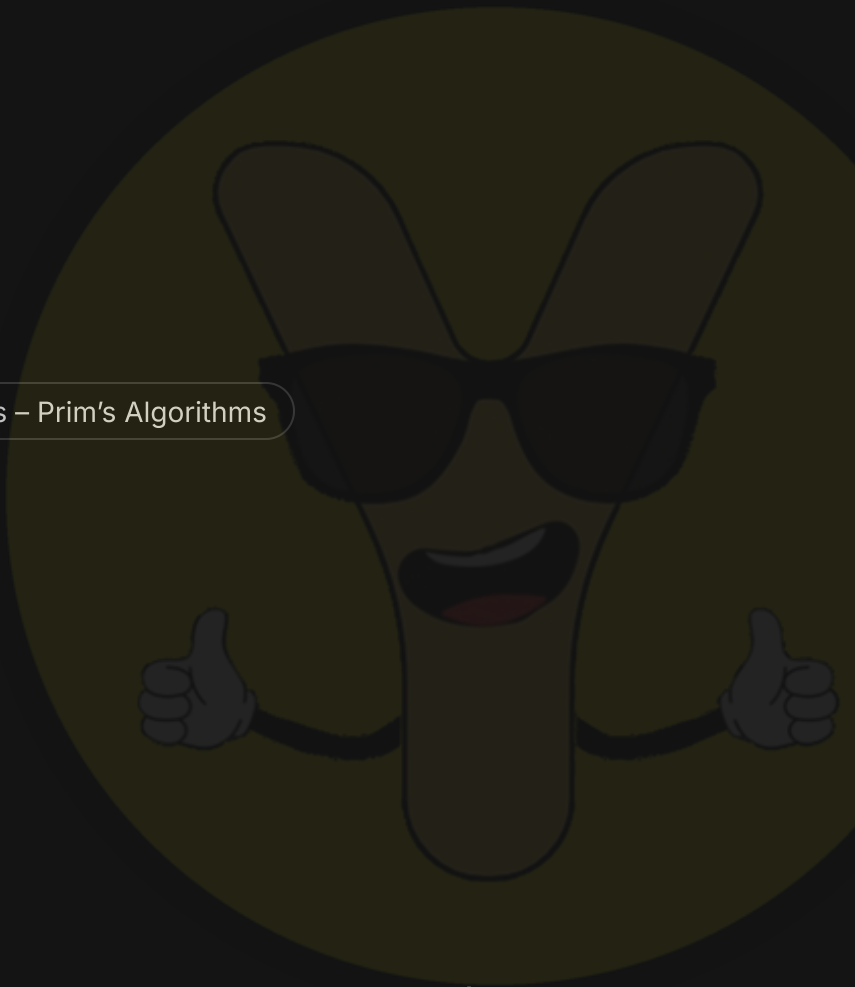
AKTU · 3RD YEAR · SEMESTER 5 · BCS503

Design and Analysis of Algorithm

Unit 3 — Divide and Conquer & Greedy Methods

- Divide and Conquer with Examples Such as Sorting
- Matrix Multiplication
- Convex Hull
- Searching
- Greedy Methods with Examples Such as Optimal Reliability Allocation
- Knapsack
- Minimum Spanning Trees – Prim's Algorithms
- Minimum Spanning Trees – Kruskal's Algorithms
- Single Source Shortest Paths - Dijkstra's Algorithms
- Single Source Shortest Paths - Bellman Ford Algorithms



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Divide and Conquer Strategy

DIVIDE AND CONQUER PARADIGM

An algorithm design technique that recursively breaks a problem into smaller subproblems of the same type, solves them independently, and combines their solutions to solve the original problem.

- Divide: Break the main problem of size n into a smaller subproblems of size n/b.
- Conquer: Solve each subproblem recursively. If subproblem size is small, solve directly (base case).
- Combine: Merge subproblem solutions into the solution for the original problem.
- General Recurrence: $T(n) = aT(n/b) + f(n)$, typically solved using Master Theorem.

Sorting via Divide and Conquer

Merge Sort and Quick Sort use Divide and Conquer to sort elements efficiently.

- Merge Sort
  - Divides array into two equal halves: $T(n) = 2T(n/2) + O(n)$.
  - Conquers by recursive sorting; combines using a two-pointer merge.
  - Time Complexity: $O(n \log n)$ in all cases. Space Complexity: $O(n)$.
- Quick Sort
  - Partitions array around a pivot: elements $\leq$ pivot on left, $\geq$ pivot on right.
  - Recurrence: Best/Average $T(n) = 2T(n/2) + O(n)$, Worst $T(n) = T(n-1) + O(n)$.
  - Time Complexity: Average $O(n \log n)$, Worst $O(n^2)$. In-place space: $O(\log n)$.

Strassen's Matrix Multiplication

Reduces scalar multiplications for matrix multiplication using Divide and Conquer.

- Standard matrix multiplication of two $n \times n$ matrices takes $O(n^3)$ time.
- Naïve Divide & Conquer splits matrices into 4 sub-blocks: $T(n) = 8T(n/2) + O(n^2) = O(n^3)$.
- Strassen's method computes only 7 matrix multiplications (M_1 to M_7) instead of 8.
- Recurrence Relation: $T(n) = 7T(n/2) + O(n^2)$.
- Time Complexity: $O(n^{\log_2 7}) \approx O(n^{2.81})$.
- Drawbacks: Higher space overhead, numerical instability, inefficient for small n.

Convex Hull: Divide and Conquer

Finds the smallest convex polygon enclosing a given set of 2D points.

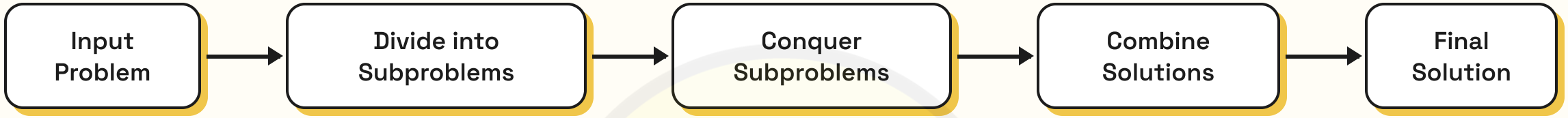
- Problem: Given n points, find the subset of points forming the bounding convex polygon.
- Divide Step: Sort points by X-coordinate and split into left set L and right set R.
- Conquer Step: Recursively compute the convex hull for L and R.
- Combine Step: Find upper and lower common tangents to merge hulls in $O(n)$ time.
- Recurrence: $T(n) = 2T(n/2) + O(n)$, yielding an overall time complexity of $O(n \log n)$.
- Applications: Computational geometry, GIS mapping, robot motion planning.

Searching: Binary Search Analysis

Locates an element in a sorted array by dividing search space in half.

- Prerequisite: Input array must be sorted in non-decreasing order.
- Algorithm Steps: Compare key with middle element $A[mid]$. If equal, return index.
- Divide Step: If $key < A[mid]$, search left half; if $key > A[mid]$, search right half.
- Recurrence Relation: $T(n) = T(n/2) + O(1)$.
- Time Complexity: Best Case $O(1)$, Average/Worst Case $O(\log_2 n)$.
- Space Complexity: Iterative implementation takes $O(1)$; recursive call stack takes $O(\log n)$.

Divide and Conquer Execution Flow



Workflow of Divide and Conquer algorithmic paradigm.

Greedy Method Paradigm

GREEDY CHOICE STRATEGY

An algorithm design technique that builds up a solution piece by piece, always choosing the next piece that offers the most immediate and local benefit.

- Greedy Choice Property: A globally optimal solution can be reached by making locally optimal choices.
- Optimal Substructure: An optimal solution to the problem contains optimal solutions to subproblems.
- Control Abstraction: Select candidate, check feasibility, include in solution, check optimization goal.
- Limitation: Never backtracks or re-evaluates past choices; may fail for problems like 0/1 Knapsack.

Optimal Reliability Allocation

Maximizes system reliability subject to cost constraints using a Greedy approach.

- Objective: Maximize total reliability by adding redundant components in parallel stages.
- System Reliability Formula: $R = \prod_{i=1}^{k} R_i(r_i)$, where r_i is component count at stage i.
- Greedy Strategy: At each iteration, allocate an extra component to the stage with highest relative gain.
- Cost Constraint: Total allocation cost $\sum(c_i \times r_i) \leq C_{max}$.
- Algorithm: Incrementally add redundancy to the bottleneck stage until cost limit is reached.
- Applications: Aerospace systems design, reliable network routing, fault-tolerant servers.

Fractional Knapsack Problem

Given items with weights W = [10, 20, 30] and values V = [60, 100, 120], fill a knapsack of capacity C = 50 to maximize value.

1. Calculate Value-to-Weight ratios (V/W): Item 1 = 60/10 = 6, Item 2 = 100/20 = 5, Item 3 = 120/30 = 4.
2. Sort items descending by ratio: Item 1 (ratio 6), Item 2 (ratio 5), Item 3 (ratio 4).
3. Take full Item 1: Weight = 10, Value = 60, Remaining Capacity = 40.
4. Take full Item 2: Weight = 20, Value = 100, Remaining Capacity = 20.
5. Take fraction of Item 3: Take fraction 20/30 of Item 3. Value = (20/30) * 120 = 80.

Maximum Total Value = 60 + 100 + 80 = 240. Time Complexity: O(n log n).

Minimum Spanning Tree (MST)

MINIMUM SPANNING TREE

A connected, undirected, edge-weighted graph's subgraph that connects all vertices together with the minimum possible total edge weight and no cycles.

- For a graph with V vertices, any MST contains exactly $V - 1$ edges.
- Adding any edge to an MST creates a unique simple cycle (Cycle Property).
- Cut Property: For any cut of graph, the minimum weight edge crossing cut belongs to MST.
- Standard Algorithms: Prim's Algorithm (vertex-based) and Kruskal's Algorithm (edge-based).

Prim's Algorithm for MST

Grows a single Minimum Spanning Tree vertex by vertex greedily.

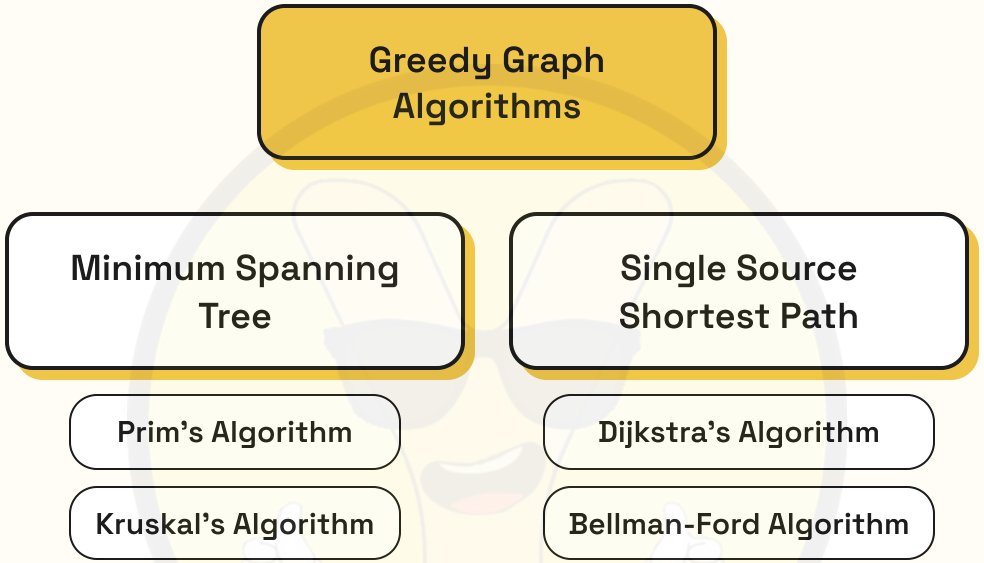
- Initialization: Start from an arbitrary root vertex; set distances of all other vertices to ∞.
- Greedy Choice: Pick non-tree vertex u with minimum weight edge connecting to current tree.
- Update Step: Add u to MST and relax edges to all adjacent non-tree vertices.
- Termination: Repeat until all V vertices are included in the MST tree set.
- Time Complexity: $O(V^2)$ using Adjacency Matrix; $O(E \log V)$ using Min-Heap and Adjacency List.
- Best Suited For: Dense graphs where $E \approx V^2$.

Kruskal's Algorithm for MST

Builds MST forest by selecting smallest edges that do not form a cycle.

- Step 1: Sort all edges in non-decreasing order of weight.
- Step 2: Initialize each vertex in its own set using Disjoint Set Union (DSU).
- Step 3: Process edges in sorted order; if edge (u, v) connects distinct sets, add to MST.
- Cycle Prevention: If $find(u) == find(v)$, reject edge to avoid forming a cycle.
- Time Complexity: Edge sorting takes $O(E \log E)$; DSU takes $O(E\alpha(V))$. Total: $O(E \log V)$.
- Best Suited For: Sparse graphs where $E \ll V^2$.

Greedy Optimization Algorithms



Classification of standard greedy graph algorithms.

Dijkstra's Algorithm (SSSP)

Computes shortest paths from a source to all vertices in non-negative weighted graphs.

- Greedy Choice: Unvisited vertex with smallest tentative distance is extracted next.
- Relaxation Formula: If $dist[u] + weight(u, v) < dist[v]$, update $dist[v] = dist[u] + weight(u, v)$.
- Precondition: Fails if graph contains negative weight edges or cycles.
- Data Structures: Priority Queue (Min-Heap) and Adjacency List.
- Time Complexity: $O((V + E) \log V)$ with Min-Heap, $O(V^2)$ with plain array.
- Space Complexity: $O(V)$ to store distances and priority queue.

Bellman-Ford Algorithm (SSSP)

Computes single-source shortest paths in graphs with arbitrary edge weights.

- Core Concept: Relaxes all E edges repeatedly $V - 1$ times.
- Relaxation Step: For each edge (u, v) with weight w, update $dist[v] = \min(dist[v], dist[u] + w)$.
- Negative Cycle Detection: Run 1 additional pass. If any distance decreases, a negative cycle exists.
- Time Complexity: $O(V \times E)$ in all cases.
- Space Complexity: $O(V)$ auxiliary space for distance tracking.
- Advantage: Handles negative edge weights correctly and detects negative cycles.

Comparison: Dijkstra vs Bellman-Ford

Feature	Dijkstra's Algorithm	Bellman-Ford Algorithm
Design Paradigm	Greedy Approach	Dynamic Programming
Edge Weight Constraint	Non-negative weights only	Positive and negative weights
Negative Cycle Handling	Fails / Undetermined	Detects negative weight cycles
Time Complexity	$O((V + E) \log V)$ with Heap	$O(V \cdot E)$
Space Complexity	$O(V)$	$O(V)$
Best Suited For	Sparse/Dense non-negative graphs	Graphs with negative edge weights
Execution Strategy	Extracts min vertex iteratively	Relaxes all edges $V - 1$ times

Comparison: Fractional vs 0/1 Knapsack

Property	Fractional Knapsack	0/1 Knapsack
Algorithmic Strategy	Greedy Method	Dynamic Programming
Item Splitting	Allowed (take fraction of item)	Not allowed (take whole item or 0)
Greedy Choice Guarantee	Always yields optimal solution	Greedy approach fails
Selection Criteria	Value-to-weight ratio (v_i/w_i)	Optimal subproblem combinations
Time Complexity	$O(n \log n)$ (due to sorting)	$O(n \cdot W)$ pseudo-polynomial
Space Complexity	$O(1)$ auxiliary space	$O(n \cdot W)$ table space

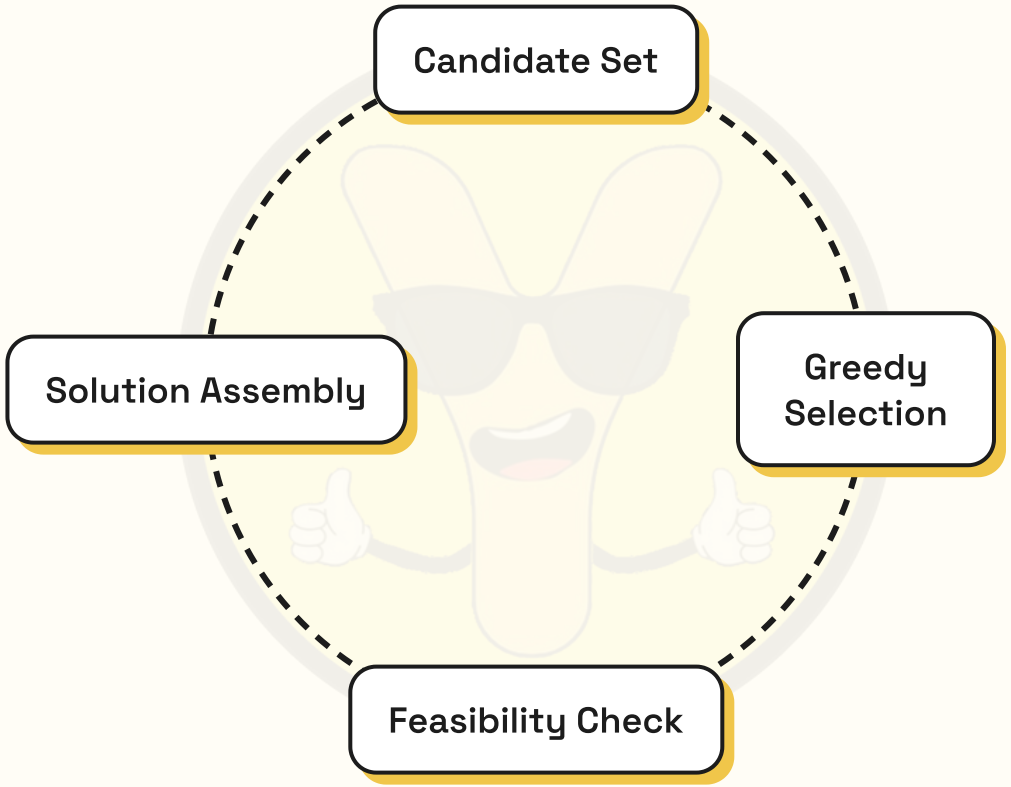
Worked Example: Prim vs Kruskal MST

Graph with vertices {A,B,C,D} and weighted edges: (A-B:1), (B-C:3), (A-C:4), (C-D:2), (B-D:5). Compute total MST weight.

1. Kruskal Strategy: Sort edges by weight → (A-B:1), (C-D:2), (B-C:3), (A-C:4), (B-D:5).
2. Select (A-B:1): No cycle formed. Add to MST. MST weight = 1.
3. Select (C-D:2): No cycle formed. Add to MST. MST weight = 1 + 2 = 3.
4. Select (B-C:3): No cycle formed. Add to MST. MST weight = 3 + 3 = 6.
5. Vertices count = 4, edges selected = 3 (which is V-1). Stop algorithm.
6. Prim Strategy (start A): Pick min edge from A → (A-B:1). Next min edge from {A,B} → (B-C:3). Next min edge from {A,B,C} → (C-D:2).

Total MST Weight = 1 + 2 + 3 = 6. Both algorithms yield identical total weight.

Greedy Choice Optimization Loop



Iterative state cycle of greedy optimization algorithms.

Exam corner

- **Q1** Explain the Divide and Conquer strategy. Solve the recurrence T(n) = 7T(n/2) + O(n^2) for Strassen's Matrix Multiplication.
- **Q2** Differentiate between Quick Sort and Merge Sort. Write their recurrence relations and compare worst-case time complexities.
- **Q3** Formulate the Fractional Knapsack problem. Explain why the Greedy approach works for Fractional Knapsack but fails for 0/1 Knapsack.
- **Q4** Write Prim's algorithm for Minimum Spanning Tree. Trace it on a given graph and derive its time complexity.
- **Q5** State Kruskal's algorithm using Disjoint Set Union (DSU). Explain how cycle detection is performed with Find-Union operations.
- **Q6** Compare Dijkstra's and Bellman-Ford algorithms for Single Source Shortest Paths. Why does Dijkstra fail for negative edge weights?
- **Q7** Describe the Convex Hull problem using the Divide and Conquer algorithm. Derive its overall time complexity.

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

