Software Engineering Prep Doc

Best of luck in your upcoming interview! Here are some tips to help you prepare. We hope you'll take the time to review the links and tips in this document. Please keep in mind that many of these are third party resources and that this advice is not directly endorsed by Google.

1) Plan Ahead:
Phone interview:
For your phone interview you'll need a computer with internet access. Before your interview you'll receive the link to the Google Doc that you'll be using in your interview, mainly for the coding questions.

Onsite interviews:
For onsite interviews make sure to give yourself plenty of time to arrive at our office on the morning of your interviews. Be ready for a full day of technical interviews, you want to be at your best for the last interview, too!

2) What to Expect:
The interviewer(s) will be interested in your knowledge of computer science principles (coding ability, data structures, algorithms, systems design, big O notation, etc.) and how they can be used in your solutions. Be prepared for some open-ended discussion of projects you've worked on in school, at previous jobs, or in your spare time.

3) Interview Questions:
Interview topics may cover: anything on your resume, whiteboard coding questions, building and developing complex algorithms and analyzing their performance characteristics, logic problems, systems design, and core computer science principles (hash tables, stacks, arrays, etc.). Computer Science fundamentals are a prerequisite for all engineering roles at Google, regardless of seniority, due to the complexities and global scale of the projects you'll work on.

To practice for your interview, you may want to visit the website www.topcoder.com. Launch the "Arena" widget and then go to the practice rooms. We suggest doing the problems in the first/second division to help you warm up for your interviews.

4) How to Succeed:
At Google, we believe in collaboration and sharing ideas. You'll likely need more information from the interviewer to analyze and answer the question to its fullest extent.

- It's OK to question your interviewer!
- If you don't understand, ask for help or clarification.
- If you need to assume something, feel free to ask your interviewer if it's a correct assumption.
- When asked to provide a solution, first define and frame the problem as you see it.
- Describe how you would like to tackle solving each part of the question.
- Let your interviewer know what you're thinking because he/she will be just as interested in your thought process as in your solution.

- Finally, LISTEN! Engineers are always collaborating here at Google, so it's important that you can listen to your interviewer - especially in the event that he/she is trying to assist you.

5) What is Google looking for?

We are not simply looking for engineers to solve the problems they already know the answers to; we are interested in engineers who can work out the answers to questions they had not come across before.

Interviewers look at the approach to the question as much as the answer. Some important questions are: Does the candidate...

- Listen carefully and comprehend the question?
- Ask the correct questions before proceeding?
- Enjoy finding multiple solutions before choosing the best one?
- Seek new ideas and methods for tackling the problem?
- Seem flexible and open to thoughts and new ideas?
- Have the ability to solve even more complex problems?

Google emphasizes really high quality, efficient, and clear code. Because all engineers (at every level) collaborate throughout the Google code base, with an efficient code review process, it's essential that every engineer works at the same high standard.

6) Technical Preparation tips

- **Algorithm Complexity:**
  - Please review complex algorithms, including big O notation. For more information on algorithms, visit the links below and your friendly local algorithms textbook.
    - Online Resources: Topcoder - Data Science Tutorials, The Stony Brook Algorithm Repository
    - Book Recommendations: *Review of Basic Algorithms: Introduction to the Design and Analysis of Algorithms* by Anany Levitin, *Algorithms* by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, *Algorithms* For Interviews by Adnan Aziz and Amit Prakash, *Algorithms* Course Materials by Jeff Erickson, *Introduction to Algorithms* by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
- **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.
- **Hash Tables:**
  - Be prepared to explain how they work, and be able to implement one using only arrays in your favorite language, in about the space of one interview.
- **Trees and Graphs:**
  - Study up on trees: 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, and you should know how it's implemented.
  - More generally, 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.

    - Tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder traversal (for trees). 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* (for graphs).
- **Other data structures:**
    - You should study up on as many other data structures and algorithms as possible. 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.
- **Operating Systems, Systems Programming and Concurrency:**
    - Know about processes, threads, and concurrency issues. Know about locks, mutexes, semaphores and monitors, and how they work. Know about deadlock and livelock and how to avoid them.
    - Know what resources a processes needs, a thread needs, how context switching works, and how it's initiated by the operating system and underlying hardware.
    - Know a little about scheduling. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs.
- **Coding:**
    - You should know at least one programming language really well, preferably C/C++, Java, Python, Go, or Javascript. (Or C# since it's similar to Java.)
    - You will be expected to write code in your interviews and you will be expected to know a fair amount of detail about your favorite programming language.
    - Book Recommendation: *Programming Interviews Exposed: Secrets to landing your next job* by John Monagan and Noah Suojanen (Wiley Computer Publishing)
- **Recursion and Induction:**
    - You should be able to solve a problem recursively, and know how to use and repurpose common recursive algorithms to solve new problems.
    - Conversely, you should be able to take a given algorithm and prove inductively that it will do what you claim it will do.
- **Data Structure Analysis and Discrete 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 on the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.
- **System Design:**
    - You should be able to take a big problem, decompose it into its basic subproblems, and talk about the pros and cons of different approaches to solving those subproblems as they relate to the original goal.
    - Google solves a lot of big problems; here are some explanations of how we solved a few to get your wheels turning.
        - Online Resources: Research at Google: Distributed Systems and Parallel Computing
        - Google File System
        - Google Bigtable
        - Google MapReduce
- **Development Practices and Open-Ended Discussion**:
    - Sample topics include validating designs, testing whiteboard code, preventing bugs, code maintainability and readability, refactor/review sample code.
    - Sample topics: biggest challenges faced, best/worst designs seen, performance analysis and optimization, testing and ideas for improving existing products.

7) Ask more questions!

Make sure you have a decent understanding of Google as a business, beyond Google's main products. Find out more about what we do here at Google.

At the end of the interview, most interviewers will ask you if you have any questions about the company, work environment, their experience, etc.. It's a good idea to have some questions ready for each interview, but don't worry too much if your mind goes blank. If you have questions about the interview process, remuneration, or your performance, please direct these to your recruiter.

8) Other Helpful Links
Interviewing at Google
Five Essential Phone Screen Questions by Steve Yegge
Project Euler
TopCoder - Data Science Tutorials
Google Products
Google Students Blog
Google+ for Students page
About Google

If you have any additional questions, please let me know.

GOOD LUCK!!!