Mental Models for Algorithmic Problem Solving

---

1. "Invert, Always Invert" (Munger/Polya)

- **Flip the question**: Instead of "how do I build a valid X?", ask "how do I eliminate all invalid Xs?"
- **Example**: Count pairs *not* divisible by `k` instead of those that *are*.

- **Application**: Proof by contradiction, complementary counting, greedy covering.

---

2. "Can I Reduce the Search Space?"

- **Hashing**: Map input to a smaller, more manageable domain (`mod k`, frequency buckets).
- **Sliding window**: If the structure is local, can we discard older state?
- **Bucket Sort / Counting**: When inputs are bounded (like `0 <= x <= 100`).

---

3. "Can I Sort the Input?"

- Sorting often unlocks greedy solutions or ordered reasoning:
  - Detect adjacent duplicates

  - Use two pointers

  - Sweep line (for intervals)

  - Binary search

Heuristic: If the problem asks about **min, max, range, median, closest, furthest**, sorting is likely helpful.

---

4. "Two Pointers / Binary Search / Sliding Window"

- Use when the data is **ordered**, **contiguous**, or can be made so.
- Common in strings, arrays, intervals, and prefix problems.

Binary search variants: - On **index** (standard) - On **answer** (e.g. minimum valid `k`, or maximum satisfying constraint)

---

5. “Geometry, Not Just Algebra”

- Treat a 2D grid or matrix like a plane:
  - Flip rows/columns = mirror symmetries
  - Diagonals = i + j and i - j invariants
  - Rotate/reflection invariance

Heuristic: If you’re dealing with a grid, consider **quadrants**, **axes**, **symmetry**.

---

6. “Can I Precompute or Reuse?”

- **Prefix sums / differences**

- **Sparse tables / RMQ**

- **Memoisation / DP**

Heuristic: If a subproblem is repeated or nested within a larger one, **memoise** or **tabulate**.

---

7. “Recursive or Divide-and-Conquer Structure?”

- Look for:
  - Problems naturally reducible into halves (QuickSort, MergeSort)
  - Balanced trees
  - “Find something in log n time”

Heuristic: If the input **size shrinks by half each step**, or **recursive dependencies** exist, try divide-and-conquer.

---

8. “Is There a Stack or Queue Structure Hidden in the Logic?”

- Monotonic stacks: for problems like “next greater”, “previous less”
- Min/max sliding windows
- DFS or BFS (esp. for graphs or grid traversal)

---

9. “Greedy?”

- If you can make a **local optimal decision that guarantees a global optimal**, consider greedy.
- Sort, process in order, always pick best remaining option.

Check: Does the problem have an **exchange argument**? (i.e., any non-greedy choice can be swapped with a greedy one without making things worse)

---

10. "Graph-ify It"

- Can the problem be modelled as:
  - Reachability (BFS/DFS)
  - Shortest path (Dijkstra/Bellman-Ford)
  - Cycles or components (Union-Find / Tarjan)
  - Topological sorting

Heuristic: If entities have **relationships, dependencies or networks**, this is probably a graph.

Bonus Models

"Transform the Problem"

- Change coordinates
- Map strings to integers (e.g. `ord c - ord 'a'`)
- Reverse the direction (e.g. simulate from goal state backwards)

"Watch the Constraints"

- If `n <= 20`: Brute force

- If `n <= 10^5`: `O(n log n)` or better

- If `n <= 10^9`: Binary search or mathematical trick

"Probabilistic / Expectation?"

- If you're asked to **minimise expected value** or **average cost**, think in terms of **linearity of expectation** or even **greedy + DP mix**.

A Sample Checklist You Could Use:

Before coding, ask: 1. **Can I reframe this problem (invert, graphify, transform)?** 2. **Can I preprocess something (prefix, sort, bucket)?** 3. **Is it recursive / has overlapping subproblems (DP)?** 4. **Is there a local optimum that implies a global one (greedy)?** 5. **Are the constraints screaming for a specific approach?** 6. **Does the problem look familiar**

(classic patterns)? 7. **Am I repeating work? Can I memoise or tabulate?** 8. **Can I find a way to binary search on the answer?**