

Introduction to Problem Solving Techniques

Introduction to Problem Solving Techniques

Study guide – concise, organized, and easy to read

1. What Is Problem Solving?

Aspect	Description
Definition	The systematic process of identifying a gap between the current state and a desired goal, then devising and executing a plan to bridge that gap.
Goal	Produce a workable solution that satisfies constraints (time, resources, quality, etc.).
Scope	Applies to math, engineering, programming, business, everyday life, and scientific research.

2. General Problem Solving Cycle

1. Understand the Problem

- Restate the problem in your own words.
- Identify known data, unknowns, and constraints.
- Clarify the desired output.

2. Devise a Plan

- Choose an appropriate strategy (see Section 3).
- Break the problem into subproblems if needed.
- Sketch a high-level algorithm or flowchart.

3. Execute the Plan

- Carry out calculations, write code, build a prototype, etc.
- Keep notes of intermediate results.

4. Verify / Reflect

- Test the solution against the original requirements.
- Check edge cases, error handling, and efficiency.

Introduction to Problem Solving Techniques

- Reflect on what worked and what could be improved.

Tip: Loop back to step 2 if verification fails—refine the plan rather than forcing a broken solution.

3. Core Techniques & When to Use Them

Technique	Key Idea	Typical Use Cases	Strengths	Weaknesses
Trial and Error	Test possible solutions until one works.	Simple puzzles, debugging small scripts.	Quick start, no heavy analysis needed.	Inefficient for large search spaces.
Algorithmic/Procedural	Follow a step-by-step method (e.g., sorting, searching).	Programming, data processing.	Predictable performance, reusable.	Requires a known algorithm.
Divide & Conquer	Split problem into independent subproblems, solve each, combine results.	Merge sort, matrix multiplication, large-scale projects.	Reduces complexity, parallelizable.	Overhead of merging; not always applicable.
Dynamic Programming	Store solutions of overlapping subproblems to avoid recomputation.	Shortest path, knapsack, sequence alignment.	Optimal for problems with optimal substructure.	Needs extra memory; may be hard to identify subproblems.
Backtracking	Build solutions incrementally; abandon a partial solution (“backtrack”) as soon as it violates constraints.	Sudoku, N-Queens, constraint satisfaction.	Guarantees finding a solution if one exists.	Can be exponential time if pruning is weak.

Introduction to Problem Solving Techniques

Technique	Key Idea	Typical Use? Cases	Strengths	Weaknesses
Greedy Methods	Make the locally optimal choice at each step.	Huffman coding, activity selection.	Simple, fast, often optimal for specific problems.	May miss global optimum.
Heuristics & Approximation	Use “rules of thumb” or relaxed criteria to get a good?enough solution quickly.	Route planning (A*), NP?hard problems.	Scalable, practical for large instances.	No guarantee of optimality.
Mathematical Modeling	Translate the problem into equations, inequalities, or statistical models.	Optimization, physics simulations, economics.	Enables analytical solutions, leverages existing theory.	Model may be oversimplified or hard to solve.
Simulation	Replicate the system			