Interview Prep - Study Plan

# General Interview Prep

1. **Go over 1331 Notes** (read the most important sections that you’re unfamiliar with)
   1. Abstract classes
   2. Interfaces
   3. Static/global/hidden variables and classes
   4. Comparators
   5. Iterators
   6. Lambda Expressions
   7. Enums
   8. StringBuilder
2. **Review Use of Built-In Java Data Structures**
   1. String
   2. Arrays
   3. ArrayList
   4. List
   5. LinkedList
   6. Stack
   7. Queue
   8. PriorityQueue
   9. Set
   10. TreeSet
   11. LinkedHashSet
   12. Map
   13. HashMap
   14. LinkedHashMap
   15. TreeMap
3. **Review from 1332 Notes** (In the past I had already covered a, b, e, f, h in the list below)
   1. Arrays and Array Lists
   2. Linked Lists
   3. Stacks
      1. Make Crib Sheet
   4. Queues
      1. Make Crib Sheet
   5. Trees
      1. Make Crib Sheet
   6. BST
   7. Heaps and Priority Queues
      1. Make Crib Sheet
   8. HashMaps
   9. SkipLists
   10. AVL Trees
   11. B-Trees
   12. Splay Trees
   13. SORTING
       1. Bubble Sort
       2. Insertion Sort
       3. Selection Sort
       4. Merge Sort
       5. Quick Sort
       6. LSD Radix Sort
       7. MSD Radix Sort
   14. Iterators
   15. String Searching
   16. Graphs
   17. Minimum Spanning Trees
   18. Dynamic Programming
4. **Read “Cracking the Coding Interview”**
   1. Chapter 1 – Arrays & Strings
      1. Problems
   2. Chapter 2 – Linked Lists
      1. Problems
   3. Chapter 3 – Stacks & Queues
      1. Problems
   4. Chapter 4 – Trees & Graphs
      1. Problems
   5. Chapter 5 – Bit Manipulation
      1. Problems
   6. Chapter 6 – Math and Logic Puzzles
      1. Problems
   7. Chapter 7 – Object-Oriented Design
      1. Problems
   8. Chapter 8 – Recursion and Dynamic Programming
      1. Problems
   9. Chapter 9 – System Design and Scalability
      1. Problems
   10. Chapter 10 – Sorting and Searching
       1. Problems
   11. Chapter 11 – Testing
       1. Problems
   12. Chapter 12 – C and C++
       1. Problems
   13. Chapter 13 – Java
       1. Problems
   14. Chapter 14 – Databases
       1. Problems
   15. Chapter 15 – Threads and Locks
       1. Problems
   16. Chapter 16 – Moderate
   17. Chapter 17 - Hard
5. **CodeDeploy** (In the past I had done a, b, c in the list below)
   1. Arrays
   2. LinkedLists
   3. Hash Tables (If time, go and do some more on other sites)
   4. *Trees: Basic*
   5. *Heaps, Stacks, Queues*
   6. *Graphs*
   7. *Trees: Advanced*
   8. *DFS & BFS*
   9. Backtracking
   10. Sorting
   11. *Dynamic Programming: Basic*
   12. Dynamic Programming: Advanced
   13. Common Techniques: Basic
   14. *Strings*
   15. Bits
   16. Common Techniques: Advanced
   17. RegEx

**Other Concepts**

* Interpreted vs. Compiled Language

**Python Study Plan**

* Review solved problems from CodeFights:

**STUDY ORDER:**

1. CodeFights Python
2. Python Notes
3. MATLAB from past Homeworks
4. Trees/Graphs, continue with original study plan!
   * Arrays
   * Linked Lists
   * Hash Tables
   * Trees: Basic
   * Heaps, Stacks, Queues
   * Graphs
   * Trees: Advanced
   * DFS and BFS
   * Backtracking
   * Sorting
   * Dynamic Programming: Basics
   * Dynamic Programming: Advanced
   * Common Techniques: Basics
   * Strings
   * Bits
   * Common Techniques: Advanced
   * RegEx

* Review my own python notes
  + …

# Google Technical Preparation Notes

This is a list of things they tell you to know before interviewing with them (from the PPT)

1. **Coding:** You will be asked to write code in a Google Doc. You may be asked to:
   1. Construct/Traverse data structures
   2. Implement System Routines
   3. Distill Large data sets to single values
   4. Transform one data set to another
2. **Algorithms:** They suggest TopCoder. You may be asked:
   1. Big-O analysis (*very important*)
   2. Sorting and Hashing
   3. Handling Large amounts of data
3. **Sorting:** Know at least two sorting algorithms (Merge sort and quicksort, for example)
   1. Know common sorting algorithms
   2. On what kind of input data are they efficient and on which are they not efficient?
   3. What does efficiency mean in these cases in terms of runtime and space used?
   4. E.g. In exceptional cases insertion-sort, or radix-sort are much better than the generic QuickSort/MergeSort/HeapSort
4. **Data Structures:** Study as many other structures and algorithms as possible
   1. Know the most famous classes of NP-complete problems (e.g. TSP and Knapsack)
   2. Be able to recognize them when the interviewer asks you in disguise
   3. Find out what NP-complete means
   4. Know about Trees, tree construction, traversal, manipulation, hash tabes, stacks, arrays, linked lists, priority queues
5. **HashTables and Maps**: “Hashtables are arguably the single most important data structure known to mankind”
   1. You should be able to implement one using only arrays in your favorite language, in
   2. about the space of one interview
   3. Know the O() characteristics of the standard library implementation for Hashtables and Maps in your chosen language
6. **Trees**
   1. Know basic tree construction, traversal, and manipulation algorithms
   2. Know binary trees, n-ary trees, tri-treas *at least*
   3. Be familiar with at least one type of balanced binary tree (red/black tree, splay tree, AVL)
   4. Know how it’s implemented
   5. Know BFS/DFS
   6. Know inorder/postorder/preorder traversals
7. **Min/Max Heaps**
   1. Understand their applications and characteristics
   2. Probably won’t be asked to implement one, but know when it makes sense to use
8. **Graphs**
   1. Know graph algorithms for distance, search, connectivity, cycle-detection, etc.
   2. There are 3 basic ways to represent graphs in memory that you need to know (including the pros and cons):
      1. Objects and pointers
      2. Matrix
      3. Adjacency list
   3. Know the basic graph traversal algorithms, BFS, and DFS
      1. Know their complexity, tradeoffs and how to implement them in code
9. **Recursion**
   1. Prepare for it
   2. Practice some problems that can be solved iteratively, but a more elegant solution is recursion
10. **Operating Systems**
    1. You understand processes, threads, concurrency issues, locks, mutexes, semaphores, monitors and how they all work
    2. Understand deadlock, livelock, and how to avoid them
    3. Know what resources a process needs and a thread needs
    4. Understand how context switching works, how it’s initiated by the operating system and underlying hardware
    5. Know a little about scheduling
    6. Know the fundamentals of “modern” concurrency constructs (multi-core)
11. **Mathematics**
    1. Spend some time reviewing the essentials of elementary probability theory and combinatorics
    2. You should be familiar with n-choose-k problems and their ilk