We are excited to have you for Software Engineering on campus interviews with Google. On Campus Interviews are Step 3 in our Hiring Process. You are halfway there! Keep in mind that Google takes an academic approach to the interviewing process. We are interested in your thought process, your approach to problem solving, skills in algorithms, coding, and performance.

This document contains tips that may be useful for your upcoming interview. We have also included additional links at the bottom that may help you to prepare. Many of the resources included here are third party advice and this advice is not directly endorsed by Google. Past new grad and intern candidates have told us that these resources were helpful in their own preparation. We hope you will find them helpful as well.

Tech Interview Tips (Adapted from Googler Steve Yegge's"Get that job at Google.")

Algorithm Complexity: you need to know Big-O. It's a must. If you struggle with basic big-O complexity analysis, then you are almost guaranteed not to get hired. It's, like, one chapter in the beginning of one theory of computation book, so just go read it. You can do it.

For more information on algorithms, visit the links below and/or your friendly local algorithms textbook.
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index
http://www.cs.sunysb.edu/~algorith/

Coding
You should know at least one programming language really well, and it should *preferably* be C++ or Java. C# is OK too, since it's pretty similar to Java. *Python is also ok for Google interviews.* You will be expected to write some code in at least some of your interviews. You will be expected to know a fair amount of detail about your favorite programming language.

Strongly recommended Coding: Programming Interviews Exposed; Secrets to landing your next job by John Monagan and Noah Suojanen (Wiley Computer Publishing). S*ee if your library or career center has this book on hand, borrow it from a friend, or just make the ~$20 investment and buy it yourself.*

Sorting:
Know how to sort. Don't do bubble-sort. You should know the details of at least one n*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.

Hashtables:
Hashtables are arguably the single most important data structure known to mankind. You *absolutely have to know how they work*. You should be able to implement one using only arrays in your favorite language, in about the space of one interview

Trees:
You should know about trees. Know basic tree construction, traversal and manipulation algorithms. You should be familiar with binary trees, n-ary trees, and trie-trees at the very least.

You should be familiar with at least one flavor of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree. You should actually know how it's implemented.You should know about tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Graphs
There are three basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list), and you should familiarize yourself with each representation and its pros and cons. You should know the basic graph traversal algorithms: breadth-first search and depth-first search. You should know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, study up on fancier algorithms, such as Dijkstra and A*, They're really great for just about anything, from game programming to distributed computing to you name it. You should know them.

Other data structures
You should study up on as many other data structures and algorithms as you can. You should especially 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. Find out what NP-complete means.
Math
Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because we are surrounded by counting problems, probability problems, and other Discrete Math 101 situations. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.

Other Stuff
The stuff I've covered is actually mostly red-flags: stuff that really worries people if you don't know it. The discrete math is potentially optional, but somewhat risky if you don't know the first thing about it. Everything else I've mentioned you should know cold, and then you'll at least be prepped for the baseline interview level.

During the actual interview

- If you are doing an on campus interview you will most definitely be coding on a whiteboard or on a piece of paper. Grab a whiteboard and a practice partner and work on coding in this way

- You may have about 5 minutes of “get to know you” time with your interviewer. They might ask you to talk about interesting projects you've worked on, how you have contributed, and other points from your resume, but the majority each of your two 45 minute interviews will be solving a technical problem on a white board.
- We’re interested in how you approach problem-solving. Think out loud. Ask questions.
- Our questions will be in-depth. We want to see how you think about complicated problems.
- The right answer would be nice but it is not necessary – your thought process is more important
- Be up beat, and show your personality. Humble is > arrogant.

Practice Questions
Project Euler
TopCoder
You can also find some technical interviewing questions on Quora and other online forums. Please note that we regularly ban questions once we discover them on the interwebs.

Other helpful links
How to Get Hired by Dan Kegel
Five Essential Phone Screen Questions by Steve Yegge
Google Products
Google Students Blog
Google+ for Students page
About Google