Approximation Algorithms

Near-optimal solutions for NP-Hard problems, Knapsack approximation

Ref: Levitin — Chapters 10, 11, 12

UNIT 5

# Decision Trees & Lower Bounds

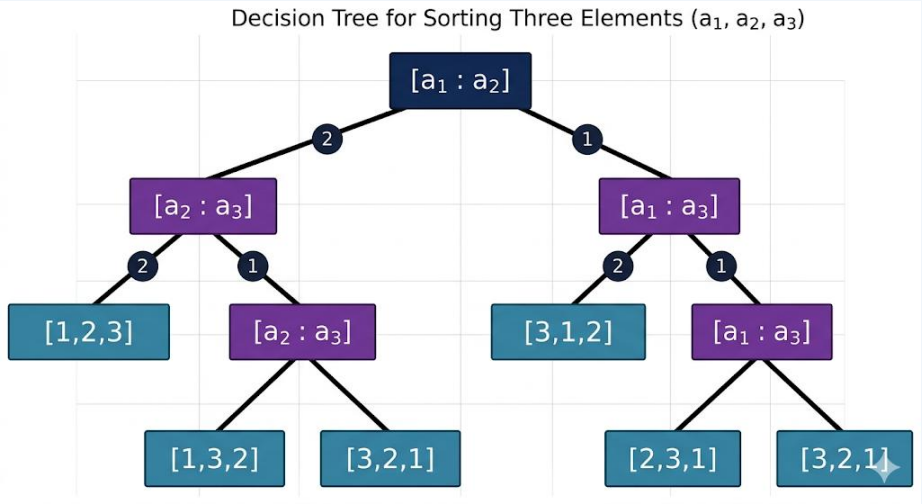
Limitations of Algorithmic Power

# Decision Trees

*A technique to establish lower bounds on algorithms*

## What is a Decision Tree?

- A rooted binary tree modeling all possible executions of a comparison-based algorithm
- Each internal node = a comparison between two elements
- Each leaf = a possible outcome (e.g., sorted order or element found)
- Any algorithm must follow a root-to-leaf path for each input
- Height of tree ≥ lower bound on worst-case comparisons




**Lower Bound Theorem:** Any comparison-based sorting algorithm must make at least $\lceil \log_2(n!) \rceil \approx n \log_2 n$ comparisons in the worst case. This proves merge sort & heap sort are optimal.

UNIT 5

# P, NP, and NP-Complete Problems

Limitations of Algorithmic Power

# Complexity Classes: P, NP, NP-Complete

*The central question of computer science: P = NP?*

## Class P

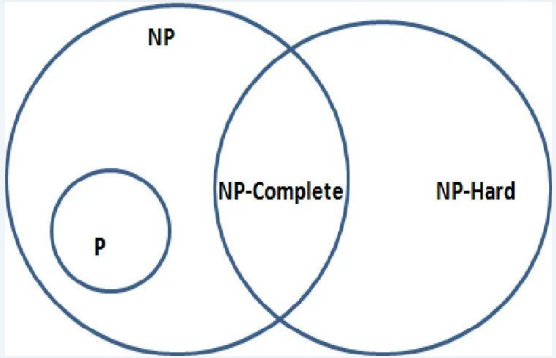
Problems solvable in polynomial time $O(n^k)$ by a deterministic algorithm.
Examples: Searching, Sorting, Shortest Path, MST, Matrix Multiplication

## Class NP

Problems verifiable in polynomial time. Solutions can be checked quickly, but finding them may be hard.
Examples: Hamiltonian Cycle, Subset Sum, Graph Coloring

## NP-Complete

Both NP and NP-Hard. Every NP problem reduces to them.
Examples: SAT, TSP, Knapsack, 3-SAT, Vertex Cover




*If P ≠ NP (widely believed), then NP-Complete problems have no efficient algorithms.*

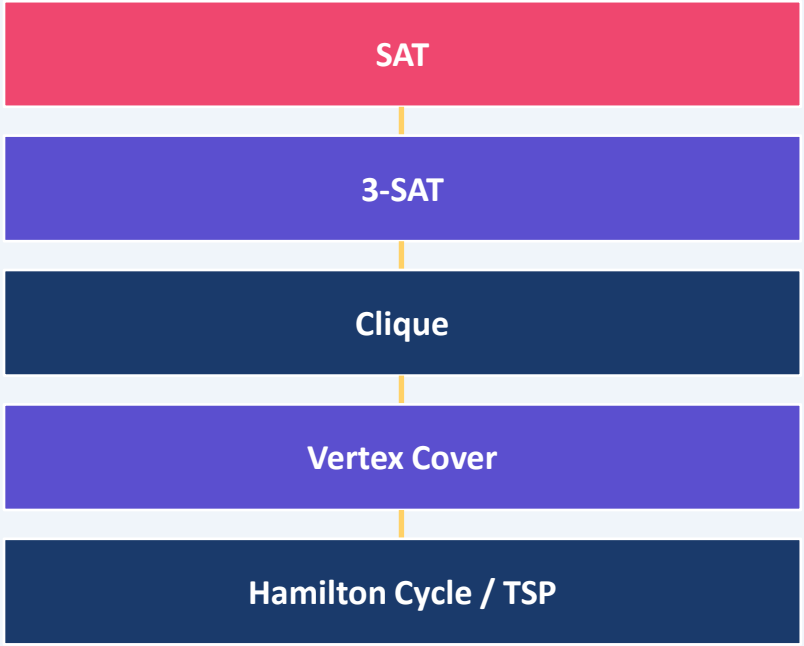
# NP-Completeness & Cook's Theorem

*Polynomial Reduction: the key to proving NP-Completeness*

## Polynomial Reduction

- Problem A reduces to B in polynomial time (A ≤p B) if any instance of A can be transformed to B in $O(n^k)$
- If B is easy (P), then A is also easy
- If A is hard, then B must also be hard
- A problem is NP-Hard if every NP problem reduces to it
- A problem is NP-Complete if it is NP-Hard AND belongs to NP

**Cook's Theorem (1971):** SAT (Boolean Satisfiability) is NP-Complete. Every NP problem can be reduced to SAT.




Chain of Polynomial Reductions

UNIT 5

# Coping with Limitations: Backtracking

Systematic exploration of the state-space tree

# Backtracking — Core Concept

*Exhaustive search with intelligent pruning*

| 1 Build Tree | 2 Check | 3 Prune | 4 Backtrack |
| --- | --- | --- | --- |
| Construct state-space tree node by node | At each node, check if constraint violated | Cut subtree if no solution possible (dead end) | Return to parent and try next branch |

## Key Idea

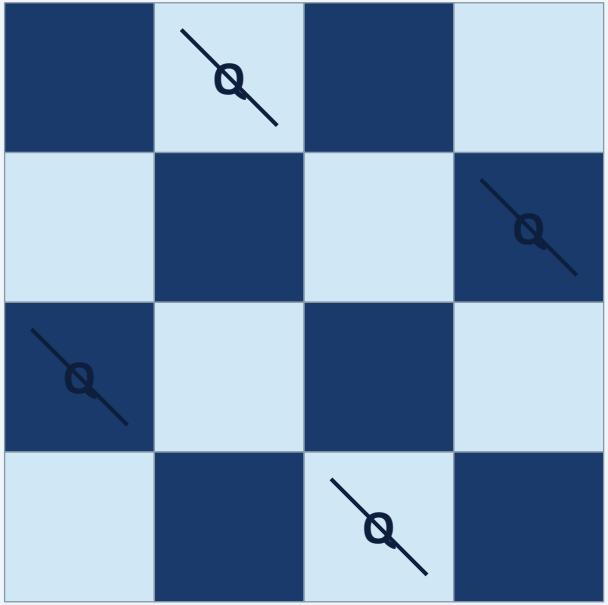
- More efficient than brute-force by avoiding unnecessary paths in the search tree
- A node is a promising node if it may lead to a valid solution (constraints not violated)
- Non-promising nodes are immediately abandoned — the subtree is pruned
- Algorithm is recursive: solve(node) → try all extensions → backtrack if none work
- State-space tree: tree of all possible partial solutions

# Backtracking: N-Queens Problem

*Place N queens on N×N board — no two queens attack each other*

## Problem Statement

- Place N queens on an N×N chessboard
- No two queens share the same row, column, or diagonal
- Classic for 8-Queens (92 solutions exist)
- State-space: n! permutations, reduced by pruning
- Constraint check: row[i] ≠ row[j], |i−j| ≠ |row[i]−row[j]|



4-Queens Solution: [2,4,1,3]

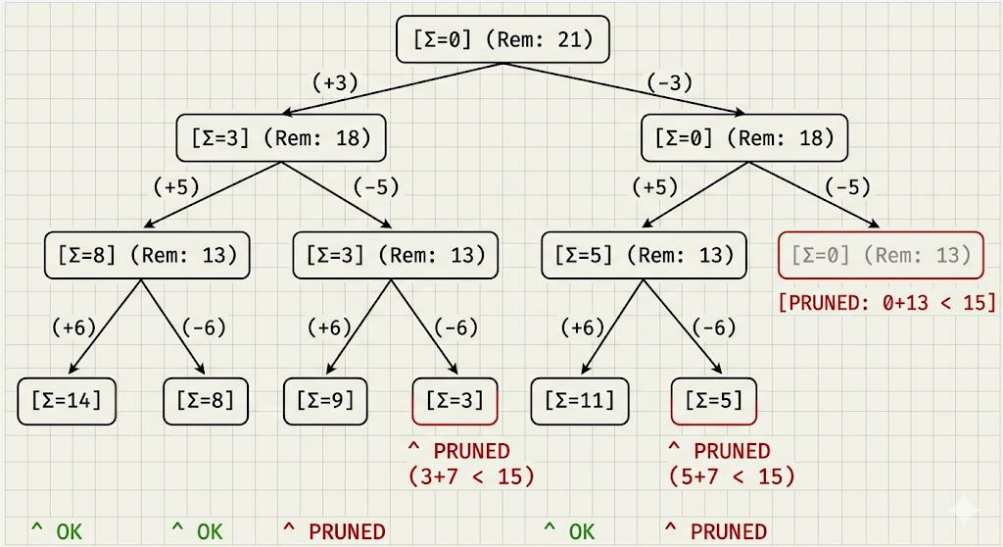
Backtracking prunes approx. 95% of the brute-force search space for the 8-Queens problem.

# Backtracking: Subset-Sum Problem

*Find a subset of a given set with elements summing to target d*

## Problem Definition

- Given: set S = {$s_1$, $s_2$, ..., $s_n$} of positive integers and target d
- Find: all subsets of S that sum exactly to d
- Example: S = {3, 5, 6, 7}, d = 15 → {3, 5, 7}
- State-space: binary tree of include/exclude decisions
- Pruning rules:
- • If sum > d: prune (too large)
- • If sum + remaining < d: prune (can't reach target)



UNIT 5

# Branch-and-Bound: Knapsack Problem

Optimal state-space search for minimization/maximization

# Branch-and-Bound — Core Idea

*Used for optimization problems; prunes using bounding functions*

| Feature | Backtracking | Branch-and-Bound |
|---|---|---|
| **Goal** | Feasibility / all solutions | Optimal solution |
| **Pruning** | Constraint violations | Bounding function |
| **Search order** | DFS (depth-first) | BFS / Best-first |
| **Problems** | Decision problems | Optimization problems |
| **Bound** | Not used | Upper/Lower bounds |

## Branch-and-Bound Algorithm Steps

1. Compute upper bound for root node
2. Branch: split into subproblems (include/exclude)
3. Compute bound for each child node
4. Prune nodes whose bound ≤ best-so-far value
5. Repeat until all nodes processed or pruned

# Branch-and-Bound: 0/1 Knapsack Problem

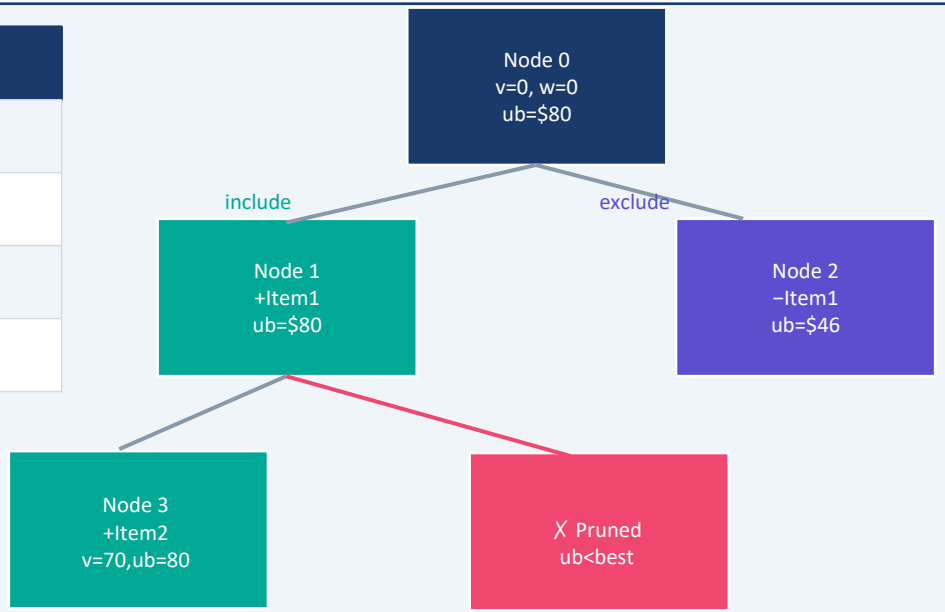
*Maximize total value subject to weight capacity constraint*

**Problem:** Given n items, each with weight $w_i$ and value $v_i$, select items to maximize $\Sigma v_i$ subject to $\Sigma w_i \leq W$ (capacity).

Items sorted by value/weight ratio (greedy order). Upper bound = value so far + fractional knapsack value of remaining items.

| Item | Weight | Value | v/w ratio |
|---|---|---|---|
| 1 | 2 | $40 | 20 |
| 2 | 5 | $30 | 6 |
| 3 | 10 | $50 | 5 |
| 4 | 5 | $10 | 2 |

**W = 16 (capacity)**




**Optimal: $70 (Items 1+2)**

UNIT 5

# Approximation Algorithms for NP-Hard Problems

Near-optimal solutions with guaranteed quality bounds

# Approximation Algorithms — Overview

*Trading optimality for efficiency when exact solutions are infeasible*

### Why Approximate?

NP-Hard problems have no known polynomial-time exact algorithm. For large inputs, we need fast near-optimal solutions.

### Approximation Ratio

An algorithm has ratio ρ(n) if:
$C / C^* \leq \rho(n)$
where C = approx solution cost, C* = optimal cost

### PTAS / FPTAS

PTAS: (1+ε)-approximation for any ε>0
FPTAS: polynomial in both n and 1/ε. Best known class.

## Knapsack Approximation Approaches

### Greedy by v/w Ratio

- Sort items by value/weight ratio (descending)
- Greedily include items while capacity allows
- NOT guaranteed optimal for 0/1 knapsack
- Can be ½-approximation in some formulations

### Dynamic Programming + Scaling (FPTAS)

- Scale down values: $v'_i = \lfloor v_i / K \rfloor$, $K = \varepsilon \cdot v_{max} / n$
- Run DP on scaled values — polynomial time
- Result is within (1+ε) of optimal
- Time: $O(n^3/\varepsilon)$, fully polynomial scheme

# Knapsack Approximation: Worked Example

*Greedy 2-approximation for the Fractional Knapsack relaxation*

| Item | Weight | Value | v/w | Greedy? |
|---|---|---|---|---|
| 1 | 2 | 40 | 20 | ✓ |
| 2 | 5 | 30 | 6 | ✓ |
| 3 | 10 | 50 | 5 | ✗ (full) |
| 4 | 5 | 10 | 2 | ✗ |

**Greedy Result**

**$70**

Items 1+2 selected
(Weight used: 7/16)

**Optimal (B&B)**

**$70**

Items 1+2 also optimal here (greedy succeeds)

**Approx Ratio**

**≥ ½**

Worst case guarantee
(FPTAS gives 1+ε)

**Key Insight: max(greedy_value, best_single_item_value) ≥ ½ · OPT**

# Strategies: Backtracking vs B&B vs Approximation

*Choosing the right approach to cope with NP-Hard problems*

| Criterion | Backtracking | Branch-and-Bound | Approximation |
|---|---|---|---|
| **Problem type** | Feasibility / Decision | Optimization | Optimization (NP-Hard) |
| **Solution quality** | Exact / all solutions | Exact optimal | Near-optimal (ρ-approx) |
| **Time complexity** | Exponential worst case | Exponential worst case | Polynomial |
| **Pruning basis** | Constraint violation | Bounding functions | N/A (no pruning) |
| **Completeness** | Complete search | Complete search | Incomplete (approx) |
| **Best for** | Small n, constraint satisfaction | Optimization, moderate n | Large n, NP-Hard problems |

**Quick Guide:** Proving NP-Completeness → Polynomial Reduction | Small input → Backtracking / B&B | Large input → Approximation | Need guarantee → FPTAS

UNIT 5 — KEY TAKEAWAYS

# Summary

## Decision Trees

Provide lower bounds for comparison algorithms. Sorting requires Ω(n log n) comparisons.

## P, NP, NP-Complete

P⊆NP. NP-Complete problems (SAT, TSP, Knapsack) likely require exponential time.

## Backtracking

State-space tree search with pruning. Used for N-Queens, Subset-Sum. Avoids exhaustive search.

## Branch-and-Bound

Optimization via best-first search with bounding. Efficiently solves 0/1 Knapsack.

## Approximation Algorithms

Polynomial-time algorithms with provable quality guarantees (ρ-approximation, FPTAS).

## Reference

Levitin, Anany.
Introduction to the Design and Analysis of Algorithms
3rd Ed. Pearson, 2017
Chapters 10, 11, 12