Coding Interview Cheat Sheet

Problem Type → **Algorithm / Data Structure to Solve the Problem**

Graph / Grid

- Deepcopy Graph (or linked list) → Recursive graph traversal with visited dictionary
- Directed graph with prerequisites → Topological Sort (Kahn's Algorithm)
- Minimum cost to connect nodes (weighted graph) → Is this a normal spanning tree problem?
  - Yes → Minimum Spanning Tree (Kruskal's Algorithm)
  - No → Kruskal's Algorithm but use some imaginary nodes
- Shortest path between nodes → Is the graph weighted?
  - No → Breadth First Search
  - Yes → Are there negitive weights?
    - No → Dijkstra's Algorithm
    - Yes → Bellman Ford Algorithm
- Find minimum operations / steps / transformations → Breadth First Search
  - Breadth First Search → Need to optimize BFS for shortest path?
    - Yes → A* (BFS, but replace queue with heap)
    - No → Breadth First Search
- Tree → Level Order Traversal
  - Tree → Depth First Search
  - Tree → Preorder Inorder Postorder
- Counting Clusters of Nodes → Union Find data structure (disjoint sets)

Continuous Subarray

- Continuous Subarray → Count subarrays with one "occurrence"
  - Count subarrays with one "occurrence" → Look for each possible occurrence and count subarrays containing that occurrence
- Continuous Subarray → Prefix Sum
- Continuous Subarray → Sliding Window

Problem asking for sorting based on special criteria

- Problem asking for sorting based on special criteria → Language specific sorting tools
- Language specific sorting tools → Python: Pass key function that returns tuple with elements in order of priority

Branching decisions and looking for min/max

- Branching decisions and looking for min/max → Dynamic Programming
  - Dynamic Programming → Top Down Recrusion + Memoization
  - Dynamic Programming → Bottom Up

Subsets of an array

- Subsets of an array → Backtracking
- Subsets of an array → Python 'combinations' function

Range of possible answers that can divided into two sections

- Range of possible answers that can divided into two sections → Binary Search
- Boundary between halfs of array → Binary Search
- Better than linear search time → Binary Search
- Binary Search → bisect_left(val) Returns index of first item >= val
- Binary Search → bisect_right(val) Retruns index of first item > val

Looking at 2 elements at a time

- Looking at 2 elements at a time → Two Pointer Technique

Find closest elements between two sorted lists

- Find closest elements between two sorted lists → Two Pointer Technique

Nothing else works?

- Nothing else works? → Greedy Algorithm

Min / Max x elements of array

- Min / Max x elements of array → Sort the array
- Min / Max x elements of array → QuickSelect Algorithum
- Min / Max x elements of array → Heap

Check for Cycle in Linked List

- Check for Cycle in Linked List → Fast Pointer + Slow Pointer

Reduce Space Complexity

- Reduce Space Complexity → Overwrite the input

Compress val out of array

- Compress val out of array → List of <index, val> pairs, excluding one val

Problem Involves Prefixes

- Problem Involves Prefixes → PrefixTree (Trie)

Rolling Window of Events

- Rolling Window of Events → TreeMap (Linked HashMap, Orded Dictionary)

Find next element greater than each index (or prev element less than)

- Find next element greater than each index (or prev element less than) → Monotonic Stack (not a stack, created with a stack)

Need more pracitce? Schedule a mock coding interview with me

http://tiny.cc/e416wz

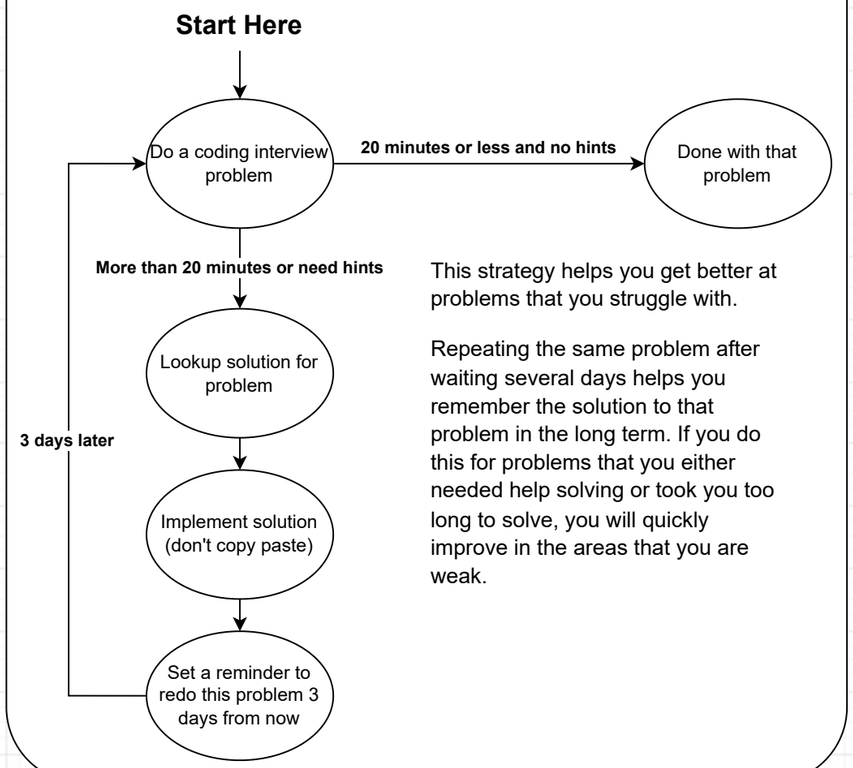
Coding Interview - Helpful Tips & Strategies

Helpful Python Features

If you're going to spend a lot of time preparing for coding interviews, I think it's worth while to learn Python. It's easy to learn, easy to write, and has a lot of helpful features that can make it much easier to solve problems. Below is a list of some helpful features in Python.

- **List comprehensions** can save time by using much shorter syntax to create a list
  - Tutorial: http://tinyurl.com/2uek6vxt
- The **combination** and **permutation functions** can save a lot of time by returning all the combinations or permuations for a given list of elements
  - Tutorial: http://tinyurl.com/3aan5kby
- **lru_cache** can be used to quickly optimize certain functions. It is often used to solve dynamic programming problems using a top down approach.
  - Tutorial: http://tiny.cc/opj0wz
- **bisect_left** and **bisect_right** are quick ways to binary serach a sorted list
  - Tutorial: http://tiny.cc/pwj0wz
- **Nested functions** can help make your code readable and maintainable
  - Tutorial: http://tiny.cc/jxj0wz
- You can easily **return multiple values** from a function as a tuple
  - Tutorial: http://tiny.cc/dyj0wz
- **Key functions** can be used to easily sort a list based on complex criteria
  - Tutorial: http://tiny.cc/dyj0wz

Spaced Repetition Strategy



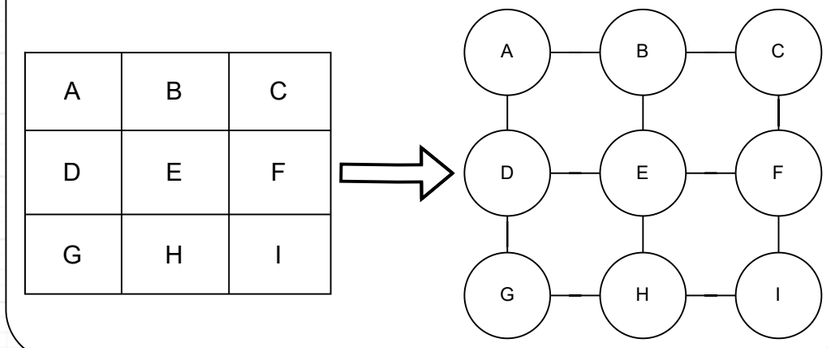
This strategy helps you get better at problems that you struggle with.

Repeating the same problem after waiting several days helps you remember the solution to that problem in the long term. If you do this for problems that you either needed help solving or took you too long to solve, you will quickly improve in the areas that you are weak.

Graphs and Grids

Many coding interview problems involve a grid. It can be helpful to think of a grid like a graph where each cell in the grid is a node with edges connecting it to neighboring cells.

Many techniques and algorithms that can be applied to a graph can also be applied to grids. Depth First Search, Breadth First Search, etc



Need more pracitce? Schedule a mock coding interview with me

http://tiny.cc/e416wz