What you should know (and what your Professors probably didn't tell you about..)

A few tips for a successful Google interview:

***Practice your whiteboard coding!!!!!!!!!!!!! :)

The interview will include topics such as coding, data structures, algorithms, computer science theory, and systems design. We recommend you spend some time exploring our website to get into the right mind frame.

Books to assist you in preparing for your interview:
"The Google Resume: How to Prepare for a Career and Land a Job at Apple, Microsoft, Google, or any Top Tech Company"
Author: Gayle Laakmann McDowell

"The Algorithm Design Manual"
Author: Steven S. Skiena

"Introduction to Algorithms"
Authors: Cormen, Leiserson, Rivest & Stein
[Also known as the “CLRS” textbook]

“Programming Interviews Exposed: Secrets to Landing Your Next Job"
Authors: John Mongan, Noah Suojanen, and Eric Giguère (Wiley Computer Publishing)
[This book will prepare you for what to expect during an interview.
White board preparation is especially useful for onsite interviews.]

"Programming Pearls"
Author: Jon Bentley
[Programming questions that get you to start thinking outside of the box-- always useful in a Google interview :)]

Also, this blog post should be helpful in letting you know what to expect and how to prepare:
http://steve-yegge.blogspot.com/2008/03/get-that-job-at-google.html

System Design Publications:
http://research.google.com/archive/mapreduce.html
http://research.google.com/archive/gfs.html
http://research.google.com/archive/bigtable.html

Please review the following topics:
Big-O notations also known as "the run time characteristic of an algorithm". You may want to refresh hash tables, heaps, binary trees, linked lists, depth-first search, recursion. For more information on Algorithms you can visit:
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index

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

Python). 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 preferred programming language and will be asked to do some coding on the whiteboard.

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 quick sort is impractical, so take a look at it.

Hashtables: Arguably the single most important data structure known to mankind. 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.

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.
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*.

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 whatNP-complete means.

Mathematics: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems, and other Discrete Math 101 situations surrounds us. Spend some time before theinterview 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.

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

Helpful links & Sites:
- http://www.youtube.com/watch?v=w887NIa_V9w
- http://www.youtube.com/lifeatgoogle
- http://codility.com/
- http://www.spoj.pl/tutorials/
- www.hackquest.com

- http://www.ets.org/gre/subject/about/content/computer_science
- http://projecteuler.net/

A few last tips:
Talk through your thought process about the questions you are asked. In all of Google's interviews, our engineers are evaluating not only your technical abilities but also how you approach problems and how you try to solve them.
Ask clarifying questions if you do not understand the problem or need more information. Many of the questions asked in Google interviews are deliberately underspecified because our engineers are looking to see how you engage the problem. In particular, they are looking to see which areas leap to your mind as the most important piece of the technological puzzle you've been presented.
Think about ways to improve the solution you'll present. In many cases, the first answer that springs to mind may not be the most elegant solution and may need some refining. It's definitely worthwhile to talk through your initial thoughts to a question and take time to compose a more efficient solution.