Interview Preparation

MUST READS:

You must read **Cracking the Coding Interview** and Steve Yegge blog!

I would also strongly recommend that you read one of the below:

Effective Java, 2nd Edition, by Xoogler Joshua Bloch. There is no substitute!

Effective C++, Volume 1, and More Effective C++ by Scott Meyers. You can skip the sections relating to details we don't use in Google C++, such as exceptions.

ADDITIONAL INFORMATION:

Google interviews thousands of extremely intelligent, experienced, competent software developers every year. Most of them, sadly, do not get hired. Google interviews are *hard*. Engineers who tend to do well in the interview process and ultimately receive offers are those that both prepare and practice.

A must watch resource, which will be extremely useful and beneficial to a good interview - http://www.youtube.com/watch?v=oWbUtlUhwa8&feature=youtu.be

Addition videos include both Interviewing at Google and An inside look at Google.

Also make sure to check out our Google code style guides below:

C++/Python: http://code.google.com/p/google-styleguide/
Java: http://google-styleguide.googlecode.com/svn/trunk/javaguide.html
Android: http://source.android.com/source/code-style.html

Prepare:

Google interviews focus very heavily on algorithms and data structures at roughly the level of a two-semester undergraduate or graduate course. You'll be expected to know and apply: lists, maps, stacks, priority queues, binary trees, graphs, bags, and sets. You'll need to to talk about how they're implemented and why you'd choose one implementation or data structure instead of another. For algorithms you'll want to know greedy algorithms, divide-and-conquer, dynamic programming, recursion, and brute force search. You'll definitely want to be conversant with big-O notation, time-space complexity, and real world performance of all of this. Most importantly you'll need to be able to pick the right data structure and algorithm for a specific problem.

Preparation tips:

● You need to know Big O complexity analysis really well - it's OK to quickly come up with a brute force solution, but that's never going to be the answer - always look for an O(n*logn) solution or ideally a linear one
● Hash tables - be able to implement one using only arrays

● Trees - know tree construction, traversal, and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees and trie-trees, and at least one type of balanced binary tree
● Mathematics relating to statistics and probability is also useful to know
● Sample topics: construct/ traverse data structures, implement system routines, distill large data sets to single values, transform one data set to another

Introduction to Algorithms is a fantastic resource, as is **Programming Pearls.**

Systems Design

One of the onsite interview will likely focus on System Design. Several tips are below:

● You need to be able to do back of the envelope calculations!
○ Know your powers of 2 (binary)
○ Be able to express mega/giga/etc. in binary and/or scientific notation
○ Explain and justify your assumptions
○ Don't be afraid of dealing with huge numbers!
● Know a variety of searching and sorting algorithms (Quicksort, Mergesort, Hash Tables,etc.) - know more than one O(n*logn) sorting algorithm - how they work
● System design questions are a test of your problem solving skills - ask qualifying questions - make sure you explain your thought process - explain and justify your assumptions

A recommended video is: Building Software Systems at Google

Practice:

Once you've reviewed the material, the second thing you'll want to do is practice. Google onsite interviews take place on a whiteboard, not a keyboard. The difference is surprising. A few of hours of actual practice writing code on a white board makes a colossal difference. Try locking yourself in a conference room and solving problems on a whiteboard. If you can do it with a colleague so much the better. Try swapping the interviewer and interviewee roles with each question. If you don't have easy access to a whiteboard, writing out answers with pen on paper is a good second best. It won't be easy, unless you've practiced.

Links to help practice:

- http://rkpprogramming.wordpress.com/
- http://paultyma.blogspot.co.uk/2007/03/howto-pass-silicon-valley-software.html
- http://www.karrels.org/Ed/ACM/
- http://www.topcoder.com/
- http://www.cs.oswego.edu/~mohammad/contest/f05/Problems.htm
- http://ioinformatics.org/locations/ioi95/contest/index.shtml

Online Courses

Don't forget to check out Google Code University. Here is a selection for courses:

Google hires a lot of graduates from Stanford and MIT, but you don't have to attend these colleges to benefit from their courses. You can find the MIT Computer Science courses through **free** Opencourseware. Here's just a selection:

● 6.006: Introduction to Algorithms
● 6.033: Computer System Engineering
● 6.172: Performance Engineering of Software Systems
● 6.717: Software Engineering for Web Applications

Generic Interview Tips:

The following are a few details that can make an interview go more smoothly:

- Do not assume all numbers are ints. Ask for clarification if necessary.
- Do know how to use some sort of map data structure from the standard library of your language of choice; e.g. java.util.HashMap in Java.
- Do know how to use a good sorting algorithm from the standard library of your preferred language; e.g. Collections.sort() and Arrays.sort() in Java.
- Do know how to copy and resize an array if necessary. (Arrays.copyOfRange in Java).
- Try to avoid mutating arguments in your solution.
- Once you've got a solution, try to walk though the code with some small input. E.g. if you're asked to calculate the nearest integer to a square root of an integer without using floating point numbers, try your solution with 1, with 3, with 4. Talk it out with your interviewer as you do this.

Common follow up questions (once you have a solution):

- What's the performance? (e.g. O(N^2), N lg N, etc.) Can you do better?
- How do you test the solution?
- Is this thread safe? If not, can you make it so?
- How do you distribute the solution for very large data sets?