Don't feel you aren't smart enough

- Google engineers are smart, but many have an insecurity that they aren't smart enough, even though they work at Google.
- The myth of the Genius Programmer

About Google

- [x] For students - Google Careers: Technical Development Guide
- [x] How Search Works:
  - [x] The Evolution of Search (video)
  - [x] How Search Works - the story
  - [x] How Search Works
  - [x] How Search Works - Matt Cutts (video)
  - [x] How Google makes improvements to its search algorithm (video)
- [ ] Series:
  - [ ] How Google Search Dealt With Mobile
  - [ ] Google's Secret Study To Find Out Our Needs
  - [ ] Google Search Will Be Your Next Brain
  - [ ] The Deep Mind Of Demis Hassabis
- [ ] Book: How Google Works
- [ ] Made by Google announcement - Oct 2016 (video)

About Video Resources

Some videos are available only by enrolling in a Coursera, EdX, or Lynda.com class. These are called MOOCs. It is free to do so, but sometimes the classes are not in session so you have to wait a couple of months, so you have no access.

```
I'd appreciate your help converting the MOOC video links to public sources to
replace the online course videos over time. I like using university lectures.
```

Interview Process & General Interview Prep

- Videos:
  - How to Work at Google - Candidate Coaching Session (video)
  - Google Recruiters Share Technical Interview Tips (video)
  - How to Work at Google: Tech Resume Preparation (video)
- Articles:
  - Becoming a Googler in Three Steps
  - Get That Job at Google
    - all the things he mentions that you need to know are listed below
  - *(very dated)* How To Get A Job At Google, Interview Questions, Hiring Process
  - Phone Screen Questions
- Additional (not suggested by Google but I added):
  - ABC: Always Be Coding
  - Four Steps To Google Without A Degree
  - Whiteboarding
  - How Google Thinks About Hiring, Management And Culture
  - Effective Whiteboarding during Programming Interviews
  - Cracking The Coding Interview Set 1:
    - Gayle L McDowell - Cracking The Coding Interview (video)
    - Cracking the Coding Interview with Author Gayle Laakmann McDowell (video)
  - How to Get a Job at the Big 4:
    - 'How to Get a Job at the Big 4 - Amazon, Facebook, Google & Microsoft' (video)
  - Failing at Google Interviews

Pick One Language for the Interview

I wrote this short article about it: Important: Pick One Language for the Google Interview

You can use a language you are comfortable in to do the coding part of the interview, but for Google, these are solid choices:

- C++
- Java
- Python

You could also use these, but read around first. There may be caveats:

- JavaScript
- Ruby

You need to be very comfortable in the language, and be knowledgeable.

Read more about choices:

- http://www.byte-by-byte.com/choose-the-right-language-for-your-coding-interview/
- http://blog.codingforinterviews.com/best-programming-language-jobs/
- https://www.quora.com/What-is-the-best-language-to-program-in-for-an-in-person-Google-interview

See language resources here

You'll see some C, C++, and Python learning included below, because I'm learning. There are a few books involved, see the bottom.

Before you Get Started

This list grew over many months, and yes, it kind of got out of hand.

Here are some mistakes I made so you'll have a better experience.

1. You Won't Remember it All

I watched hours of videos and took copious notes, and months later there was much I didn't remember. I spent 3 days going through my notes and making flashcards so I could review (see below).

2. Use Flashcards

To solve the problem, I made a little flashcards site where I could add flashcards of 2 types: general and code. Each card has different formatting.

I made a mobile-first website so I could review on my phone and tablet, whereever I am.

Make your own for free:

- Flashcards site repo
- My flash cards database: Keep in mind I went overboard and have cards covering everything from assembly language and Python trivia to machine learning and statistics. It's way too much for what's required by Google.

Note on flashcards: The first time you recognize you know the answer, don't mark it as known. You have to see the same card and answer it several times correctly before you really know it. Repetition will put that knowledge deeper in your brain.

3. Review, review, review

I keep a set of cheatsheets on ASCII, OSI stack, Big-O notations, and more. I study them when I have some spare time.

Take a break from programming problems for a half hour and go through your flashcards.

4. Focus

There are a lot of distractions that can take up valuable time. Focus and concentration is hard.

What you won't see covered

This big list all started as a personal to-do list made from Google interview coaching notes. These are prevalent technologies but were not mentioned in those notes:

- SQL
- Javascript
- HTML, CSS, and other front-end technologies

The Daily Plan

Some subjects take one day, and some will take multiple days. Some are just learning with nothing to implement.

Each day I take one subject from the list below, watch videos about that subject, and write an implementation in: C - using structs and functions that take a struct * and something else as args. C++ - without using built-in types C++ - using built-in types, like STL's std::list for a linked list Python - using built-in types (to keep practicing Python) and write tests to ensure I'm doing it right, sometimes just using simple assert() statements You may do Java or something else, this is just my thing.

Why code in all of these? Practice, practice, practice, until I'm sick of it, and can do it with no problem (some have many edge cases and bookkeeping details to remember) Work within the raw constraints (allocating/freeing memory without help of garbage collection (except Python)) Make use of built-in types so I have experience using the built-in tools for real-world use (not going to write my own linked list implementation in production)

I may not have time to do all of these for every subject, but I'll try.

You can see my code here:

- C
- C++
- Python

You don't need to memorize the guts of every algorithm.

Write code on a whiteboard, not a computer. Test with some sample inputs. Then test it out on a computer.

Prerequisite Knowledge

- **How computers process a program:**
  - How does CPU execute program (video)
  - Machine Code Instructions (video)
- **Compilers**
  - How a Compiler Works in ~1 minute (video)
  - Hardvard CS50 - Compilers (video)
  - C++ (video)
  - Understanding Compiler Optimization (C++) (video)
- **How floating point numbers are stored:**
  - simple 8-bit: Representation of Floating Point Numbers - 1 (video - there is an error in calculations - see video description)
  - 32 bit: IEEE754 32-bit floating point binary (video)

Algorithmic complexity / Big-O / Asymptotic analysis

- nothing to implement
- Harvard CS50 - Asymptotic Notation (video)
- Big O Notations (general quick tutorial) (video)
- Big O Notation (and Omega and Theta) - best mathematical explanation (video)
- Skiena:
  - video
  - slides
- A Gentle Introduction to Algorithm Complexity Analysis
- Orders of Growth (video)

- [ ] Asymptotics (video)
- [ ] UC Berkeley Big O (video)
- [ ] UC Berkeley Big Omega (video)
- [ ] Amortized Analysis (video)
- [ ] Illustrating "Big O" (video)
- [ ] TopCoder (includes recurrence relations and master theorem):
  - Computational Complexity: Section 1
  - Computational Complexity: Section 2
- [ ] Cheat sheet

If some of the lectures are too mathy, you can jump down to the bottom and watch the discrete mathematics videos to get the background knowledge.

Data Structures

- Arrays
  - Implement an automatically resizing vector.
  - [ ] Description:
    - Arrays (video)
    - Basic Arrays (video)
    - Multi-dim (video)
    - Dynamic Arrays (video)
    - Jagged Arrays (video)
    - Resizing arrays (video)
  - [ ] Implement a vector (mutable array with automatic resizing):
    - [ ] Practice coding using arrays and pointers, and pointer math to jump to an index instead of using indexing.
    - [ ] new raw data array with allocated memory
      - can allocate int array under the hood, just not use its features
      - start with 16, or if starting number is greater, use power of 2 - 16, 32, 64, 128

        - [ ] size() - number of items
        - [ ] capacity() - number of items it can hold
        - [ ] is_empty()
        - [ ] at(index) - returns item at given index, blows up if index out of bounds
        - [ ] push(item)
        - [ ] insert(index, item) - inserts item at index, shifts that index's value and trailing elements to the right
        - [ ] prepend(item) - can use insert above at index 0
        - [ ] pop() - remove from end, return value
        - [ ] delete(index) - delete item at index, shifting all trailing elements left
        - [ ] remove(item) - looks for value and removes index holding it (even if in multiple places)
        - [ ] find(item) - looks for value and returns first index with that value, -1 if not found
        - [ ] resize(new_capacity) // private function
            - when you reach capacity, resize to double the size
            - when popping an item, if size is 1/4 of capacity, resize to half
    - [ ] Time
        - O(1) to add/remove at end (amortized for allocations for more space), index, or update
        - O(n) to insert/remove elsewhere
    - [ ] Space
        - contiguous in memory, so proximity helps performance
        - space needed = (array capacity, which is >= n) * size of item, but even if 2n, still O(n)

- Linked Lists
    - [ ] Description:
        - [ ] Singly Linked Lists (video)

        - [ ] CS 61B - Linked Lists (video)
    - [ ] C Code (video) - not the whole video, just portions about Node struct and memory allocation.
    - [ ] Linked List vs Arrays:
        - Core Linked Lists Vs Arrays (video)
        - In The Real World Linked Lists Vs Arrays (video)
    - [ ] why you should avoid linked lists (video)
    - [ ] Gotcha: you need pointer to pointer knowledge: (for when you pass a pointer to a function that may change the address where that pointer points) This page is just to get a grasp on ptr to ptr. I don't recommend this list traversal style. Readability and maintainability suffer due to cleverness.
        - Pointers to Pointers
    - [ ] implement (I did with tail pointer & without):
        - [ ] size() - returns number of data elements in list
        - [ ] empty() - bool returns true if empty
        - [ ] value_at(index) - returns the value of the nth item (starting at 0 for first)
        - [ ] push_front(value) - adds an item to the front of the list
        - [ ] pop_front() - remove front item and return its value
        - [ ] push_back(value) - adds an item at the end
        - [ ] pop_back() - removes end item and returns its value
        - [ ] front() - get value of front item
        - [ ] back() - get value of end item
        - [ ] insert(index, value) - insert value at index, so current item at that index is pointed to by new item at index
        - [ ] erase(index) - removes node at given index
        - [ ] value_n_from_end(n) - returns the value of the node at nth position from the end of the list
        - [ ] reverse() - reverses the list
        - [ ] remove_value(value) - removes the first item in the list with this value

- [ ] Doubly-linked List
  - Description (video)
  - No need to implement

- Stack
  - [ ] Stacks (video)
  - [ ] Using Stacks Last-In First-Out (video)
  - [ ] Will not implement. Implementing with array is trivial.

- Queue
  - [ ] Using Queues First-In First-Out(video)
  - [ ] Queue (video)
  - [ ] Circular buffer/FIFO
  - [ ] Priority Queues (video)
  - [ ] Implement using linked-list, with tail pointer:
    - enqueue(value) - adds value at position at tail
    - dequeue() - returns value and removes least recently added element (front)
    - empty()
  - [ ] Implement using fixed-sized array:
    - enqueue(value) - adds item at end of available storage
    - dequeue() - returns value and removes least recently added element
    - empty()
    - full()
  - [ ] Cost:
    - a bad implementation using linked list where you enqueue at head and dequeue at tail would be O(n) because you'd need the next to last element, causing a full traversal each dequeue
    - enqueue: O(1) (amortized, linked list and array [probing])
    - dequeue: O(1) (linked list and array)
    - empty: O(1) (linked list and array)

- Hash table
  - Videos:
    - Hashing with Chaining (video)
    - Table Doubling, Karp-Rabin (video)
    - Open Addressing, Cryptographic Hashing (video)
    - PyCon 2010: The Mighty Dictionary (video)
    - (Advanced) Randomization: Universal & Perfect Hashing (video)
    - (Advanced) Perfect hashing (video)
  - Online Courses:
    - Understanding Hash Functions (video
    - Using Hash Tables (video)
    - Supporting Hashing (video)
    - Language Support Hash Tables (video)
    - Core Hash Tables (video)
    - Data Structures (video)
    - Phone Book Problem (video)
    - distributed hash tables:
      - Instant Uploads And Storage Optimization In Dropbox (video)
      - Distributed Hash Tables (video)
  - implement with array using linear probing
    - hash(k, m) - m is size of hash table
    - add(key, value) - if key already exists, update value
    - exists(key)
    - get(key)
    - remove(key)

More Knowledge

- Binary search
  - [ ] Binary Search (video)
  - [ ] Binary Search (video)
  - [ ] detail
  - [ ] Implement:
    - binary search (on sorted array of integers)
    - binary search using recursion
- Bitwise operations
  - [ ] Bits cheat sheet - you should know many of the powers of 2 from (2^1 to 2^16 and 2^32)
  - [ ] Get a really good understanding of manipulating bits with: &, |, ^, ~, >>, <<
    - [ ] words
    - [ ] Good intro: Bit Manipulation (video)
    - [ ] C Programming Tutorial 2-10: Bitwise Operators (video)
    - [ ] Bit Manipulation
    - [ ] Bitwise Operation
    - [ ] Bithacks
    - [ ] The Bit Twiddler
    - [ ] The Bit Twiddler Interactive
  - [ ] 2s and 1s complement
    - Binary: Plusses & Minuses (Why We Use Two's Complement) (video)
    - 1s Complement
    - 2s Complement
  - [ ] count set bits
    - 4 ways to count bits in a byte (video)

        - Count Bits
        - How To Count The Number Of Set Bits In a 32 Bit Integer
    - round to next power of 2:
        - Round Up To Next Power Of Two
    - swap values:
        - Swap
    - absolute value:
        - Absolute Integer

Trees

- Trees - Notes & Background
    - Series: Core Trees (video)
    - Series: Trees (video)
    - basic tree construction
    - traversal
    - manipulation algorithms
    - BFS (breadth-first search)
        - MIT (video)
        - level order (BFS, using queue) time complexity: O(n) space complexity: best: O(1), worst: O(n/2)=O(n)
    - DFS (depth-first search)
        - MIT (video)
        - notes: time complexity: O(n) space complexity: best: O(log n) - avg. height of tree worst: O(n)
        - inorder (DFS: left, self, right)
        - postorder (DFS: left, right, self)
        - preorder (DFS: self, left, right)
- Binary search trees: BSTs
    - Binary Search Tree Review (video)
    - Series (video)
        - starts with symbol table and goes through BST applications

- Introduction (video)
- MIT (video)
- C/C++:
  - Binary search tree - Implementation in C/C++ (video)
  - BST implementation - memory allocation in stack and heap (video)
  - Find min and max element in a binary search tree (video)
  - Find height of a binary tree (video)
  - Binary tree traversal - breadth-first and depth-first strategies (video)
  - Binary tree: Level Order Traversal (video)
  - Binary tree traversal: Preorder, Inorder, Postorder (video)
  - Check if a binary tree is binary search tree or not (video)
  - Delete a node from Binary Search Tree (video)
  - Inorder Successor in a binary search tree (video)
- Implement:
  - insert // insert value into tree
  - get_node_count // get count of values stored
  - print_values // prints the values in the tree, from min to max
  - delete_tree
  - is_in_tree // returns true if given value exists in the tree
  - get_height // returns the height in nodes (single node's height is 1)
  - get_min // returns the minimum value stored in the tree
  - get_max // returns the maximum value stored in the tree
  - is_binary_search_tree
  - delete_value
  - get_successor // returns next-highest value in tree after given value, -1 if none

- Heap / Priority Queue / Binary Heap
  - visualized as a tree, but is usually linear in storage (array, linked list)
  - [ ] Heap
  - [ ] Introduction (video)
  - [ ] Naive Implementations (video)
  - [ ] Binary Trees (video)
  - [ ] Tree Height Remark (video)
  - [ ] Basic Operations (video)
  - [ ] Complete Binary Trees (video)
  - [ ] Pseudocode (video)
  - [ ] Heap Sort - jumps to start (video)
  - [ ] Heap Sort (video)
  - [ ] Building a heap (video)
  - [ ] MIT: Heaps and Heap Sort (video)
  - [ ] CS 61B Lecture 24: Priority Queues (video)
  - [ ] Linear Time BuildHeap (max-heap)
  - [ ] Implement a max-heap:
    - [ ] insert
    - [ ] sift_up - needed for insert
    - [ ] get_max - returns the max item, without removing it
    - [ ] get_size() - return number of elements stored
    - [ ] is_empty() - returns true if heap contains no elements
    - [ ] extract_max - returns the max item, removing it
    - [ ] sift_down - needed for extract_max
    - [ ] remove(i) - removes item at index x
    - [ ] heapify - create a heap from an array of elements, needed for heap_sort

        - [ ] heap_sort() - take an unsorted array and turn it into a sorted array in-place using a max heap
            - note: using a min heap instead would save operations, but double the space needed (cannot do in-place).

- Tries
    - Note there are different kinds of tries. Some have prefixes, some don't, and some use string instead of bits to track the path.
    - I read through code, but will not implement.
    - [ ] Notes on Data Structures and Programming Techniques
    - [ ] Short course videos:
        - [ ] Introduction To Tries (video)
        - [ ] Performance Of Tries (video)
        - [ ] Implementing A Trie (video)
    - [ ] The Trie: A Neglected Data Structure
    - [ ] TopCoder - Using Tries
    - [ ] Stanford Lecture (real world use case) (video)
    - [ ] MIT, Advanced Data Structures, Strings (can get pretty obscure about halfway through)

- Balanced search trees
    - Know least one type of balanced binary tree (and know how it's implemented):
    - "Among balanced search trees, AVL and 2/3 trees are now passé, and red-black trees seem to be more popular. A particularly interesting self-organizing data structure is the splay tree, which uses rotations to move any accessed key to the root." - Skiena
    - Of these, I chose to implement a splay tree. From what I've read, you won't implement a balanced search tree in your interview. But I wanted exposure to coding one up and let's face it, splay trees are the bee's knees. I did read a lot of red-black tree code.
        - splay tree: insert, search, delete functions If you end up implementing red/black tree try just these:
        - search and insertion functions, skipping delete

- I want to learn more about B-Tree since it's used so widely with very large data sets.
- [ ] Self-balancing binary search tree
- [ ] **AVL trees**
  - In practice: From what I can tell, these aren't used much in practice, but I could see where they would be: The AVL tree is another structure supporting O(log n) search, insertion, and removal. It is more rigidly balanced than red–black trees, leading to slower insertion and removal but faster retrieval. This makes it attractive for data structures that may be built once and loaded without reconstruction, such as language dictionaries (or program dictionaries, such as the opcodes of an assembler or interpreter).
  - [ ] MIT AVL Trees / AVL Sort (video)
  - [ ] AVL Trees (video)
  - [ ] AVL Tree Implementation (video)
  - [ ] Split And Merge
- [ ] **Splay trees**
  - In practice: Splay trees are typically used in the implementation of caches, memory allocators, routers, garbage collectors, data compression, ropes (replacement of string used for long text strings), in Windows NT (in the virtual memory, networking, and file system code) etc.
  - [ ] CS 61B: Splay Trees (video)
  - [ ] MIT Lecture: Splay Trees:
    - Gets very mathy, but watch the last 10 minutes for sure.
    - Video
- [ ] **2-3 search trees**
  - In practice: 2-3 trees have faster inserts at the expense of slower searches (since height is more compared to AVL trees).
  - You would use 2-3 tree very rarely because its implementation involves different types of nodes. Instead, people use Red Black trees.

        - [ ] 23-Tree Intuition and Definition (video)
        - [ ] Binary View of 23-Tree
        - [ ] 2-3 Trees (student recitation) (video)
    - [ ] **2-3-4 Trees (aka 2-4 trees)**
        - In practice: For every 2-4 tree, there are corresponding red–black trees with data elements in the same order. The insertion and deletion operations on 2-4 trees are also equivalent to color-flipping and rotations in red–black trees. This makes 2-4 trees an important tool for understanding the logic behind red–black trees, and this is why many introductory algorithm texts introduce 2-4 trees just before red–black trees, even though **2-4 trees are not often used in practice**.
        - [ ] CS 61B Lecture 26: Balanced Search Trees (video)
        - [ ] Bottom Up 234-Trees (video)
        - [ ] Top Down 234-Trees (video)
    - [ ] **B-Trees**
        - fun fact: it's a mystery, but the B could stand for Boeing, Balanced, or Bayer (co-inventor)
        - In Practice: B-Trees are widely used in databases. Most modern filesystems use B-trees (or Variants). In addition to its use in databases, the B-tree is also used in filesystems to allow quick random access to an arbitrary block in a particular file. The basic problem is turning the file block i address into a disk block (or perhaps to a cylinder-head-sector) address.
        - [ ] B-Tree
        - [ ] Introduction to B-Trees (video)
        - [ ] B-Tree Definition and Insertion (video)
        - [ ] B-Tree Deletion (video)
        - [ ] MIT 6.851 - Memory Hierarchy Models (video) - covers cache-oblivious B-Trees, very interesting data structures - the first 37 minutes are very technical, may be skipped (B is block size, cache line size)

    - **Red/black trees**
        - In practice: Red–black trees offer worst-case guarantees for insertion time, deletion time, and search time. Not only does this make them valuable in time-sensitive applications such as real-time applications, but it makes them valuable building blocks in other data structures which provide worst-case guarantees; for example, many data structures used in computational geometry can be based on red–black trees, and the Completely Fair Scheduler used in current Linux kernels uses red–black trees. In the version 8 of Java, the Collection HashMap has been modified such that instead of using a LinkedList to store identical elements with poor hashcodes, a Red-Black tree is used.
        - Aduni - Algorithms - Lecture 4 (link jumps to starting point) (video)
        - Aduni - Algorithms - Lecture 5 (video)
        - Black Tree
        - An Introduction To Binary Search And Red Black Tree

- N-ary (K-ary, M-ary) trees
    - note: the N or K is the branching factor (max branches)
        - binary trees are a 2-ary tree, with branching factor = 2
        - 2-3 trees are 3-ary
    - K-Ary Tree

Sorting

- Notes:
    - Implement sorts & know best case/worst case, average complexity of each:
        - no bubble sort - it's terrible - O(n^2), except when n <= 16
    - stability in sorting algorithms ("Is Quicksort stable?")
        - Sorting Algorithm Stability
        - Stability In Sorting Algorithms
        - Stability In Sorting Algorithms

        - Sorting Algorithms - Stability
    - [ ] Which algorithms can be used on linked lists? Which on arrays? Which on both?
        - I wouldn't recommend sorting a linked list, but merge sort is doable.
        - Merge Sort For Linked List

- For heapsort, see Heap data structure above. Heap sort is great, but not stable.

- [ ] Bubble Sort (video)
- [ ] Analyzing Bubble Sort (video)
- [ ] Insertion Sort, Merge Sort (video)
- [ ] Insertion Sort (video)
- [ ] Merge Sort (video)
- [ ] Quicksort (video)
- [ ] Selection Sort (video)
- [ ] Stanford lectures on sorting:
    - [ ] Lecture 15 | Programming Abstractions (video)
    - [ ] Lecture 16 | Programming Abstractions (video)
- [ ] Shai Simonson, Aduni.org:
    - [ ] Algorithms - Sorting - Lecture 2 (video)
    - [ ] Algorithms - Sorting II - Lecture 3 (video)
- [ ] Steven Skiena lectures on sorting:
    - [ ] lecture begins at 26:46 (video)
    - [ ] lecture begins at 27:40 (video)
    - [ ] lecture begins at 35:00 (video)
    - [ ] lecture begins at 23:50 (video)

- UC Berkeley:
  - CS 61B Lecture 29: Sorting I (video)
  - CS 61B Lecture 30: Sorting II (video)
  - CS 61B Lecture 32: Sorting III (video)
  - CS 61B Lecture 33: Sorting V (video)
- - Merge sort code:
  - Using output array
  - In-place
- - Quick sort code:
  - Implementation
  - Implementation
- Implement:
  - Mergesort: O(n log n) average and worst case
  - Quicksort O(n log n) average case
  - Selection sort and insertion sort are both O(n^2) average and worst case
  - For heapsort, see Heap data structure above.
- For curiosity - not required:
  - Radix Sort
  - Radix Sort (video)
  - Radix Sort, Counting Sort (linear time given constraints) (video)
  - Randomization: Matrix Multiply, Quicksort, Freivalds' algorithm (video)
  - Sorting in Linear Time (video)

Graphs

Graphs can be used to represent many problems in computer science, so this section is long, like trees and sorting were.

- Notes from Yegge:
    - There are three basic ways to represent a graph in memory:
        - objects and pointers
        - matrix
        - adjacency list
    - Familiarize yourself with each representation and its pros & cons
    - BFS and DFS - know their computational complexity, their tradeoffs, and how to implement them in real code
    - When asked a question, look for a graph-based solution first, then move on if none.
- Skiena Lectures - great intro:
    - CSE373 2012 - Lecture 11 - Graph Data Structures (video)
    - CSE373 2012 - Lecture 12 - Breadth-First Search (video)
    - CSE373 2012 - Lecture 13 - Graph Algorithms (video)
    - CSE373 2012 - Lecture 14 - Graph Algorithms (con't) (video)
    - CSE373 2012 - Lecture 15 - Graph Algorithms (con't 2) (video)
    - CSE373 2012 - Lecture 16 - Graph Algorithms (con't 3) (video)
- Graphs (review and more):
    - 6.006 Single-Source Shortest Paths Problem (video)
    - 6.006 Dijkstra (video)
    - 6.006 Bellman-Ford (video)
    - 6.006 Speeding Up Dijkstra (video)
    - Aduni: Graph Algorithms I - Topological Sorting, Minimum Spanning Trees, Prim's Algorithm - Lecture 6 (video)
    - Aduni: Graph Algorithms II - DFS, BFS, Kruskal's Algorithm, Union Find Data Structure - Lecture 7 (video)
    - Aduni: Graph Algorithms III: Shortest Path - Lecture 8 (video)

    - [ ] Aduni: Graph Alg. IV: Intro to geometric algorithms - Lecture 9 (video)
    - [ ] CS 61B 2014 (starting at 58:09) (video)
    - [ ] CS 61B 2014: Weighted graphs (video)
    - [ ] Greedy Algorithms: Minimum Spanning Tree (video)
    - [ ] Strongly Connected Components Kosaraju's Algorithm Graph Algorithm (video)
- Full Coursera Course:
    - [ ] Algorithms on Graphs (video)
- Yegge: If you get a chance, try to study up on fancier algorithms:
    - [ ] Dijkstra's algorithm - see above - 6.006
    - [ ] A*
        - [ ] A Search Algorithm
        - [ ] A* Pathfinding Tutorial (video)
        - [ ] A* Pathfinding (E01: algorithm explanation) (video)
- I'll implement:
    - [ ] DFS with adjacency list (recursive)
    - [ ] DFS with adjacency list (iterative with stack)
    - [ ] DFS with adjacency matrix (recursive)
    - [ ] DFS with adjacency matrix (iterative with stack)
    - [ ] BFS with adjacency list
    - [ ] BFS with adjacency matrix
    - [ ] single-source shortest path (Dijkstra)
    - [ ] minimum spanning tree
    - DFS-based algorithms (see Aduni videos above):
        - [ ] check for cycle (needed for topological sort, since we'll check for cycle before starting)

        - [ ] topological sort
        - [ ] count connected components in a graph
        - [ ] list strongly connected components
        - [ ] check for bipartite graph

You'll get more graph practice in Skiena's book (see Books section below) and the interview books

Even More Knowledge

- Recursion
    - [ ] Stanford lectures on recursion & backtracking:
        - [ ] Lecture 8 | Programming Abstractions (video)
        - [ ] Lecture 9 | Programming Abstractions (video)
        - [ ] Lecture 10 | Programming Abstractions (video)
        - [ ] Lecture 11 | Programming Abstractions (video)
    - when it is appropriate to use it
    - how is tail recursion better than not?
        - [ ] What Is Tail Recursion Why Is It So Bad?
        - [ ] Tail Recursion (video)

- Dynamic Programming
    - This subject can be pretty difficult, as each DP soluble problem must be defined as a recursion relation, and coming up with it can be tricky.
    - I suggest looking at many examples of DP problems until you have a solid understanding of the pattern involved.
    - [ ] Videos:
        - the Skiena videos can be hard to follow since he sometimes uses the whiteboard, which is too small to see
        - [ ] Skiena: CSE373 2012 - Lecture 19 - Introduction to Dynamic Programming (video)
        - [ ] Skiena: CSE373 2012 - Lecture 20 - Edit Distance (video)

        - [ ] Skiena: CSE373 2012 - Lecture 21 - Dynamic Programming Examples (video)
        - [ ] Skiena: CSE373 2012 - Lecture 22 - Applications of Dynamic Programming (video)
        - [ ] Simonson: Dynamic Programming 0 (starts at 59:18) (video)
        - [ ] Simonson: Dynamic Programming I - Lecture 11 (video)
        - [ ] Simonson: Dynamic programming II - Lecture 12 (video)
        - [ ] List of individual DP problems (each is short): Dynamic Programming (video)
    - [ ] Yale Lecture notes:
        - [ ] Dynamic Programming
    - [ ] Coursera:
        - [ ] The RNA secondary structure problem (video)
        - [ ] A dynamic programming algorithm (video)
        - [ ] Illustrating the DP algorithm (video)
        - [ ] Running time of the DP algorithm (video)
        - [ ] DP vs. recursive implementation (video)
        - [ ] Global pairwise sequence alignment (video)
        - [ ] Local pairwise sequence alignment (video)

- Combinatorics (n choose k) & Probability
    - [ ] Math Skills: How to find Factorial, Permutation and Combination (Choose) (video)
    - [ ] Make School: Probability (video)
    - [ ] Make School: More Probability and Markov Chains (video)
    - [ ] Khan Academy:
        - Course layout:
            - [ ] Basic Theoretical Probability
        - Just the videos - 41 (each are simple and each are short):
            - [ ] Probability Explained (video)

- NP, NP-Complete and Approximation Algorithms
    - Know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise.
    - Know what NP-complete means.
    - [ ] Computational Complexity (video)
    - [ ] Simonson:
        - [ ] Greedy Algs. II & Intro to NP Completeness (video)
        - [ ] NP Completeness II & Reductions (video)
        - [ ] NP Completeness III (Video)
        - [ ] NP Completeness IV (video)
    - [ ] Skiena:
        - [ ] CSE373 2012 - Lecture 23 - Introduction to NP-CompletenessNP Completeness IV (video)
        - [ ] CSE373 2012 - Lecture 24 - NP-Completeness Proofs (video)
        - [ ] CSE373 2012 - Lecture 25 - NP-Completeness Challenge (video)
    - [ ] Complexity: P, NP, NP-completeness, Reductions (video)
    - [ ] Complexity: Approximation Algorithms (video)
    - [ ] Complexity: Fixed-Parameter Algorithms (video)
    - Peter Norvik discusses near-optimal solutions to traveling salesman problem:
        - Jupyter Notebook
    - Pages 1048 - 1140 in CLRS if you have it.

- Caches
    - [ ] LRU cache:
        - [ ] The Magic of LRU Cache (100 Days of Google Dev) (video)
        - [ ] Implementing LRU (video)
        - [ ] LeetCode - 146 LRU Cache (C++) (video)
    - [ ] CPU cache:

        - [ ] MIT 6.004 L15: The Memory Hierarchy (video)
        - [ ] MIT 6.004 L16: Cache Issues (video)

- Processes and Threads
    - [ ] Computer Science 162 - Operating Systems (25 videos):
        - for precesses and threads see videos 1-11
        - Operating Systems and System Programming (video)
    - What Is The Difference Between A Process And A Thread?
    - Covers:
        - Processes, Threads, Concurrency issues
            - difference between processes and threads
            - processes
            - threads
            - locks
            - mutexes
            - semaphores
            - monitors
            - how they work
            - deadlock
            - livelock
        - CPU activity, interrupts, context switching
        - Modern concurrency constructs with multicore processors
        - Process resource needs (memory: code, static storage, stack, heap, and also file descriptors, i/o)
        - Thread resource needs (shares above (minus stack) with other threads in same process but each has its own pc, stack counter, registers and stack)
        - Forking is really copy on write (read-only) until the new process writes to memory, then it does a full copy.
        - Context switching
            - How context switching is initiated by the operating system and underlying hardware
    - [ ] threads in C++ (series - 10 videos)
    - [ ] concurrency in Python (videos):
        - [ ] Short series on threads

        - [ ] Python Threads
        - [ ] Understanding the Python GIL (2010)
            - reference
        - [ ] David Beazley - Python Concurrency From the Ground Up: LIVE! - PyCon 2015
        - [ ] Keynote David Beazley - Topics of Interest (Python Asyncio)
        - [ ] Mutex in Python

    Scalability and System Design are very large topics with many topics and resources, since there is a lot to consider when designing a software/hardware system that can scale. Expect to spend quite a bit of time on this.

- System Design, Scalability, Data Handling
    - Considerations from Yegge:
        - scalability
            - Distill large data sets to single values
            - Transform one data set to another
            - Handling obscenely large amounts of data
        - system design
            - features sets
            - interfaces
            - class hierarchies
            - designing a system under certain constraints
            - simplicity and robustness
            - tradeoffs
            - performance analysis and optimization
    - [ ] **START HERE**: System Design from HiredInTech
    - [ ] How Do I Prepare To Answer Design Questions In A Technical Inverview?
    - [ ] 8 Things You Need to Know Before a System Design Interview
    - [ ] Algorithm design
    - [ ] Database Normalization - 1NF, 2NF, 3NF and 4NF (video)

- System Design Interview - There are a lot of resources in this one. Look through the articles and examples. I put some of them below.
- How to ace a systems design interview
- Numbers Everyone Should Know
- How long does it take to make a context switch?
- Transactions Across Datacenters (video)
- A plain english introduction to CAP Theorem
- Paxos Consensus algorithm:
  - short video
  - extended video with use case and multi-paxos
  - paper
- Consistent Hashing
- NoSQL Patterns
- Optional: UML 2.0 Series (vido)
- OOSE: Software Dev Using UML and Java (21 videos):
  - Can skip this if you have a great grasp of OO and OO design practices.
  - OOSE: Software Dev Using UML and Java
- SOLID OOP Principles:
  - Bob Martin SOLID Principles of Object Oriented and Agile Design (video)
  - SOLID Design Patterns in C# (video)
  - SOLID Principles (video)
  - S - Single Responsibility Principle | Single responsibility to each Object
    - more flavor
  - O - Open/Closed Principal | On production level Objects are ready for extension for not for modification
    - more flavor
  - L - Liskov Substitution Principal | Base Class and Derived class follow 'IS A' principal
    - more flavor

        - [ ] I - Interface segregation principle | clients should not be forced to implement interfaces they don't use
            - Interface Segregation Principle in 5 minutes (video)
            - more flavor
        - [ ] D -Dependency Inversion principle | Reduce the dependency In composition of objects.
            - Why Is The Dependency Inversion Principle And Why Is It Important
            - more flavor
    - [ ] Scalability:
        - [ ] Great overview (video)
        - [ ] Short series:
            - Clones
            - Database
            - Cache
            - Asynchronism
        - [ ] Scalable Web Architecture and Distributed Systems
        - [ ] Fallacies of Distributed Computing Explained
        - [ ] Pragmatic Programming Techniques
            - extra: Google Pregel Graph Processing
        - [ ] Jeff Dean - Building Software Systems At Google and Lessons Learned (video)
        - [ ] Introduction to Architecting Systems for Scale
        - [ ] Scaling mobile games to a global audience using App Engine and Cloud Datastore (video)
        - [ ] How Google Does Planet-Scale Engineering for Planet-Scale Infra (video)
        - [ ] The Importance of Algorithms
        - [ ] Sharding
        - [ ] Scale at Facebook (2009)
        - [ ] Scale at Facebook (2012), "Building for a Billion Users" (video)
        - [ ] Engineering for the Long Game - Astrid Atkinson Keynote(video)

- [ ] 7 Years Of YouTube Scalability Lessons In 30 Minutes
  - video
- [ ] How PayPal Scaled To Billions Of Transactions Daily Using Just 8VMs
- [ ] How to Remove Duplicates in Large Datasets
- [ ] A look inside Etsy's scale and engineering culture with Jon Cowie (video)
- [ ] What Led Amazon to its Own Microservices Architecture
- [ ] To Compress Or Not To Compress, That Was Uber's Question
- [ ] Asyncio Tarantool Queue, Get In The Queue
- [ ] When Should Approximate Query Processing Be Used?
- [ ] Google's Transition From Single Datacenter, To Failover, To A Native Multihomed Architecture
- [ ] Spanner
- [ ] Egnyte Architecture: Lessons Learned In Building And Scaling A Multi Petabyte Distributed System
- [ ] Machine Learning Driven Programming: A New Programming For A New World
- [ ] The Image Optimization Technology That Serves Millions Of Requests Per Day
- [ ] A Patreon Architecture Short
- [ ] Tinder: How Does One Of The Largest Recommendation Engines Decide Who You'll See Next?
- [ ] Design Of A Modern Cache
- [ ] Live Video Streaming At Facebook Scale
- [ ] A Beginner's Guide To Scaling To 11 Million+ Users On Amazon's AWS
- [ ] How Does The Use Of Docker Effect Latency?
- [ ] Does AMP Counter An Existential Threat To Google?
- [ ] A 360 Degree View Of The Entire Netflix Stack

- [ ] Latency Is Everywhere And It Costs You Sales - How To Crush It
- [ ] Serverless (very long, just need the gist)
- [ ] What Powers Instagram: Hundreds of Instances, Dozens of Technologies
- [ ] Cinchcast Architecture - Producing 1,500 Hours Of Audio Every Day
- [ ] Justin.Tv's Live Video Broadcasting Architecture
- [ ] Playfish's Social Gaming Architecture - 50 Million Monthly Users And Growing
- [ ] TripAdvisor Architecture - 40M Visitors, 200M Dynamic Page Views, 30TB Data
- [ ] PlentyOfFish Architecture
- [ ] Salesforce Architecture - How They Handle 1.3 Billion Transactions A Day
- [ ] ESPN's Architecture At Scale - Operating At 100,000 Duh Nuh Nuhs Per Second
- [ ] See "Messaging, Serialization, and Queueing Systems" way below for info on some of the technologies that can glue services together
- [ ] Twitter:
  - O'Reilly MySQL CE 2011: Jeremy Cole, "Big and Small Data at @Twitter" (video)
  - Timelines at Scale
- For even more, see "Mining Massive Datasets" video series in the Video Series section.

- [ ] Practicing the system design process: Here are some ideas to try working through on paper, each with some documentation on how it was handled in the real world:
  - review: System Design from HiredInTech
  - cheat sheet
  - flow:
    a. Understand the problem and scope:
       - define the use cases, with interviewer's help
       - suggest additional features

        - remove items that interviewer deems out of scope
        - assume high availability is required, add as a use case
    b. Think about constraints:
        - ask how many requests per month
        - ask how many requests per second (they may volunteer it or make you do the math)
        - estimate reads vs. writes percentage
        - keep 80/20 rule in mind when estimating
        - how much data written per second
        - total storage required over 5 years
        - how much data read per second
    c. Abstract design:
        - layers (service, data, caching)
        - infrastructure: load balancing, messaging
        - rough overview of any key algorithm that drives the service
        - consider bottlenecks and determine solutions
- Exercises:
    - Design a CDN network: old article
    - Design a random unique ID generation system
    - Design an online multiplayer card game
    - Design a key-value database
    - Design a function to return the top k requests during past time interval
    - Design a picture sharing system
    - Design a recommendation system
    - Design a URL-shortener system: copied from above
    - Design a cache system

- Papers
    - These are Google papers and well-known papers.
    - Reading all from end to end with full comprehension will likely take more time than you have. I recommend being selective on papers and their sections.
    -  1978: Communicating Sequential Processes
        - implemented in Go

        - Love classic papers?
    - [ ] 2003: The Google File System
        - replaced by Colossus in 2012
    - [ ] 2004: MapReduce: Simplified Data Processing on Large Clusters
        - mostly replaced by Cloud Dataflow?
    - [ ] 2007: What Every Programmer Should Know About Memory (very long, and the author encourages skipping of some sections)
    - [ ] 2012: Google's Colossus
        - paper not available
    - [ ] 2012: AddressSanitizer: A Fast Address Sanity Checker:
        - paper
        - video
    - [ ] 2013: Spanner: Google's Globally-Distributed Database:
        - paper
        - video
    - [ ] 2014: Machine Learning: The High-Interest Credit Card of Technical Debt
    - [ ] 2015: Continuous Pipelines at Google
    - [ ] 2015: High-Availability at Massive Scale: Building Google's Data Infrastructure for Ads
    - [ ] 2015: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems
    - [ ] 2015: How Developers Search for Code: A Case Study
    - [ ] 2016: Borg, Omega, and Kubernetes

- Testing
    - To cover:
        - how unit testing works
        - what are mock objects
        - what is integration testing
        - what is dependency injection
    - [ ] Agile Software Testing with James Bach (video)
    - [ ] Open Lecture by James Bach on Software Testing (video)

- [ ] Steve Freeman - Test-Driven Development (that's not what we meant) (video)
  - slides
- [ ] TDD is dead. Long live testing.
- [ ] Is TDD dead? (video)
- [ ] Video series (152 videos) - not all are needed (video)
- [ ] Test-Driven Web Development with Python
- [ ] Dependency injection:
  - [ ] video
  - [ ] Tao Of Testing
- [ ] How to write tests

- Scheduling
  - in an OS, how it works
  - can be gleaned from Operating System videos
- Implement system routines
  - understand what lies beneath the programming APIs you use
  - can you implement them?
- String searching & manipulations
  - [ ] Search pattern in text (video)
  - [ ] Rabin-Karp (videos):
    - Rabin Karps Algorithm
    - Table Doubling, Karp-Rabin
  - [ ] Precomputing
  - [ ] Optimization: Implementation and Analysis
  - [ ] Knuth-Morris-Pratt (KMP):
    - Pratt Algorithm
    - Tutorial: The Knuth-Morris-Pratt (KMP) String Matching Algorithm
  - [ ] Boyer–Moore string search algorithm

        - Boyer-Moore String Search Algorithm
        - Advanced String Searching Boyer-Moore-Horspool Algorithms (video)
    - [ ] Coursera: Algorithms on Strings

Final Review

```
This section will have shorter videos that can you watch pretty quickly to
review most of the important concepts.
It's nice if you want a refresher often.
(More items will be added here)
```

General:

- [ ] Series of 2-3 minutes short subject videos (23 videos)
    - Videos
- [ ] Series of 2-5 minutes short subject videos - Michael Sambol (18 videos):
    - Videos

Sorts:

- [ ] Merge Sort: https://www.youtube.com/watch?v=GCae1WNvnZM

Books

Mentioned in Google Coaching

Read and do exercises:

- [ ] The Algorithm Design Manual (Skiena)
    - Book (can rent on kindle):
        - Algorithm Design Manual
    - Half.com is a great resource for textbooks at good prices.
    - Answers:
        - Solutions

        - Solutions
    - Errata

  Once you've understood everything in the daily plan, and read and done exercises from the the books above, read and do exercises from the books below. Then move to coding challenges (further down below)

Read first:

- [ ] Programming Interviews Exposed: Secrets to Landing Your Next Job, 2nd Edition

Read second (recommended by many, but not in Google coaching docs):

- [ ] Cracking the Coding Interview, 6th Edition
    - If you see people reference "The Google Resume", it was a book replaced by "Cracking the Coding Interview".

Additional books

```
These were not suggested by Google but I added because I needed the background knowledge
```

- [ ] C Programming Language, Vol 2
    - answers to questions
- [ ] C++ Primer Plus, 6th Edition
- [ ] The Unix Programming Environment
- [ ] Programming Pearls
- [ ] Algorithms and Programming: Problems and Solutions

If you have time

- [ ] Introduction to Algorithms
    - Half.com is a great resource for textbooks at good prices.

- [ ] Elements of Programming Interviews
    - all code is in C++, if you're looking to use C++ in your interview
    - good book on problem solving in general.

Coding exercises/challenges

Once you've learned your brains out, put those brains to work. Take coding challenges every day, as many as you can.

Programming Question Prep:

- [ ] Great intro (copied from System Design section): Algorithm design:
- [ ] How to Find a Solution
- [ ] How to Dissect a Topcoder Problem Statement
- [ ] Mathematics for Topcoders
- [ ] Dynamic Programming – From Novice to Advanced
- MIT Interview Materials
- Exercises for getting better at a given language

Programming:

- LeetCode
- TopCoder
- Project Euler (math-focused)
- Codewars
- HackerRank
- Codility
- InterviewCake
- InterviewBit
- Mock interviewers from big companies

Once you're closer to the interview

- Cracking The Coding Interview Set 2 (videos):
  - Cracking The Code Interview
  - Cracking the Coding Interview - Fullstack Speaker Series
  - Ask Me Anything: Gayle Laakmann McDowell (author of Cracking the Coding Interview)

Your Resume

- Ten Tips for a (Slightly) Less Awful Resume
- Great stuff at the back of Cracking The Coding Interview

Be thinking of for when the interview comes

```
Think of about 20 interview questions you'll get, along the lines of the items
below.
Have 2-3 answers for each
Have a story, not just data, about something you accomplished
```

- Why do you want this job?
- What's a tough problem you've solved?
- Biggest challenges faced?
- Best/worst designs seen?
- Ideas for improving an existing Google product.
- How do you work best, as an individual and as part of a team?
- Which of your skills or experiences would be assets in the role and why?
- What did you most enjoy at [job x / project y]?
- What was the biggest challenge you faced at [job x / project y]?
- What was the hardest bug you faced at [job x / project y]?
- What did you learn at [job x / project y]?
- What would you have done better at [job x / project y]?

Have questions for the interviewer

```
Some of mine (I already may know answer to but want their opinion or team
perspective):
```

- How large is your team?
- What is your dev cycle look like? Do you do waterfall/sprints/agile?

- Are rushes to deadlines common? Or is there flexibility?
- How are decisions made in your team?
- How many meetings do you have per week?
- Do you feel your work environment helps you concentrate?
- What are you working on?
- What do you like about it?
- What is the work life like?

Once You've Got The Job

Congratulations!

- 10 things I wish I knew on my first day at Google

Keep learning.

You're never really done.

```
**************************************************************************
**********************
**************************************************************************
**********************

Everything below this point is optional. These are my recommendations, not
Google's.
By studying these, you'll get greater exposure to more CS concepts, and will
be better prepared for
any software engineering job.

**************************************************************************
**********************
**************************************************************************
**********************
```

Additional Learning

- Unicode

    - [ ] The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets
    - [ ] What Every Programmer Absolutely, Positively Needs To Know About Encodings And Character Sets To Work With Text

- Endianness
    - [ ] Big And Little Endian
    - [ ] Big Endian Vs Little Endian (video)
    - [ ] Big And Little Endian Inside/Out (video)
        - Very technical talk for kernel devs. Don't worry if most is over your head.
        - The first half is enough.

- Emacs and vi(m)
    - suggested by Yegge, from an old Amazon recruiting post: Familiarize yourself with a unix-based code editor
    - vi(m):
        - video
        - set of 4 (vidoes):
            - The vi/vim editor - Lesson 1
            - The vi/vim editor - Lesson 2
            - The vi/vim editor - Lesson 3
            - The vi/vim editor - Lesson 4
        - Using Vi Instead of Emacs
    - emacs:
        - Basics Emacs Tutorial
        - set of 3 (videos):
            - Emacs Tutorial (Beginners) -Part 1- File commands, cut/copy/paste, cursor commands
            - Emacs Tutorial (Beginners) -Part 2- Buffer management, search, M-x grep and rgrep modes
            - Emacs Tutorial (Beginners) -Part 3- Expressions, Statements, ~/.emacs file and packages
        - Evil Mode: Or, How I Learned to Stop Worrying and Love Emacs (video)
        - Writing C Programs With Emacs

        - (maybe) Org Mode In Depth: Managing Structure (video)

- Unix command line tools
    - suggested by Yegge, from an old Amazon recruiting post. I filled in the list below from good tools.
    - [ ] bash
    - [ ] cat
    - [ ] grep
    - [ ] sed
    - [ ] awk
    - [ ] curl or wget
    - [ ] sort
    - [ ] tr
    - [ ] uniq
    - [ ] strace
    - [ ] tcpdump

- Information theory (videos)
    - [ ] Khan Academy
    - [ ] more about Markov processes:
        - [ ] Core Markov Text Generation
        - [ ] Core Implementing Markov Text Generation
        - [ ] Project = Markov Text Generation Walk Through
    - See more in MIT 6.050J Information and Entropy series below.

- Parity & Hamming Code (videos)
    - [ ] Intro
    - [ ] Parity
    - [ ] Hamming Code:

        - Error detection
        - Error correction
    - [ ] Error Checking

- Entropy
    - also see videos below
    - make sure to watch information theory videos first
    - [ ] Information Theory, Claude Shannon, Entropy, Redundancy, Data Compression & Bits (video)

- Cryptography
    - also see videos below
    - make sure to watch information theory videos first
    - [ ] Khan Academy Series
    - [ ] Cryptography: Hash Functions
    - [ ] Cryptography: Encryption

- Compression
    - make sure to watch information theory videos first
    - [ ] Computerphile (videos):
        - [ ] Compression
        - [ ] Entropy in Compression
        - [ ] Upside Down Trees (Huffman Trees)
        - [ ] EXTRA BITS/TRITS - Huffman Trees
        - [ ] Elegant Compression in Text (The LZ 77 Method)
        - [ ] Text Compression Meets Probabilities
    - [ ] Compressor Head videos
    - [ ] (optional) Google Developers Live: GZIP is not enough!

- Networking (videos)

- [ ] Khan Academy
- [ ] UDP and TCP: Comparison of Transport Protocols
- [ ] TCP/IP and the OSI Model Explained!
- [ ] Packet Transmission across the Internet. Networking & TCP/IP tutorial.
- [ ] HTTP
- [ ] SSL and HTTPS
- [ ] SSL/TLS
- [ ] HTTP 2.0
- [ ] Video Series (21 videos)
- [ ] Subnetting Demystified - Part 5 CIDR Notation

- Computer Security
  - MIT (23 videos)
    - [ ] Introduction, Threat Models
    - [ ] Control Hijacking Attacks
    - [ ] Buffer Overflow Exploits and Defenses
    - [ ] Privilege Separation
    - [ ] Capabilities
    - [ ] Sandboxing Native Code
    - [ ] Web Security Model
    - [ ] Securing Web Applications
    - [ ] Symbolic Execution
    - [ ] Network Security
    - [ ] Network Protocols
    - [ ] Side-Channel Attacks

- Garbage collection

    - [ ] Garbage collection (Java); Augmenting data str (video)
    - [ ] Compilers (video)
    - [ ] GC in Python (video)
    - [ ] Deep Dive Java: Garbage Collection is Good!
    - [ ] Deep Dive Python: Garbage Collection in CPython (video)

- Parallel Programming
    - [ ] Coursera (Scala)
    - [ ] Efficient Python for High Performance Parallel Computing (video)

- Design patterns
    - [ ] Quick UML review (video)
    - [ ] Learn these patterns:
        - [ ] strategy
        - [ ] singleton
        - [ ] adapter
        - [ ] prototype
        - [ ] decorator
        - [ ] visitor
        - [ ] factory, abstract factory
        - [ ] facade
        - [ ] observer
        - [ ] proxy
        - [ ] delegate
        - [ ] command
        - [ ] state
        - [ ] memento
        - [ ] iterator

        - [ ] composite
        - [ ] flyweight
    - [ ] Chapter 6 (Part 1) - Patterns (video)
    - [ ] Chapter 6 (Part 2) - Abstraction-Occurrence, General Hierarchy, Player-Role, Singleton, Observer, Delegation (video)
    - [ ] Chapter 6 (Part 3) - Adapter, Facade, Immutable, Read-Only Interface, Proxy (video)
    - [ ] Series of videos (27 videos)
    - [ ] Head First Design Patterns
        - I know the canonical book is "Design Patterns: Elements of Reusable Object-Oriented Software", but Head First is great for beginners to OO.
    - [ ] Handy reference: 101 Design Patterns & Tips for Developers

- Messaging, Serialization, and Queueing Systems
    - [ ] Thrift
        - Tutorial
    - [ ] Protocol Buffers
        - Tutorials
    - [ ] gRPC
        - gRPC 101 for Java Developers (video)
    - [ ] Redis
        - Tutorial
    - [ ] Amazon SQS (queue)
    - [ ] Amazon SNS (pub-sub)
    - [ ] RabbitMQ
        - Get Startet
    - [ ] Celery
        - First Steps With Celery
    - [ ] ZeroMQ
        - Intro - Read The Manual
    - [ ] ActiveMQ

    - [ ] Kafka
    - [ ] MessagePack
    - [ ] Avro

- Fast Fourier Transform
    - [ ] What is a Fourier transform? What is it used for?
    - [ ] What is the Fourier Transform? (video)
    - [ ] Divide & Conquer: FFT (video)
    - [ ] Understanding The FFT

- Bloom Filter
    - Given a Bloom filter with m bits and k hashing functions, both insertion and membership testing are O(k)
    - Bloom Filters
    - Bloom Filters | Mining of Massive Datasets | Stanford University
    - Tutorial
    - How To Write A Bloom Filter App

- van Emde Boas Trees
    - [ ] Divide & Conquer: van Emde Boas Trees (video)
    - [ ] MIT Lecture Notes

- Augmented Data Structures
    - [ ] CS 61B Lecture 39: Augmenting Data Structures

- Skip lists
    - "These are somewhat of a cult data structure" - Skiena
    - [ ] Randomization: Skip Lists (video)
    - [ ] For animations and a little more detail

- Network Flows
  - [ ] Ford-Fulkerson in 5 minutes (video)
  - [ ] Ford-Fulkerson Algorithm (video)
  - [ ] Network Flows (video)
- Disjoint Sets & Union Find
  - [ ] Disjoint Set
  - [ ] UCB 61B - Disjoint Sets; Sorting & selection (video)
  - [ ] Coursera (not needed since the above video explains it great):
    - [ ] Overview
    - [ ] Naive Implementation
    - [ ] Trees
    - [ ] Union By Rank
    - [ ] Path Compression
    - [ ] Analysis Options
- Math for Fast Processing
  - [ ] Integer Arithmetic, Karatsuba Multiplication (video)
  - [ ] The Chinese Remainder Theorem (used in cryptography) (video)
- Treap
  - Combination of a binary search tree and a heap
  - [ ] Treap
  - [ ] Data Structures: Treaps explained (video)
  - [ ] Applications in set operations
- Linear Programming (videos)

    - [ ] Linear Programming
    - [ ] Finding minimum cost
    - [ ] Finding maximum value
- Geometry, Convex hull (videos)
    - [ ] Graph Alg. IV: Intro to geometric algorithms - Lecture 9
    - [ ] Geometric Algorithms: Graham & Jarvis - Lecture 10
    - [ ] Divide & Conquer: Convex Hull, Median Finding
- Discrete math
    - see videos below
- Machine Learning
    - [ ] Why ML?
        - [ ] How Google Is Remaking Itself As A Machine Learning First Company
        - [ ] Large-Scale Deep Learning for Intelligent Computer Systems (video)
        - [ ] Deep Learning and Understandability versus Software Engineering and Verification by Peter Norvig
    - [ ] Google's Cloud Machine learning tools (video)
    - [ ] Google Developers' Machine Learning Recipes (Scikit Learn & Tensorflow) (video)
    - [ ] Tensorflow (video)
    - [ ] Tensorflow Tutorials
    - [ ] Practical Guide to implementing Neural Networks in Python])http://www.analyticsvidhya.com/blog/2016/04/neural-networks-python-theano/)
    - Courses:
        - [ ] Great starter course: Machine Learning
            - videos only

        - see videos 12-18 for a review of linear algebra (14 and 15 are duplicates)
    - [ ] Neural Networks for Machine Learning
    - [ ] Google's Deep Learning Nanodegree
    - [ ] Google/Kaggle Machine Learning Engineer Nanodegree
    - [ ] Self-Driving Car Engineer Nanodegree
    - [ ] Metis Online Course ($99 for 2 months)
  - Resources:
    - Great book: Data Science from Scratch: First Principles with Python: https://www.amazon.com/Data-Science-Scratch-Principles-Python/dp/149190142X
    - Data School: http://www.dataschool.io/

- Go
  - [ ] Videos:
    - [ ] Why Learn Go?
    - [ ] Go Programming
    - [ ] A Tour of Go
  - [ ] Books:
    - [ ] An Introduction to Programming in Go (read free online)
    - [ ] The Go Programming Language (Donovan & Kernighan)
  - [ ] Bootcamp

Additional Detail on Some Subjects

```
I added these to reinforce some ideas already presented above, but didn't want
to include them
above because it's just too much. It's easy to overdo it on a subject.
You want to get hired in this century, right?
```

- [ ] **More Dynamic Programming** (videos)
  - [ ] 6.006: Dynamic Programming I: Fibonacci, Shortest Paths
  - [ ] 6.006: Dynamic Programming II: Text Justification, Blackjack

    - [ ] 6.006: DP III: Parenthesization, Edit Distance, Knapsack
    - [ ] 6.006: DP IV: Guitar Fingering, Tetris, Super Mario Bros.
    - [ ] 6.046: Dynamic Programming & Advanced DP
    - [ ] 6.046: Dynamic Programming: All-Pairs Shortest Paths
    - [ ] 6.046: Dynamic Programming (student recitation)
- [ ] **Advanced Graph Processing** (videos)
    - [ ] Synchronous Distributed Algorithms: Symmetry-Breaking. Shortest-Paths Spanning Trees
    - [ ] Asynchronous Distributed Algorithms: Shortest-Paths Spanning Trees
- [ ] MIT **Probability** (mathy, and go slowly, which is good for mathy things) (videos):
    - [ ] MIT 6.042J - Probability Introduction
    - [ ] MIT 6.042J - Conditional Probability
    - [ ] MIT 6.042J - Independence
    - [ ] MIT 6.042J - Random Variables
    - [ ] MIT 6.042J - Expectation I
    - [ ] MIT 6.042J - Expectation II
    - [ ] MIT 6.042J - Large Deviations
    - [ ] MIT 6.042J - Random Walks
- [ ] Simonson: Approximation Algorithms (video)

Video Series

Sit back and enjoy. "netflix and skill" :P

- [ ] List of individual Dynamic Programming problems (each is short)
- [ ] x86 Architecture, Assembly, Applications (11 videos)

- MIT 18.06 Linear Algebra, Spring 2005 (35 videos)
- Excellent - MIT Calculus Revisited: Single Variable Calculus
- Computer Science 70, 001 - Spring 2015 - Discrete Mathematics and Probability Theory
- Discrete Mathematics (19 videos)
- CSE373 - Analysis of Algorithms (25 videos)
    - Skiena lectures from Algorithm Design Manual
- UC Berkeley 61B (Spring 2014): Data Structures (25 videos)
- UC Berkeley 61B (Fall 2006): Data Structures (39 videos)
- UC Berkeley 61C: Machine Structures (26 videos)
- OOSE: Software Dev Using UML and Java (21 videos)
- UC Berkeley CS 152: Computer Architecture and Engineering (20 videos)
- MIT 6.004: Computation Structures (49 videos)
- Carnegie Mellon - Computer Architecture Lectures (39 videos)
- MIT 6.006: Intro to Algorithms (47 videos)
- MIT 6.033: Computer System Engineering (22 videos)
- MIT 6.034 Artificial Intelligence, Fall 2010 (30 videos)
- MIT 6.042J: Mathematics for Computer Science, Fall 2010 (25 videos)
- MIT 6.046: Design and Analysis of Algorithms (34 videos)
- MIT 6.050J: Information and Entropy, Spring 2008 (19 videos)

- [ ] MIT 6.851: Advanced Data Structures (22 videos)
- [ ] MIT 6.854: Advanced Algorithms, Spring 2016 (24 videos)
- [ ] MIT 6.858 Computer Systems Security, Fall 2014
- [ ] Stanford: Programming Paradigms (17 videos)
    - Course on C and C++
- [ ] Introduction to Cryptography
    - more in series (not in order)
- [ ] Mining Massive Datasets - Stanford University (94 videos)

Computer Science Courses

- Directory of Online CS Courses
- Directory of CS Courses (many with online lectures)