INTERVIEW QUESTIONS:
Interview topics may cover anything on your CV (especially if you have stated that you are an expert!), 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 prerequisite for all engineering roles at Google, regardless of seniority, due to the complexities and global scale of the projects you would end up participating in.

HOW TO SUCCEED:
At Google, we believe in collaboration and sharing ideas. Most importantly, you'll need more information from the interviewer to analyze and answer the question to its full extent.
* Its OK to question your interviewer.
* When asked to provide a solution, first define and frame the problem as you see it.
* If you don't understand - ask for help or clarification.
* If you need to assume something - verbally check its a correct assumption!
* Describe how you want to tackle solving each part of the question.
* Always let your interviewer know what you are thinking as he/she will be as interested in your process of thought as your solution. Also, if you're stuck, they may provide hints if they know what you're doing.
* Finally... listen - don't miss a hint if your interviewer is trying to assist you!

Interviewers will be looking at the approach to questions as much as the answer:
* Does the candidate listen carefully and comprehend the question?
* Are the correct questions asked before proceeding? (important!)
* Is brute force used to solve a problem? (not good!)
* Are things assumed without first checking? (not good!)
* Are hints heard and heeded?
* Is the candidate slow to comprehend / solve problems? (not good!)
* Does the candidate enjoy finding multiple solutions before choosing the
best one?
* Are new ideas and methods of tackling a problem sought?
* Is the candidate inventive and flexible in their solutions and open to
new ideas?
* Can questioning move up to more complex problem solving?

Google is keen to see really high quality, efficient, clear code without typing mistakes. 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.

TECHNICAL PREPARATION TIPS
The main areas software engineers should prepare to succeed at interviewing at Google:

1.) Coding: You should know at least one programming language really well, preferably C++ or Java. 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.

2.) Algorithm Complexity: It's fairly critical that you understand big-O complexity analysis. Again run some practice problems to get this down in application. For more information on Algorithms you can visit:
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index

3.) 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.

4.) Hashtables: You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

5.) Trees: Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees, and trie-trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

6.) Graphs: Graphs are really important at Google. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons.
You should know the basic graph traversal algorithms: breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A*.

7.) 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. Find out what NP-complete means.

8.) Mathematics: 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.

9.) Operating Systems: Know about processes, threads and concurrency issues. Know about locks and mutexes and semaphores and monitors and how they work. Know about deadlock and livelock and how to avoid them. Know what resources a processes needs, and a thread needs, and 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. For information on System Design:
http://research.google.com/pubs/DistributedSystemsandParallelComputing.html

SAMPLE TOPICS

Coding
Sample topics: construct / traverse data structures, implement system routines, distill large data sets to single values, transform one data set to another.

Algorithm Design / Analysis
Sample topics: big-O analysis, sorting and hashing, handling obscenely large amounts of data. Also see topics listed under 'Coding'.

System Design
Sample topics: features sets, interfaces, class hierarchies, designing a system under certain constraints, simplicity and robustness, tradeoffs.

Open-Ended Discussion
Sample topics: biggest challenges faced, best/worst designs seen, performance analysis and optimization, testing, ideas for improving existing products.

READING LIST

(Extra references as an FYI only this is not mandatory)
- "Programming Interviews Exposed: Secrets to Landing Your Next Job" by John Mongan and Noah Suojanen
- "Programming Pearls" by Jon Bentley
- "Cormen/Leiserson/Rivest/Stein: Introduction to Algorithms" or the CLR textbook.