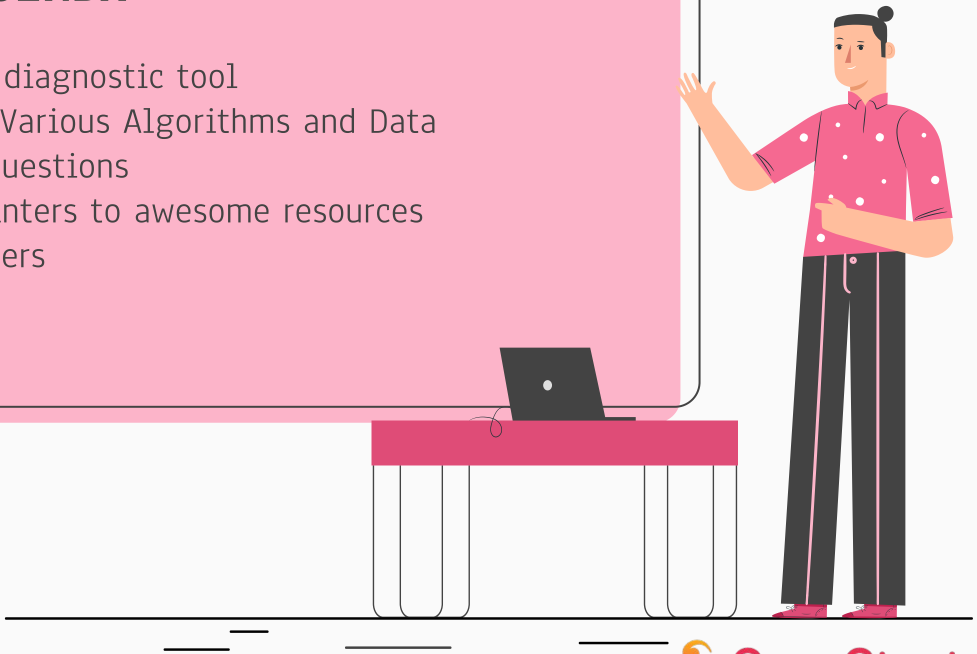
Common Coding Patterns in Technical Coding Screens

SureStart

AGENDA

1. Interactive: Coding diagnostic tool
2. Typical patterns in Various Algorithms and Data Structures related questions
3. Conclusion with pointers to awesome resources
4. Questions and Answers



DAY I

BINARY SEARCH

Binary search compares the target value to the middle element of the array. The array must be sorted in ascending or descending order. Practice it here or with sqrt(x).

Searching

Algorithm	Data Structure	Time Complexity		Space Complexity
		Average	Worst	Worst
Depth First Search (DFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Breadth First Search (BFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Binary search	Sorted array of n elements	O(log(n))	O(log(n))	O(1)
Linear (Brute Force)	Array	O(n)	O(n)	O(1)
Shortest path by Dijkstra, using a Min-heap as priority queue *	Graph with |V| vertices and |E| edges	O((|V| + |E|) log |V|)	O((|V| + |E|) log |V|)	O(|V|)
Shortest path by Dijkstra, using an unsorted array as priority queue *	Graph with |V| vertices and |E| edges	O(|V|^2)	O(|V|^2)	O(|V|)
Shortest path by Bellman-Ford *	Graph with |V| vertices and |E| edges	O(|V||E|)	O(|V||E|)	O(|V|)

1. **Rotated Sorted Arrays**: Search in Rotated Sorted Array, Minimum in Rotated Sorted Array
2. **Matrices**: Kth Smallest Element in a Sorted Matrix, Search a Matrix
3. **Data Stream**: Find Median in a Data Stream, Data Stream as Disjoint Interval
4. **Counting**: Count Negative Numbers
   (If you want to try harder problems - Count of Smaller Numbers After Self, Count of Range Sum)

* less likely to encounter in a coding interview

TIME & SPACE COMPLEXITY

- **Time complexity:** the amount of time taken by an algorithm to run as a function of the length of the input.
- **Space complexity:** the amount of memory taken by an algorithm to run as a function of the length of the input.
- The complexity can be analyzed as **worst case (usually), average case (sometimes) and best case (not useful).**
- **Analysis of loops**
- Review the average and worst case **complexity of common data structure operations and sorting algorithms**
  - Insertion into a sorted linked list is linear in the number of nodes.
  - Deletion from a balanced binary tree is logarithmic in the number of elements
- If you have time, review **Amortized Analysis**
  - Used for algorithms where an occasional operation is very slow, but most of the other operations are faster.
  - Amortization (e.g. "this operation is amortized constant time") is used to discuss average time/space complexity in situations where operations are sometimes much more expensive than usual, but only rarely.
  - E.g. growing an array by doubling from N to 2N requires copying N items, but it takes at least N/2 insertions since the previous doubling to get to the point where regrowth is again necessary.
  - Check out "amortized analysis" examples.

HOW TO SPEAK O(N)

- “This algorithm takes O(n) time and O(n log n) space, where n is the number of …”
    - “… oh of n time and oh of n log n space, where …”
    - “O(n^2)” is pronounced “oh of n squared”
- Shorthand
    - “O(1)” is sometimes pronounced “constant”
    - “O(n)” is sometimes pronounced “linear”
    - “O(log n)” is sometimes pronounced “logarithmic”
    - “O(n^2) is sometimes pronounced “quadratic”

DAY 2

SureStart

SORTING

Sorting - we need to rearrange a given array or list elements according to a comparison operator on the elements.

Frequently asked sorting algorithms: Merge, Quick, Insertion sort

Things to note: stability, in-place-ness, element type, whether there are any duplicates, size of the array/list.

1. **Duplicates**: value that appears at least twice, remove duplicates from sorted array *
2. **Anagrams**: group anagrams, valid anagrams
3. **Linked Lists Sorting**: Sort a Linked List *
4. **In-place**: sort colors, find and replace

* *Hint: use two pointers*

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

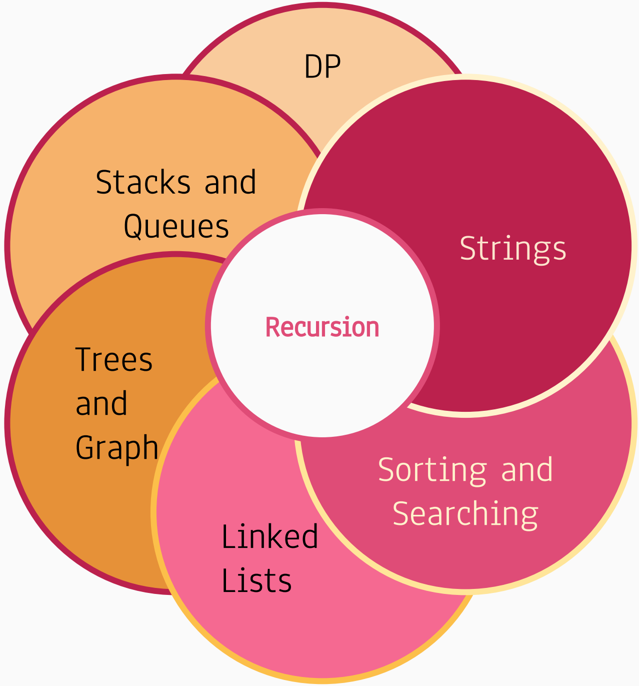
DAY 3

SureStart

Knowing Recursion is very important!

It overlaps with several categories of problem that you can get asked:

- Linked lists? Print a linked list in reverse order.
- Strings? Determine if a string is a palindrome.
- Tree traversals? DFS/BFS
- ... And even a simple algorithm like finding a factorial of a number.
- Analysis of recursive algorithms



RECURSION

1. **Iteration:** Any problem that can be solved via loops can be solved via recursion
   E.g.; print a linked list in reverse order.
2. **Subproblems:** Problems where each sub-problem is solved using the same decision-making function
   E.g.: Fibonacci series, Tower of Hanoi
3. **Selection problems:** Going through all the possible solutions and selecting the ones which match a particular condition;
   - Optimized by validating as we go/**backtracking**
   - E.g.: Combination problems, Knapsack, N-Queens, Word break
4. **Ordering:** Similar to selection but order matters. E.g. Permutations
5. **Divide & Conquer:** Solve the problem for each half of the input and easily combine the result
   - Common in searching, sorting, trees
   - E.g.: Merge Sort, Valid Parentheses and its variants
6. **Depth First Search (or BFS)**
   - Critical to remember the path to a particular node --- not just how to search nodes
   - DFS is often core to other larger problems, e.g. knight's path on a chessboard

Watch this video: https://www.youtube.com/watch?v=BibDrTCGXRM
Read this article: https://medium.com/swlh/the-six-core-patterns-of-recursion-b1b4ea878f27

DAY 4

STRINGS

1. Strings are a sequence of characters (ASCII or Unicode). Know related functions in your chosen language:
   - Get the ASCII value of a character
   - Convert an ASCII value into a character
   - Convert a digit character into its integer value (ie. convert "5" into 5)

2. **Using a Length-256 Integer Array:** Index -> ASCII value of character; and value -> number of times char occurs is in string
   - Anagrams
   - Sorting the characters in a string
   - Longest substring without a repeating character

3. String Math:
   - Convert larger numbers into strings
   - Convert strings to binary
   - String to integer
   - Compare version numbers

Watch this video: https://www.youtube.com/watch?v=9clnwaqOU2E
Read this article: https://www.byte-by-byte.com/strings/

STRINGS

** 4. **Using two pointers**

- Remove duplicates
- Is string a palindrome
- Reverse words in a string

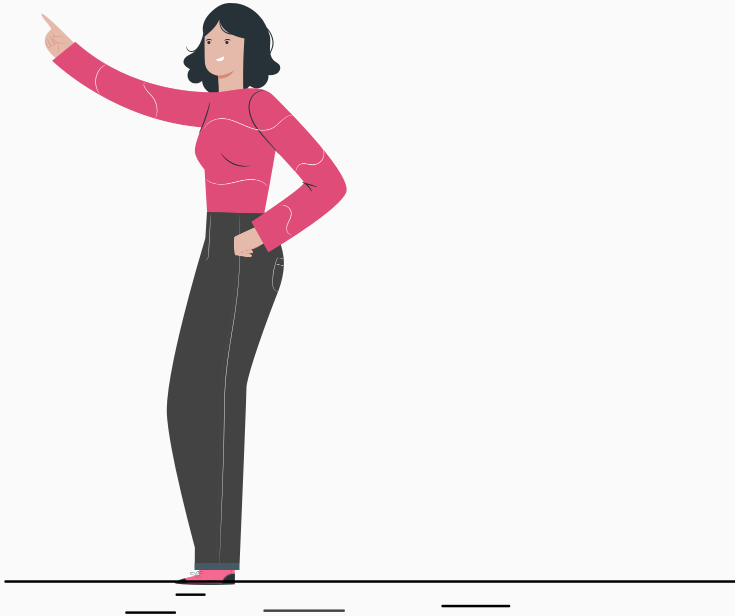
** 5. **Sliding windows:** special case of the two pointer pattern

- Longest substring without-repeating-char
- Find all anagrams in a String
- Minimum window substring
- Substring with concatenation of all words

6. **String comparisons**

- Check substring present given string
- Longest common substring
- Edit distance
- Regex matching

** Used a lot in interview questions



DAY 5

HASH TABLES

Hash Table Data Structure Review:

- **What is a hash table?** A data structure that can map keys to values. A hash table uses a "hash function" to compute an index (a hash value) into an array of buckets, from which the desired value can be found.
- **Properties of hash functions:** very fast computation; one way non-reversible functions; output does not reveal information on input, and others.
- **How are hash tables implemented? What is hash collision?**
- Know the hash table functionality in your language of choice (in Python, it is implemented as Dictionary)
- Design Hashset
- Design Hashmap

Problems involving hashing

1. Related to numbers
- First non-repeating integer in an array
- Find two numbers that add up to "n"

2. Related to lists
- Detect if a list is cyclic using hashmap
- Remove duplicates from list ("dedup")
- Union and intersection of 2 linked lists using hashing

3. Related to arrays
- Find the kth most frequent number in the array
- Check if two arrays are equal or not
- Find symmetric pairs in an array
- Find two pairs in an array such that a+b = c+d
- Check if an array is a subarray of another array

SureStart

LEETCODE CHALLENGES FOR HASH TABLES

Related to Numbers

- Happy Number
- N-Repeated Element
- Next Greater Element I

Related to Lists

- Linked List Cycle
- Intersection of Two Linked Lists
- Minimum Index Sum of Two Lists

Related to Arrays

- Intersection of Two Arrays
- Find All Disappeared Numbers
- Degree of an Array

DAY 6

ARRAYS/LISTS

1. **Array search:**

Find duplicates: Find all the duplicates in the array of integers

2. **Subarray selection:**

- Consecutive array: Given an unsorted array, find the length of the longest sequence of consecutive numbers
- Count of range sum: Given an array, return the number of range sums that lie in [lower, upper] inclusive

3. **Array Reordering:**

Permutations: Return all permutations of a given list

4. **Exploring two arrays in tandem:**

- Merge arrays: Given k sorted arrays, merge them into a single sorted array
- Merge arrays without additional memory: Given 2 sorted arrays, A and B, where A is long enough to hold the contents of A and B, write a function to copy the contents of B into A without using any buffer
- Median of arrays: Find the median of two sorted arrays
- Anagrams: Given two strings, write a function to determine whether they are anagrams

5. Matrices related problems:

- Matrix product:
  Given a matrix, find the path from top left to bottom right with the greatest product by moving only down and right

- Zero Matrix:
  Given a boolean matrix, update it so that if any cell is true, all the cells in that row and column are true

- Diagonals:
  Sort the values of each of the matrix diagonals

- Matrix Search:
  Given an n x m array where all rows and columns are in sorted order, write a function to determine whether the array contains an element x

DAY 7

LINKED LISTS

Know the list structure variations:

1. Singly vs. doubly linked; Circular lists, use of sentinel nodes
2. List reversals:

   Reverse a singly linked list in O(1) space
3. Node deletions:
   - Delete an element of a singly linked list, given only a pointer to that element
   - Remove Nth Node From End of List
   - Remove duplicates letters
4. Cycles detections:
   - Cycle detection in singly linked lists: “tortoise and hare” pointers or Floyd’s algorithm
   - Find the starting point of the loop
5. Split/merge/copy:
   - Split a linked list
   - Merge N sorted singly linked lists into 1 sorted list.
   - Deep copy a list with random pointers
6. Transform to tree:

   Convert a binary tree into a doubly linked list

DAY 8

STACKS AND QUEUES

1. **Implementation related:**
    - Implement stacks with arrays, then with linked lists. Ditto for queues.
    - Implement N stacks using one array
    - Implement a stack from a queue
    - Implement stacks supporting an extra max operation. (same for queues)
    - Implement Stack using Queues
    - Implement Queue using Stacks
    - Design a Circular Queue
    - Design Circular Deque
    - Design a Min Stack

2. **Structure manipulations:**
    - Reverse a stack or queue in each of the above implementations
    - Sort a stack
    - Flatten a Nested List
    - Reorder List

3. Puzzles (highly constrained problems):

- Sort a queue or stack.
- Implement a stack using 2 queues. Implement a queue using 2 stacks

4. Typical uses of stacks:

- Evaluating nested function calls
- Backtracking search (e.g solving a maze)
- Example: postfix calculator
- Binary Tree Postorder Traversal
- N-ary Tree Postorder Traversal

5. Typical uses of queues:

- "First come, first served" event handling
- Level-order tree traversal. Breadth-first graph traversal.
- Example: Interesting way to count in binary using a queue of strings

- Use both Stacks and Queues

DAY 9

TREES

** 1. **Binary tree versus binary search tree (BST)** Is given tree a BST?

2. **Depth first tree traversals:**
- In order: Usually used for binary tree search; review Binary Tree Inorder Traversal
- Pre-order; review Binary Tree Preorder Traversal
  - Create a copy of a binary tree
  - Compare two trees
- Post-order; review Binary Tree Postorder Traversal
- **Traversal comfort diagnostic:** Construct binary tree from in-order and post-order traversal

** 3. Level order traversal for breadth first trees; useful when you have to examine a tree level by level
- **Related questions:**
  - Binary Tree Level Order Traversal
  - Maximum level sum of a binary-tree
  - Average of levels in binary tree
  - Print leftmost and rightmost nodes of a binary-tree

4. **Request for additional data beyond tree traversal:**
    - The required answer deals with the node's depth (or # of levels).
    - The answer may require passing information about nodes between nodes (e.g. build a path or string)

Examples:

- Maximum Depth of a Binary Tree – Store depth and node.
- Merge Two Binary Trees – Store both nodes.
- Binary Tree Longest Consecutive Sequence – Store node and current length.
- Find the Largest Value in Each Tree Row – Store node and depth. Keep track of nodes in each depth using a dictionary.
- Smallest String Starting from Leaf – Store node and string.
- Binary Tree Paths – Store node and current path.
- Max Level Sum of a Binary Tree – Store node + depth. Keep track of nodes at each depth via a dictionary.

DAY 10

GRAPHS

1. Graph representations:
- **Adjacency matrix:** V X V matrix with (i, j) = 1 or 0, depending on whether vertices i and j are adjacent or not.
- **Adjacency list:** an array of lists, where each list corresponds to a vertex and represents its adjacent vertices
- Number of Provinces

2. **Graph search (and time complexity, when it is usually used)**
- **DFS (Depth-first search):** Used for finding Shortest path
- **BFS (Breadth-first search):** Used for finding if path exists

3. **Graph Cycles:**
- Detect cycle in a directed graph or cycle in an undirected graph

Related problems: Is Graph Bipartite?, Is graph a tree?, Number of connected components, Clone Graph, Course Schedule, Pacific Atlantic Water Flow, Number of Islands, Longest Consecutive Sequence

Less common for interview questions:
- Topological sort: only works for Directed Acyclic Graph (DAG), i.e., directed edges and no cycles
- Djikstra's algorithm: works for directed and undirected graphs as long as they do not have negative weight on any edge; usually implemented with a priority queue. (Try: Cheapest Flights Within K Stops)

THANK YOU!

AND GOOD LUCK FOR THE NEXT SEMESTER.

SureStart

RESOURCES

An useful overview: 30-day guide to the technical interview process

Some books to help with interview prep:

- Cracking the Coding Interview
- Problem Solving with Algorithms and Data Structures Using Python

Some online resources:

- Khan Academy, Algorithms
- Interviewcake.com
- 30dayscoding Data Structures and Algorithms Guide
- 50 Common Coding Interview Questions
- 75 practice questions on various data structures and algorithms

Websites to practice coding:

- Leetcode resources for coding
- The HackerRank Interview Preparation Kit

Time complexity review:

- Big-O Cheat Sheet
- Big-O Explained

Watch mock interviews with interviewing.io and sign up for your mock Algorithms and Data Structures interview!

SureStart

ADDITIONAL TREE QUESTION PATTERNS

- **Searching:** https://leetcode.com/problems/search-in-a-binary-search-tree/
- **Ancestor problem:** https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/
- **Root to leaf path problems:** https://leetcode.com/problems/binary-tree-paths/
- **Leaves related problem:** https://leetcode.com/problems/sum-of-left-leaves/
- **Level order traversal:** https://leetcode.com/problems/average-of-levels-in-binary-tree/
- **Node deletion:** https://leetcode.com/problems/delete-node-in-a-bst/
- **Tree construction:** https://leetcode.com/problems/construct-binary-search-tree-from-preorder-traversal/
- **Distance between two nodes:** https://leetcode.com/problems/minimum-distance-between-bst-nodes/
- **Check binary tree:** https://leetcode.com/problems/check-completeness-of-a-binary-tree/
- **Depth problem:** https://leetcode.com/problems/maximum-depth-of-binary-tree/

More problems related to each pattern and additional patterns:
https://leetcode.com/discuss/study-guide/1337373/Tree-question-pattern-oror2021-placement