Stevey's Blog Rants

RANDOM WHINING AND STUFF.

Wednesday, March 12, 2008

Get that job at Google

I've been meaning to write up some tips on interviewing at Google for a good long time now. I keep putting it off, though, because it's going to make you mad. Probably. For some statistical definition of "you", it's very likely to upset you.

Why? Because... well, here, I wrote a little ditty about it:

```
Hey man, I don't know that stuff
Stevey's talking aboooooout
If my boss thinks it's important
I'm gonna get fiiiiiiiiiired
Oooh yeah baaaby baaaay-beeeeee....
```

I didn't realize this was such a typical reaction back when I first started writing about interviewing, way back at other companies. Boy-o-howdy did I find out in a hurry.

See, it goes like this:

Me: blah blah blah, I like asking question X in interviews, blah blah blah...

You: Question X? Oh man, I haven't heard about X since college! I've never needed it for my job! He asks that in *interviews*? But that means someone out there thinks it's important to know, and, and... I don't know it! If they detect my ignorance, not only will I be summarily fired for incompetence without so much as a thank-you, I will also be unemployable by people who ask question X! If people listen to Stevey, that will be everyone! I will become homeless and destitute! For not knowing something I've never needed before! This is horrible! I would attack X itself, except that I do not want to pick up a book and figure enough out about it to discredit it. Clearly I must yell a lot about how stupid Stevey is so that nobody will listen to him!

Me: So in conclusion, blah blah... huh? Did you say "fired"? "Destitute?" What are you talking about?

About Me

STEVE YEGGE
KIRKLAND, WASHINGTON, UNITED STATES

VIEW MY COMPLETE PROFILE

Previous Posts

- Portrait of a Noob
- Emergency Elisp
- Blogging Theory 201: Size Does Matter
- Code's Worst Enemy
- Boring Stevey Status Update
- Ten Tips for a (Slightly) Less Awful Resume
- Stevey's Tech News, Issue #1
- How To Make a Funny Talk Title Without Using The W...
- Rhino on Rails
- Rich Programmer Food

I Power Blogger

You: Aaaaaaauuuggh!!! *stab* *stab* *stab*

Me: That's it. I'm never talking about interviewing again.

It doesn't matter what X is, either. It's arbitrary. I could say: "I really enjoy asking the candidate *(their name)* in interviews", and people would still freak out, on account of insecurity about either interviewing in general or their knowledge of their own name, hopefully the former.

But THEN, time passes, and interview candidates come and go, and we always wind up saying: "Gosh, we sure wish that obviously smart person had prepared a little better for his or her interviews. Is there any way we can help future candidates out with some tips?"

And then nobody actually does anything, because we're all afraid of getting stabbed violently by People Who Don't Know X.

I considered giving out a set of tips in which I actually use variable names like X, rather than real subjects, but decided that in the resultant vacuum, *everyone* would get upset. Otherwise that approach seemed pretty good, as long as I published under a pseudonym.

In the end, people really need the tips, regardless of how many feelings get hurt along the way. So rather than skirt around the issues, I'm going to give you a few mandatory substitutions for X along with a fair amount of general interview-prep information.

Caveats and Disclaimers

This blog is not endorsed by Google. Google doesn't know I'm publishing these tips. It's just between you and me, OK? Don't tell them I prepped you. Just go kick ass on your interviews and we'll be square.

I'm only talking about general software engineering positions, and interviews for those positions.

These tips are actually generic; there's nothing specific to Google vs. any other software company. I could have been writing these tips about my first software job 20 years ago. That implies that these tips are also timeless, at least for the span of our careers.

These tips obviously won't get you a job on their own. My hope is that by following them you will perform your very best during the interviews.

Oh, and um, why Google?

Oho! Why Google, you ask? Well let's just have that dialog right up front, shall we?

You: Should I work at Google? Is it all they say it is, and more? Will I be serenely happy there? Should I apply immediately?

Me: Yes.

You: To which ques... wait, what do you mean by "Yes?" I didn't even say who I am!

Me: Dude, the answer is Yes. (You may be a woman, but I'm still calling you Dude.)

You: But... but... I am paralyzed by inertia! And I feel a certain comfort level at my current company, or at least I have become relatively inured to the discomfort. I know people here and nobody at Google! I would have to learn Google's build system and technology and stuff! I have no credibility, no reputation there – I would have to start over virtually from scratch! I waited too long, there's no upside! I'm afraaaaaaid!

Me: DUDE. The answer is Yes already, OK? It's an invariant. Everyone else who came to Google was in the *exact same position* as you are, modulo a handful of famous people with beards that put Gandalf's to shame, but they're a very tiny minority. Everyone who applied had the same reasons for *not* applying as you do. And everyone here says: "GOSH, I SURE AM HAPPY I CAME HERE!" So just apply already. But prep first.

You: But what if I get a mistrial? I might be smart and qualified, but for some random reason I may do poorly in the interviews and not get an offer! That would be a huge blow to my ego! I would rather pass up the opportunity altogether than have a chance of failure!

Me: Yeah, that's at least partly true. Heck, I kinda didn't make it in on my first attempt, but I begged like a street dog until they gave me a second round of interviews. I caught them in a weak moment. And the second time around, I prepared, and did much better.

The thing is, Google has a well-known false negative rate, which means we sometimes turn away qualified people, because that's considered better than sometimes hiring unqualified people. This is actually an industry-wide thing, but the dial gets turned differently at different companies. At Google the false-negative rate is pretty high. I don't know what it is, but I do know a lot of smart, qualified people who've not made it through our interviews. It's a bummer.

But the really important takeaway is this: *if you don't get an offer, you may still be qualified to work here*. So it needn't be a blow to your ego at all!

As far as anyone I know can tell, false negatives are completely random, and are unrelated to your skills or qualifications. They can happen from a variety of factors, including but not limited to:

1. you're having an off day
2. one or more of your interviewers is having an off day
3. there were communication issues invisible to you and/or one or more of the interviewers
4. you got unlucky and got an Interview Anti-Loop

Oh no, not the Interview Anti-Loop!

Yes, I'm afraid you have to worry about this.

What is it, you ask? Well, back when I was at Amazon, we did (and they undoubtedly still do) a LOT of soul-searching about this exact problem. We eventually concluded that every single employee E at Amazon has at least one "Interview Anti-Loop": a set of other employees S who would not hire E. The root cause is important for you to understand when you're going into interviews, so I'll tell you a little about what I've found over the years.

First, you can't tell interviewers what's important. Not at any company. Not unless they're specifically asking you for advice. You have a very narrow window of perhaps one year after an engineer graduates from college to inculcate them in the art of interviewing, after which the window closes and they believe they are a "good interviewer" and they don't need to change their questions, their question styles, their interviewing style, or their feedback style, *ever again*.

It's a problem. But I've had my hand bitten enough times that I just don't try anymore.

Second problem: every "experienced" interviewer has a set of pet subjects and possibly specific questions that he or she feels is an accurate gauge of a candidate's abilities. The question sets for any two interviewers can be widely different and even entirely non-overlapping.

A classic example found everywhere is: Interviewer A always asks about C++ trivia, filesystems, network protocols and discrete math. Interviewer B always asks about Java trivia, design patterns, unit testing, web frameworks, and software project management. For any given candidate with both A and B on the interview loop, A and B are likely to give very different votes. A and B would probably not even hire each other, given a chance, but they both happened to go through interviewer C, who asked them both about data structures, unix utilities, and processes versus threads, and A and B both happened to squeak by.

That's almost always what happens when you get an offer from a tech company. You just happened to squeak by. Because of the inherently flawed nature of the interviewing process, it's highly likely that *someone* on the loop will be unimpressed with you, even if you are Alan Turing. Especially if you're Alan Turing, in fact, since it means you obviously don't know C++.

The bottom line is, if you go to an interview at *any* software company, you should plan for the contingency that you might get genuinely unlucky, and wind up with one or more people from your Interview Anti-Loop on your interview loop. If this happens, you will struggle, then be told that you were not a fit at this time, and then you will feel bad. Just as long as you don't feel meta-bad, everything is OK. You should feel *good* that you feel bad after this happens, because hey, it means you're human.

And then you should wait 6-12 months and re-apply. That's pretty much the best solution we (or anyone else I know of) could come up with for the false-negative problem. We wipe the slate clean and start over again. There are lots of people here who got in on their second or third attempt, and they're kicking butt.

You can too.

OK, I feel better about potentially not getting hired

Good! So let's get on to those tips, then.

If you've been following along *very* closely, you'll have realized that I'm interviewer D. Meaning that my personal set of pet questions and topics is just my own, and it's no better or worse than anyone else's. So I can't tell you what it is, no matter how much I'd like to, because I'll offend interviewers A through X who have slightly different working sets.

Instead, I want to prep you for some general topics that I believe are shared by the majority of tech interviewers at Google-like companies. Roughly speaking, this means the company builds a lot of their own software and does a lot of distributed computing. There are other tech-company footprints, the opposite end of the spectrum being companies that outsource everything to consultants and try to use as much third-party software as possible. My tips will be useful only to the extent that the company resembles Google.

So you might as well make it Google, eh?

First, let's talk about non-technical prep.

The Warm-Up

Nobody goes into a boxing match cold. Lesson: you should bring your boxing gloves to the interview. No, wait, sorry, I mean: warm up beforehand!

How do you warm up? Basically there is short-term and long-term warming up, and you should do both.

Long-term warming up means: study and practice for a week or two before the interview. You want your mind to be in the general "mode" of problem solving on whiteboards. If you can do it on a whiteboard, every other medium (laptop, shared network document, whatever) is a cakewalk. So plan for the whiteboard.

Short-term warming up means: get lots of rest the night before, and then do intense, fast-paced warm-ups the morning of the interview.

The two best long-term warm-ups I know of are:

1) **Study a data-structures and algorithms book**. Why? Because it is the most likely to help you beef up on problem identification. Many interviewers are happy when you understand the broad class of question they're asking without explanation. For instance, if they ask you about coloring U.S. states in different colors, you get major bonus points if you recognize it as a graph-coloring problem, even if you don't actually remember exactly how graph-coloring works.

And if you do remember how it works, then you can probably whip through the answer pretty quickly. So your best bet, interview-prep wise, is to practice the art of recognizing that certain problem classes are best solved with certain algorithms and data structures.

My absolute favorite for this kind of interview preparation is Steven Skiena's The

Algorithm Design Manual. More than any other book it helped me understand just how astonishingly commonplace (and important) graph problems are – they should be part of every working programmer's toolkit. The book also covers basic data structures and sorting algorithms, which is a nice bonus. But the gold mine is the second half of the book, which is a sort of encyclopedia of 1-pagers on zillions of useful problems and various ways to solve them, without too much detail. Almost every 1-pager has a simple picture, making it easy to remember. This is a great way to learn how to identify hundreds of problem types.

Other interviewers I know recommend Introduction to Algorithms. It's a true classic and an invaluable resource, but it will probably take you more than 2 weeks to get through it. But if you want to come into your interviews *prepped*, then consider deferring your application until you've made your way through that book.

2) **Have a friend interview you.** The friend should ask you a random interview question, and you should go write it on the board. You should keep going until it is complete, no matter how tired or lazy you feel. Do this as much as you can possibly tolerate.

I didn't do these two types of preparation before my first Google interview, and I was absolutely shocked at how bad at whiteboard coding I had become since I had last interviewed seven years prior. It's hard! And I also had forgotten a bunch of algorithms and data structures that I used to know, or at least had heard of.

Going through these exercises for a week prepped me mightily for my second round of Google interviews, and I did way, way better. It made all the difference.

As for short-term preparation, all you can really do is make sure you are as alert and warmed up as possible. Don't go in cold. Solve a few problems and read through your study books. Drink some coffee: it actually helps you think faster, believe it or not. Make sure you spend at *least* an hour practicing immediately before you walk into the interview. Treat it like a sports game or a music recital, or heck, an exam: if you go in warmed up you'll give your best performance.

Mental Prep

So! You're a hotshot programmer with a long list of accomplishments. Time to forget about all that and focus on interview survival.

You should go in humble, open-minded, and focused.

If you come across as arrogant, then people will question whether they want to work with you. The best way to appear arrogant is to question the validity of the interviewer's question – it really ticks them off, as I pointed out earlier on. Remember how I said you can't tell an interviewer how to interview? Well, that's *especially* true if you're a candidate.

So don't ask: "gosh, are algorithms really all that important? do you ever need to do that kind of thing in real life? I've never had to do that kind of stuff." You'll just get rejected, so don't say that kind of thing. Treat every question as legitimate, even if you are frustrated that you don't know the answer.

Feel free to ask for help or hints if you're stuck. Some interviewers take points off for that, but occasionally it will get you past some hurdle and give you a good performance on what would have otherwise been a horrible stony half-hour silence.

Don't say "choo choo choo" when you're "thinking".

Don't try to change the subject and answer a different question. Don't try to divert the interviewer from asking you a question by telling war stories. Don't try to bluff your interviewer. You should *focus* on each problem they're giving you and make your best effort to answer it fully.

Some interviewers will not ask you to write code, but they will *expect* you to start writing code on the whiteboard at some point during your answer. They will give you hints but won't necessarily come right out and say: "I want you to write some code on the board now." If in doubt, you should ask them if they would like to see code.

Interviewers have vastly different expectations about code. I personally don't care about syntax (unless you write something that could obviously never work in any programming language, at which point I will dive in and verify that you are not, in fact, a circus clown and that it was an honest mistake). But some interviewers are really picky about syntax, and some will even silently mark you down for missing a semicolon or a curly brace, *without telling you*. I think of these interviewers as – well, it's a technical term that rhymes with "bass soles", but they think of themselves as brilliant technical evaluators, and there's no way to tell them otherwise.

So ask. Ask if they care about syntax, and if they do, try to get it right. Look over your code carefully from different angles and distances. Pretend it's someone else's code and you're tasked with finding bugs in it. You'd be amazed at what you can miss when you're standing 2 feet from a whiteboard with an interviewer staring at your shoulder blades.

It's OK (and highly encouraged) to ask a few clarifying questions, and occasionally verify with the interviewer that you're on the track they want you to be on. Some interviewers will mark you down if you just jump up and start coding, *even if you get the code right*. They'll say you didn't think carefully first, and you're one of those "let's not do any design" type cowboys. So even if you think you know the answer to the problem, ask some questions and talk about the approach you'll take a little before diving in.

On the flip side, don't take too long before actually solving the problem, or some interviewers will give you a delay-of-game penalty. Try to move (and write) quickly, since often interviewers want to get through more than one question during the interview, and if you solve the first one too slowly then they'll be out of time. They'll mark you down because they couldn't get a full picture of your skills. The benefit of the doubt is rarely given in interviewing.

One last non-technical tip: bring your own whiteboard dry-erase markers. They sell pencil-thin ones at office supply stores, whereas most companies (including Google) tend to stock the fat kind. The thin ones turn your whiteboard from a 480i standard-definition tube into a 58-inch 1080p HD plasma screen. You need all the help you can get, and free whiteboard space is a real blessing.

You should also practice whiteboard space-management skills, such as not starting on the right and coding down into the lower-right corner in Teeny Unreadable Font. Your interviewer will not be impressed. Amusingly, although it always irks me when people do this, I did it during my interviews, too. Just be aware of it!

Oh, and don't let the marker dry out while you're standing there waving it. I'm tellin' ya: you want minimal distractions during the interview, and that one is surprisingly common.

OK, that should be good for non-tech tips. On to X, for some value of X! Don't stab me!

Tech Prep Tips

The best tip is: go get a computer science degree. The more computer science you have, the better. You don't have to have a CS degree, but it helps. It doesn't have to be an advanced degree, but that helps too.

However, you're probably thinking of applying to Google a little sooner than 2 to 8 years from now, so here are some shorter-term tips for you.

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.

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.

For God's sake, don't try sorting a linked list during the interview.

Hashtables: hashtables are arguably the single most important data structure known to mankind. You *absolutely have to know how they work*. Again, it's like one chapter in one data structures book, so just go read about them. 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. I'm tellin' ya: this is basic stuff, and it's embarrassing to bring it up, but some of you out there don't know basic tree construction, traversal and manipulation algorithms. You should be familiar with binary trees, n-ary trees, and trie-trees at the very *very* least. Trees are probably the best source of practice problems for your long-term warmup exercises.

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.

You might not use trees much day-to-day, but if so, it's because you're avoiding tree problems. You won't need to do that anymore once you know how they work. Study up!

Graphs

Graphs are, like, really *really* important. More than you think. Even if you already think they're important, it's probably more than you think.

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.

You should try to study up on fancier algorithms, such as Dijkstra and A*, if you get a chance. They're really great for just about anything, from game programming to distributed computing to you name it. You should know them.

Whenever someone gives you a problem, *think graphs*. They are the most fundamental and flexible way of representing any kind of a relationship, so it's about a 50-50 shot that any interesting design problem has a graph involved in it. Make absolutely sure you can't think of a way to solve it using graphs before moving on to other solution types. This tip is important!

Other data structures

You should study up on as many other data structures and algorithms as you can fit in that big noggin of yours. 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.

You should find out what NP-complete means.

Basically, hit that data structures book hard, and try to retain as much of it as you can, and you can't go wrong.

Math

Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other places I've been, and I consider it a Good Thing, even though I'm not particularly good at discrete math. We're surrounded by counting problems, probability problems, and other Discrete Math 101 situations, and those innumerate among us blithely hack around them without knowing what we're doing.

Don't get mad if the interviewer asks math questions. Do your best. Your best will be a heck of a lot better if you 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.

I know, I know, you're short on time. But this tip can really help make the difference between a "we're not sure" and a "let's hire her". And it's actually not all that bad – discrete math doesn't use much of the high-school math you studied and forgot. It starts back with elementary-school math and builds up from there, so you can probably pick up what you need for interviews in a couple of days of intense study.

Sadly, I don't have a good recommendation for a Discrete Math book, so if you do, please mention it in the comments. Thanks.

Operating Systems

This is just a plug, from me, for you to know about processes, threads and concurrency issues. A lot of interviewers ask about that stuff, and it's pretty fundamental, so you should know it. 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, and you'll be a dinosaur in a real hurry if you don't understand the fundamentals of "modern" (which is to say, "kinda broken") concurrency constructs.

The best, most practical book I've ever personally read on the subject is Doug Lea's Concurrent Programming in Java. It got me the most bang per page. There are obviously lots of other books on concurrency. I'd avoid the academic ones and focus on the practical stuff, since it's most likely to get asked in interviews.

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. 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.

Other Stuff

Because of the rules I outlined above, it's still possible that you'll get Interviewer A, and none of the stuff you've studied from these tips will be directly useful (except being warmed up.) If so, just do your best. Worst case, you can always come back in 6-12 months, right? Might seem like a long time, but I assure you it will go by in a flash.

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. It could be a lot harder than that, depending on the interviewer, or it could be easy.

It just depends on how lucky you are. Are you feeling lucky? Then give it a try!

Send me your resume

I'll probably batch up any resume submissions people send me and submit them weekly. In the meantime, study up! You have a lot of warming up to do. Real-world work makes you rusty.

I hope this was helpful. Let the flames begin, etc. Yawn.

POSTED BY STEVE YEGGE AT 6:16 PM

---

171 COMMENTS:

Ben said...

Thanks, Steve; that was very helpful, although it would've been more helpful before I had a phonescreen with you guys last fall and totally brainlocked on a tree traversal. I kid you not, I could hear the guy interviewing me impatiently tapping his fingers on the table. He was just ITCHING to pencilwhip my ass out of there. I don't interview all that well, even though I like to pretend I'm not a dumbass.

Big +1 on data structures and algorithm study, though. Not knowing the answer to the tree stuff made me go out and read algorithm/DS books, and it was very, VERY helpful. Plus I got to use it in an interview that I managed not to fail.

7:07 PM, MARCH 12, 2008

Silas Snider said...

The best discrete math book I've ever read has to be "Concrete Mathematics: A Foundation for Computer Science" by Graham, Knuth, and Patashnik

7:15 PM, MARCH 12, 2008

Greg said...

Great post. As a programmer who's hitting that "5 years out of college" threshold soon, some of those algorithms read less like everyday tools and more like old friends. Not good, not good, time to blow off the dust and crack open the books :)

7:28 PM, MARCH 12, 2008

Garret said...

MIT OpenCourseWare has really good lecture notes on discrete math for CS. They served as the textbook for a college class I took on the subject.

http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-042JMathematics-for-Computer-ScienceFall2002/Readings/index.htm

8:07 PM, MARCH 12, 2008

Basu said...

Considering that I'm still in my freshman year, I'm going to try to make full use of all this....I'll let you know how it goes in about....say 5 years.

8:17 PM, MARCH 12, 2008

~kristen said...

Thanks Steve. Great advice for a graduating senior. I got my first offer last week, but I never rule out Google!

A degree in CS is nice, but don't get discouraged by not having one. I get by just fine with my BS in Math and a pickup of basic CS class (plus some extra studying on my own). This is a great summary of everything a person should hit if you don't have that CS background.

For Combinatorics, try "Applied Combinatorics" (Roberts & Tessman). There are lots of examples and problems. It's a good book.

9:07 PM, MARCH 12, 2008

Alex Gartrell said...

http://www.cs.cmu.edu/~15251/

great lecture notes on discrete math on the wiki (for free).

9:26 PM, MARCH 12, 2008

John McCaskey said...

Thanks for a great set of interview tips Steve.

For those of you who might be considering taking the dive and applying at Google I have one more suggestion. Apply with us at Valve Software (you know, the video game company? Half-Life? Counter-Strike? Steam?) too. You'll get put through a very tough interview process similar to Google's and it will be a great warmup for your Google interviews if nothing more. Best case you'll pass with flying colors and we'll convince you that not even Google could be cooler or more fun than Valve.

http://www.valvesoftware.com/jobs.html

9:30 PM, MARCH 12, 2008

orion said...

Oh man, where was this a week ago! I just went through a four-hour brain squeezer of an interview for a company here in Manhattan and felt very stupid a few times that I'd forgotten some fairly basic stuff. (In my defense, I've been at architecture-astronaut level of abstraction in a very niche field for the last year as a consultant, so while I can talk at length about how one particular problem domain is handled in the financial industry I was brain-farting on things like what exactly the servlet life cycle is.) Oh well, at the very least an honest swing and miss would be respectable in my mind and it helped me figure out what my study list needs to be in the near term. Thank you very much for posting this list of pointers.

It looks like there's a bunch of good suggestions already for discrete maths texts both online and off, but here's a decent one (imho) that also has the virtues of being widely available and dirt cheap (~$15), "Discrete Mathematics (Schaum's Outlines)" by Lipschutz and Lipson (2e). I feel almost bad about suggesting it next to a list of "real" texts (e.g. anything by Knuth) but if you're long out of school or sold your existing text and want dead tree it's a solution.

10:03 PM, MARCH 12, 2008

Dave Mackie said...

Great post (as always). I can't say I'm interested in working at Google since I live in North Carolina (although I see you guys have opened a data center here...), but the book suggestions alone make it a good blog post. I have a copy of "Applied Combinatorics" (which someone else recommended), and I concur that it's a great book on the subject.

10:08 PM, MARCH 12, 2008

David Rupp said...

"Discrete Mathematics with Graph Theory", by E. Goodaire and M. Parmenter. ISBN: 0131679953.

10:39 PM, MARCH 12, 2008

Cooper said...

I did a phone interview @google about a year and a half ago just for giggles.

Gots no degree at all so I was curious how far I would get. And I love a challenge!

It was pretty brutal and it was just for an ops position. I didn't get it. First rejection of my career, which was pretty humbling!

My advice for an operations/IT gig, make sure you understand everything about the gig from the bottom up. I realized to my horror in mid-sentence that in all my career I had never really understood exactly how DNS was implemented. Fun!

Plus, don't ask the interviewer to repeat themselves. Take notes or interview on a speakerphone in front of a whiteboard.

Anyway, great article. I'm going to go make sure I understand all the CS topics from the bottom up. That should make for a fun weekend!

P.S. I wouldn't work for Google anyway. The fact that they advertise fraudulent services, like psychics, galls me. Especially given their motto and supposed dedication to science.

11:28 PM, MARCH 12, 2008

Robert Konigsberg said...

"But the really important takeaway is this: if you don't get an offer, you may still be qualified to work here. So it needn't be a blow to your ego at all!"

Bullshit. That's exactly what I would have said in between the thoughts of what questions I clearly got wrong.

11:42 PM, MARCH 12, 2008

Michael Head said...

Dang... your post comes about a month and a half late! Oh well, I'm sure I'll interview again when I'm closer to graduation.

11:44 PM, MARCH 12, 2008

Robert Konigsberg said...

OK besides my 'bullshit' comment, this is great. The idea of bringing in your own whiteboard markers is smart. I'd give credit for preparedness. It wouldn't turn a mediocre candidate into a good candidate, but it would be a good soft-touch credit.

There are definitely parts of this list I don't know, and parts I just have forgotten, and while I'm not looking for a job, this could turn in to an excellent skills check-list.

11:53 PM, MARCH 12, 2008

Dennis Roberts said...

Great write up! I am a system administrator trying to get into programming and this gives me a great place to start.

12:02 AM, MARCH 13, 2008

Michael Head said...

BTW, I am looking for work (specifically in the NYC office), and my resume is online: http://www.core.binghamton.edu/~burner/new/res.html

wink

12:16 AM, MARCH 13, 2008

Michael Head said...

One follow up question, is there a particularly good way of dealing with the "now, what questions do you have for us?" question?

12:23 AM, MARCH 13, 2008

CH Gowri Kumar said...

The Best Answers to Tough Interview Questions should be of some help for non-technical questions.

1:34 AM, MARCH 13, 2008

compiling said...

Steve, you are the best, as always. I had my Google interviews about month ago - and I can't agree more on everything you mentioned (well, maybe except for thin pencil thing ;).

One little addition to the tech prep skills section would be dynamic programming - the tasks on this one appear to be quite common.

2:29 AM, MARCH 13, 2008

Barry Kelly said...

FWIW, Adjacency lists and the object-oriented form for graph representation are effectively the same. The problem with adjacency lists is you have to decide where to put the lists. If you happen refer to the lists from your nodes, then you end up at the OO form without necessarily distinguishing it.

2:48 AM, MARCH 13, 2008

Dário said...

On the subject of Discrete Mathematics I would recommend "Concrete Mathematics" By Knuth, Graham and Patashnik.

3:54 AM, MARCH 13, 2008

Serge said...

Send me your resume
I'll probably batch up any resume submissions people send me and submit them weekly.

OMG, Steve, how much do you plan to cut on these referral bonuses?
If I get hired, can we share 50/50? :)

4:27 AM, MARCH 13, 2008

George Pikoulas said...

Very helpful post Steve thanx!

Also I am not sure why no one mentioned the free online educational resource form ArsDigita available at http://www.archive.org/details/arsdigita .

They span a wide range of CS topics.

5:07 AM, MARCH 13, 2008

Brian Di Croce said...

So that's it? I thought there was more to software than 1's and 0's. What about knowing how to deal with requirements or change requests? What about knowing a few principles about software quality? What about knowing some foundations on software architecture? What about some knowledge of effective development processes?

You said that these questions were more oriented to software engineers, but it sounds more oriented to computer scientists.

I'm not saying that algorithms and data structures aren't important, but there's more to development than just knowing where and how to place a bunch of 1's and 0's.

But, as you have said, there are many types of interviewers at tech companies. I think there should be two or more different types of interviewers per interview so the candidate can get a better chance of answering the questions (it might help to decrease the false negative results).

Thanks for writing this post, it was very informative.

--
www.BrianDiCroce.com

5:35 AM, MARCH 13, 2008

mike said...

I did the whole phone screen (I should say screen x 3) and flyout to Mountain View with Google. It was a fantastic experience regardless of the fact that I was sent home without an offer in hand (nor did one come since). A few people that I have spoken with regarding their experiences in the Google interview process are quite bitter. That is, they tend to fall into two categories: 1) "How dare they ask those college course questions!" and 2) "Screw them I didn't want to work there anyway" (these are not xor). However, I was never bitter at the way things shook out, and viewed it as a motivating factor for making myself smarter. You better believe that the next time someone asks me to design a concurrent queuing system, I will knock their damn socks off. ;)

Great post.

-m

6:24 AM, MARCH 13, 2008

cmonkey said...

had you just posted this 2 weeks ago I might have not been rejected by Google. Very good programming tips. I wonder, can you get rejected by google and reapply again anyway? I mean is there a timeout period I should wait for before reapplying?

7:47 AM, MARCH 13, 2008

Raviprakash said...

Thanks, Steve; that was very helpful.

I was recently interviewed with one of those goggle type companies and I was rejected after second round of phone interview.

The first one went ok and the second one went really well (at least from my perspective). The second interview was scheduled for 45mins but it went almost 1:15 mins. All the questions are exactly as you described in this blog. Alogorithms, operating systems and finally some coding in Java (yes, over the phone. He wanted me to read out the code for him on the phone).

After the second phone interview I was pretty sure I will be called for an onsite interview :-) but to my surprise I got an email from recruiter saying "After serious consideration, the team has decided to pursue other candidates at this time. We are currently reviewing your resume for other opportunities." Well I know that is a standard rejection email but I would have felt really good if I got a good feedback from the recruiter.

Is it possible to get a feedback on my interview? Can I write to the recruiter back asking for feedback or should I just leave it aside and move on with other opportunities. Of course 6months is not a long time :-).

Thanks again for your excellent post.

8:20 AM, MARCH 13, 2008

Tim said...

I just want to question the obsession with writing code on a whiteboard. Nobody ever writes code on a whiteboard (except maybe to very roughly show some structure). That's what computers, with nice text editors, and syntax highlighting are for. Everybody types (and certainly edits) faster than they can write on a whiteboard. I'd rather write code in notepad than on a whiteboard. Give your interviewees a shot at writing code on an actual computer, you know, like what they would actually do on their job. As a bonus that makes it really easy for you to test if the code really works.

There's a lot of good stuff in this post though. I enjoyed it.

Tim

8:21 AM, MARCH 13, 2008

Jeff said...

I'm curious, why Google? I mean, aside from working for a huge conglomerate and perhaps it's "the chic place to work", I don't see why Google can afford to be so picky through it's interview process. In fact, I'd say the arrogance and apparent growing bureaucracy throughout the HR process that is in Google seems to be almost such a turn-off that I don't think someone could get me to work at Google even if they paid me some grandiose amount.

That would be a great article Steve, the arrogance of Google. I'm not trying to troll or be flamebait and I don't mean everyone or even you, I love your blog & writing style but it's impossible to ignore this elephant in the room.

8:36 AM, MARCH 13, 2008

mike said...

Jeff,

Why not? It seems to me that Google is in an interesting position that only a few companies in the past could claim. That is, they are the desirable place to work in tech, probably the most desirable. I haven't seen the numbers, but I have to imagine that there is a mountain of resumes piled in the HR department. If that is the case, then why shouldn't they be picky? It is a win/win situation as far as I can see.
-m

8:50 AM, MARCH 13, 2008

yogurtearl said...

Mathematics: A Discrete Introduction is a good book on discrete math.

8:55 AM, MARCH 13, 2008

Cosmin said...

Just a minor issue, but I had to get it out: The guy's name is Dijkstra, not Djikstra.

Otherwise, thanks for the great write-up.

9:10 AM, MARCH 13, 2008

Pablo said...

My previous company hired a guy who has mad whiteboard skills and couldn't code or design for shite.

Um, if you select for whiteboarding skills, that's what you're going to get.

If I ever hire an employee the interview process is going to consist of putting him/her on a computer with his/her favorite editor and go from there.

Why not put your candidate in the environment they're going to be working in?

Who the hell codes on a whiteboard with hyper-annoying misguided uber-geeks staring at them over their shoulder.

I'll be avoiding the Google interview process like the plague. Thanks.

9:58 AM, MARCH 13, 2008

mike said...

Pablo,

That's just the thing. You will *never* be able to properly simulate that potential employee working in your environment. Sitting them in front of a computer no more shows their propensity for working in your company environment than making them use a slide rule.
-m

10:08 AM, MARCH 13, 2008

Tim said...

Sitting them in front of a computer and writing code is a lot closer to what they will be doing on their job than standing them in front of a whiteboard and writing code.
Here at GHS everybody who comes on-site is expected to write a small but interesting program, from scratch (about 4 hours of time). I think that's a hell of a lot better than writing code fragments on a whiteboard, and it really isn't all that hard to administer such a test.

Tim

10:19 AM, MARCH 13, 2008

Michael said...

Note to Self: no need to apply for a job at Google. Your process is filtering for CS-bots, not necessarily top-quality programmers or engineers. The best programmers I have worked with with have either walked out on your interview or been asked to leave. Most didn't have traditional CS degrees and they couldn't care less about Big-O. They knew when they hit a problem that required more study and tools to be brought to bear, but the key is having the skills to know what you need to know and the ability to go seek out that knowledge. And graph theory? Please - get over yourself.

10:49 AM, MARCH 13, 2008

eightoclock said...

I would just like to add a suggestion: do some company-specific technical reading. This is especially easy when preparing to apply at Google, which has a few white papers out there. Reading up on map reduce, and being able to ask some questions about the paper (and talk about how I had done a distributed process at another company) might very well be what got me in.

And, guys, don't blame Steve for what Google's interview process is. He didn't say it's optimal, he's just telling people how to prepare for the terrain. The process appears to screen for CS-bots, as some of you say, and I'd like to see more questions asked about process -- but since they likely won't be, the burden is on you to just mention as much of that stuff as you can while answering the technical problem. For example, scribble some quick Javadoc on the whiteboard while saying "in real life this would probably have to be exposed in a user manual", make a quick stab at writing unit tests, etc.

12:57 PM, MARCH 13, 2008

Steve Yegge said...

> And graph theory? Please - get over yourself.

> I would attack X itself, except that I do not want to pick up a book and figure enough out about it to discredit it. Clearly I must yell a lot about how stupid Stevey is so that nobody will listen to him!

1:27 PM, MARCH 13, 2008

Jurgen said...

Why I Would Never Hire Steve Yegge:

I'm quite sure that less than 10% of all software developers in this world are able to understand *big-O complexity*, *n*log(n) sorting algorithms*, *hashtable implementations*, *graph theory*, *n-ary trees*, *NP-completeness*, *mutexes and semaphores*. But Steve states that (for him) this is all basic knowledge, and that his requirements for candidates are not much different from those of any other software company. Now, if this was really true then 90% of the software developers in this world would not be able to find themselves a new job at this time, if they needed to. As this is definately *not* the case -- millions of them are changing jobs every year, despite all their shortcomings -- Steve's statement is evidently false. I know companies that hire demented trolls only because they look a lot like software developers, and they know which side of the computer they need to bang with their club. (And in my own interviews, I prefer socially-aware software engineers with common sense over uebergeeks from outer space. But that's another story.) Therefore... *1 point off for tunnel vision, distorted sense of reality and false reasoning. I can't abide know-it-alls.*

In my opinion, building software is about delivering value to customers and making users happy. -- Oh, and it would be nice if you enjoyed doing that, but it's no requirement. -- Software engineering is so much more than just knowing your "basic" algorithms and data structures. It not only entails Construction, but also Requirements, Design, Testing, Maintenance, Configuration Management, Project Management, Process Management, Tools, Methods and Quality. According to SWEBOK, the knowledge area of Construction -- for many of us, including me, the most enjoyable part -- accounts for only **1/10th** of the body of knowledge for a software engineer. Many of us need that part to enjoy ourselves.

However, we need the other **9/10th** to make our customers and users happy. *Therefore... 1 point off for forgetting whom you're building for.*

I hate it when people talk to long. The KISS principle is just as valid for blog posts as it is for code. If Steve's blog writing is any measure of the volume of code he writes, then I understand why Google is so busy building these super server farms here in my country. -- *Therefore... 1 point off for not knowing when to stop.*

Anyone who misspells the name of one of the greatest thinkers in the history of our Software Engineering discipline must be turned down immediately. No matter how many sorting algorithms he knows by heart. Now, I wouldn't mind if people accidentally referred to Stevey Yiggo. That would be understandable. But come on, misspelling Edsger W. Dijkstra is quite something else! -- *Therefore... not 1 but 2 points off. Because Dijkstra was Dutch, just like me.*

That's five points in the negative, mr. Yegge. Thank you for coming, that will be it. Don't call us, we'll call you. Please leave the markers on the table. Thank you.

Noop.nl

2:10 PM, MARCH 13, 2008

Daniel Martin said...

One other technical X I would add to the list: know how the web works down to the level of IP packets.

Fortunately, this is a technical area that there's a very easy home test for, so you can assess yourself well in advance of your interview and read up on what you don't understand. Go get a packet tracing tool, such as wireshark. Capture all the network activity that happens when you ask a freshly opened browser window to go to www.google.com (just the front page, not counting what happens when you start typing).

Now make sure that you can explain every single packet you just captured in detail. This should cover DNS lookups, the three-way TCP handshake to port 80, HTTP headers on both the request and response, etc.

2:51 PM, MARCH 13, 2008

GS said...

The description "software engineering" is starting to cover too broad a field and seems to be the main cause of confusion for many here (evident in some comments above).

If you're building software from scratch or very near the metal, rather than very high level 3rd party bespoke business agile solutions, of course knowing graph theory is going to be more useful than studying stuff like requirements and configuration. they're completely different areas and it's just disappointing to read confused defensive comments by some seemingly insecure "offended" types here.

Perhaps worth writing a post some day about different layers of the software cake sometime to clarify such things.

I don't have the credentials for Google, and nor do I intend to work on that kind of software anytime soon - most of my work is with high level languages/frameworks and problem solving with those, but *still* found your content valuable.

Would have thought fewer words are better to describe most things well, but your blog proves otherwise. don't be put off, ever, and keep up the great work.

3:15 PM, MARCH 13, 2008

Pelkor said...

"I'm curious, why Google?"

Start on this page and work your way down...

http://www.google.com/support/jobs/bin/static.py?page=benefits.html

3:28 PM, MARCH 13, 2008

qwzybug said...

Jurgen said: "Anyone who misspells the name of one of the greatest thinkers in the history of our Software Engineering discipline must be turned down immediately."

You should Google around and see what Dijkstra said about "software engineers" (hint: he put it in scare-quotes, too).

It's also odd that you don't think that programmers need to know big-O notation or *n* log *n* sorting algorithms (!) but still have kind words for Dijkstra. Another hint: the kindness would not be mutual.

If you don't know *why* you're doing what you're doing, you're just an Eclipse macro that happens to breathe. Educate yourself.

3:39 PM, MARCH 13, 2008

greytrench said...

I've only just started programming, and I've gotta say, that's an intimidating list. Still, speaking as the neo-est of neophytes, it looks like a lot of fun stuff to study! Thanks for the ideas!

5:26 PM, MARCH 13, 2008

Matt Blodgett said...

Steve,

Are there software engineer jobs at Google that aren't so academic in nature?

For example, do the people who work on Gmail have to memorize all of that "baseline" stuff you listed?

Is there room for smart software engineers at Google who like working at a higher level?

I'm genuinely curious to hear your answer. Google seems like a great company to work for, but I'd sooner be bashed in the face with a sledgehammer than have to commit all of that low-level nastiness to memory.

7:35 PM, MARCH 13, 2008

Cooper said...

My impression of Google is that they can afford to hire only the '10' employees. Can't really blame them.

I like to think of myself as maybe a 9 tops, so no free lunch for me!

The important thing I think is that if someone understands the fundamentals, whether its graph theory or an http session from layer 1-7, they can figure out the high-level stuff no problem.

I do think they are hurting their bottom line in the long run though, as hiring only one "flavor" of employee can make for dull products. Compare/contrast with Apple for example.

I'm also willing to bet that the seeds of the next Google are being sown within their own walls as we speak. Lots of smart people with money in close proximity tends to make new companies.

Btw, from what I hear from people I know actually at Google, its a very mixed experience. The senior folks (pre ipo) are, not surprisingly, pretty happy. The worker bee types tend not to like it so much.

8:40 PM, MARCH 13, 2008

Mark Harrison said...

Hi Steve, great post as always.

I really like the Grimaldi "Discrete and Combinatorial Mathematics" book. This was the book (along of course with the discrete math class) that made me really feel I had a Vocation as a programmer.

A lot of people will recommend Concrete Mathematics by Knuth, but went through that and preferred the Grimaldi.

For the people questioning why this is useful in interviews... it's sort of like auditioning for a job in the classical music field. You'll be asked to play not just some set audition pieces and some sight reading, but also scales, arpeggios, etc.

If you can play an F minor arpeggio and an E major scale confidently on demand, the reviewer can probably safely make some assumptions about your basic musical knowledge and skill. Of course, you might be able to do those and still suck, but hopefully the rest of the audition will take care of that.

1:47 AM, MARCH 14, 2008

oligophagy said...

Hey now, quicksort isn't O(n*log(n))! Are you sure you work at Google?

3:47 AM, MARCH 14, 2008

Gwenhwyfaer said...

Those people criticising Steve or Google for being "too academic" (in requiring *basic knowledge of the domain*) are merely exposing their own ignorance - or worse, inadequacy. Data structures and algorithms are the foundation upon which solid programs are built; those who believe that other people are supposed to worry about them are condemned to live out their days are mere interior decorators - or worse, to build unintentionally collapsible buildings.

Nonetheless, there are legitimate reasons to shy away from Google as an employer, not least its size. Whether you believe that large organisations are simply unworkable, have a zero tolerance for bureaucracy, even relatively benign forms (*any* large organisation has to spend some effort making sure it continues to go in the same direction, which is what bureaucracy *is* - it comes with the territory), or simply can't stand to be in close proximity with any number of your fellow man (especially those fellows who might work at Google) - as it happens, I'm afflicted by all three - Google wouldn't be a good fit, and no long and exciting list of benefits or cool technical challenges (and believe me, I can see how cool they are!) can offset that.

Now, if you'll excuse me, I have to go and refresh my ADS knowledge :)

4:02 AM, MARCH 14, 2008

Michael Head said...

@oligophagy
Actually, Stevey didn't specify whether he was using an average case or worst case analysis.

Now, if he had said worst-case n*log(n), then sure.

Which actually raises a question... what to do when the interviewer is wrong?

I heard of one interviewee that was asked how to lazily instantiate a singleton in Java without locking every single time the getter is called (the infamous "double checked locking" issue). This is effectively impossible, and the interviewee informed the interviewer about the problem with the question. The interviewer then proceeded to argue (wrongly) that DCL is a perfectly proper solution...

4:18 AM, MARCH 14, 2008

Noam Mor said...

"For example, do the people who work on Gmail have to memorize all of that "baseline" stuff you listed?"

I'm intimidated by the amount of complexity in Gmail, as i think about it. Like everything else in Google, it is a distributed application, with millions of users, each of which has gigabytes of emails stored. Emails are interconnected between conversations and between users; Conversations are interconnected between users; Attachments are interconnected between emails and between users; Everything is stored in some sort of a grid, possibly in an efficient manner (no need to save the attachment for both the sender and the receiver).

What have we? A graph of email conversations modeled in a distributed storage system. All the text is scanned in favor of presenting relevant ads, which is not the easiest thing in the world. If any part of this gigantic system is implemented inefficiently, the whole thing becomes much too slow for use by millions of people. And let's not forget that the UI and the server/UI communication are written in a subset of Java that is compiled to Javascript, which I am sure is a method developed specifically for Gmail. So all in all, I'd say yes, the people who're working on Gmail do need to be proficient in Computer Science. It is just not possible to create such an application without deeply knowing what you are doing.

5:51 AM, MARCH 14, 2008

Shahms said...

From a fellow Kirkland Googler:

An excellent post that I highly recommend Google candidates read before their phone screens or interviews. I can't stress the importance of hashtables or big-O enough. Whether or not the topics are actually important, they come up regularly in interviews often in the same question: "So, you've given me a O(n*logn) solution using a tree, can you come up with a better solution?" should serve as a huge hint.

I also want to echo the extremely high false negative rate and it's arbitrary nature. It's true and it's frustrating from both sides of the table (a candidate several interviewers may like can still get rejected), but dust yourself off and try again.

One minor quibble though, depending on the job for which you are applying you may be asked to rate yourself in several areas. This is, generally, to alleviate some of the problems with getting an interview quizzing you about Java when you're more familiar with C++ (or Python). It's not perfect, but interviewers typically will take that into consideration before asking questions.

Related to the point about asking questions, just demonstrating some enthusiasm goes a long way. There's often not much you can do about this, if the interviewer is asking you particularly inane questions, but in a typical interview there should be at least one question that can be taken in an interesting direction.

8:39 AM, MARCH 14, 2008

Frank Pape said...

"Is there room for smart software engineers at Google who like working at a higher level?"

From another Googler, I say yes, absolutely. But you'll still be expected to know the basics. It's near impossible to be a really good software engineer at *any* level if you don't understand why to use one data structure or algorithm vs. another, even if you're not implementing them.

Google even expects product managers to have a good foundation in this stuff, so you can bet you'll be expected to for any kind of programming position.

One tip I haven't seen, that helped me a lot before interviewing, was to read Wikipedia. The articles on various data structures and algorithms are great, and free. Read them!

If your language of choice is Java, study the source code of the Collections classes. Understand how *they* implement a HashMap, and then you'll be able to do it too. You might be surprised at how simple most of that code is.

10:31 AM, MARCH 14, 2008

andypatrick said...

*I'm quite sure that less than 10% of all software developers in this world are able to understand big-O complexity, n*log(n) sorting algorithms, hashtable implementations, graph theory, n-ary trees, NP-completeness, mutexes and semaphores.*

Really? Then more than 90% of coders in the world aren't up to the job. Unless you count "coders" as someone writing HTML for a webpage.

Out of that stuff, I'd say knowledge of hashtree implementations isn't critical so long as they're aware of the pros and cons of hashtree (and other container) usage... that's about it. For me to hire someone, they'd have to have at least the ability to hold a conversation about everything else in that list.

How can anyone claim that Big-O notation is not important??? Unless your datasets never grow above a few KB maybe.

I mean, I wasn't bang alongside *everything* Steve just said, but seriously... your comment is like saying "people don't need to know how to program to be programmers". Um... they do, y'know. They really do.

12:36 PM, MARCH 14, 2008

Benji said...

I intereviewed at Google a few years ago and had a great time. I wasn't offered the job, and I can choose to chalk that up to either the false-negative problem or the fact that I blew a few questions.

But one thing I'd like to comment on from the article is the recommendation to know one programming language *really* well, "preferably C++ or Java".

That should read "preferably C++. We will allow you to write code samples in Java, but we will silently dock you points for not knowing C++."

One of my interviewers asked me to reverse a String.

I wrote the code to get the char array from the String object and then I reversed the char array (in place), passing the reversed array into the constructor of a new String.

The interviewer frowned and asked if there was anything wrong with the code. I looked at it for a moment, and then stepped through the lines, debugger-like, on the whiteboard.

No, there was nothing wrong with the code.

The interviewer was annoyed, finally telling me "you forgot to null-terminate your array".

In reply, I said "actually, in Java, a String object is like a C struct containing both an array of characters, and an integer indicating the *length* of the string."

Now he was really annoyed. "Yeah. But you didn't put a NULL at the end of the array. You have to put a NULL at the end of the string."

I could sense his annoyance, and wasn't quite sure what to do. But I trudged on, explaining "Well... since the String object knows its own length, it doesn't need to have a null at the end. Java Strings are not null-terminated. They're like Pascal Strings."

He paused for a second and then shrugged his shoulders dismissively. "Yeah, well I never really knew much about Java."

The moral of the story is this:

Java may be a decent programming language, but you loose points for writing code samples in Java, since many of your interviewers probably consider it a lesser platform than C++ coding.

12:44 PM, MARCH 14, 2008

Chris said...

Thanks. Guess I have a lot to learn. Because I don't know all that stuff.

But I figure I'm ok right now because I'm very new, still in High School and just concentrating on completing the AP test and graduation. When I get to college I can work on the more advanced stuff.

Good to know that Google accepts Java, even though it has its problems (I dislike having to call 4 different methods from one object if I need a whole bunch of variables. Really needs to be a way to return a bunch of stuff at once besides setting up an array, because that can get weird).

2:47 PM, MARCH 14, 2008

mlvanbie said...

@oligophagy, @michael head:

The running time of quicksort depends on your pivot element selection. If you use a linear-time median selection algorithm then quicksort is O(n lg n). I believe that you can read about the algorithm in CLR's *Introduction to Algorithms*. However, their is an inefficiency in their pivoting code (at least in the first edition). When is it bad? What running time does it give in the worst case? One of my interviewers had a similar problem in a question that he asked me to code, which I found.

If you like solving problems and using your knowledge of algorithms, then Google interviews can be fun. I suggest ACM programming contest problems (easier ones from the finals and some regionals, for example) as good sources of practice material. You could easily be asked similar problems during an interview.

9:50 PM, MARCH 14, 2008

Michael Head said...

@mlvanbie

Sure, you can tweak the selection algorithm and guarantee an n log(n) runtime, but you kill your performance, so nobody does it that in practice. That's why the textbook answer is that quicksort is n log(n) in the average case, and n^2 in the worst, but everyone picks it over mergesort because its constants are so much lower in the average case.

Special pivot selection is more of theoretical interest -- though I agree it is interesting and worth being aware of.

11:12 PM, MARCH 14, 2008

Ben said...

@Michael Head

>lazily instantiate a
>singleton in Java without
>locking every single time
>... impossible.

Wrong. DCL will work in the Java5 memory model if the reference is volatile. Even then, its an ugly approach. The better approach is to use a static inner class as a holder of the instance and rely on the fact that class loading is synchronous.

In fact, stating ignorance is a fine answer. Stating something is impossible is always a wrong answer, unless mathematically provable.

1:37 AM, MARCH 15, 2008

Cooper said...

This is really a great discussion.

Can anyone @Google provide a short list of either a "Google Library" of recommended texts or at least bulleted list of Wiki articles? I have about 60% of a CS education under my belt and would love to really dive back into this stuff.

On a completely unrelated note, I think there is an elephant in the room here. For all the supposed brainiacs, no one has supposed that there may be more people qualified to work @Google then there are positions available. So no surprise there are false negatives.

Google isn't the alpha and the omega. They've already IPO'ed, so you ain't gonna be googlenaire anytime soon. And trust me, once you realize you spent your life working overtime in order make someone else rich, you will realize those dinners weren't 'free' after all.

11:30 AM, MARCH 15, 2008

Michael Head said...

@ben

Thanks, that is true, I should have written the problem as "without any overhead" rather than "without using synchronized".

11:37 AM, MARCH 15, 2008

Michael Head said...

And, BTW, the point of my comment wasn't about the specifics of DCL in Java, but questioned what one may do when the interviewer is actually wrong on a point.

Do you just write off the interview at that point, accept an incorrect premise, or try to correct the interviewer?

11:49 AM, MARCH 15, 2008

Michael Head said...

This is probably a useless comment, but I'm hoping this will stave off anyone giving me more guff for mentioning Singleton vs. DCL.

Sure, I could have been more specific about the anecdote, but we all understand that there are issues with respect to double checked-locking. I misposed the supposed interviewers question. But that detail isn't critical to the point. The point of my story was to imply that the interviewer wasn't aware of those issues.

And my question was, what does a hypothetical interviewee do in that situation?

Don't let the anecdote distract from the message! (I say this knowing exactly how I'd respond to an imprecisely/incorrectly posed question).

12:38 PM, MARCH 15, 2008

ravehanker said...

what's wrong with trying to sort linked lists?

2:57 AM, MARCH 16, 2008

craig said...

Discrete Mathematics with Applications, Susan Epp.
That's the book we use at our univeristy in London for alot of discrete math stuff in my Computer Science degree. Highly recommended.

3:00 PM, MARCH 16, 2008

Rod said...

Positively invaluable, thanks Steve.

It's nice to hear this sort of feedback from someone who made it through the google interview process.

I applied about a year ago but didn't get through. I wrote a post about my experience on my own blog here: http://www.nomachetejuggling.com/2006/12/30/my-interview-with-google/

Your comments actually lend a lot of credibility to what I said, for which I thank you. I posted a recommendation that people have a computer science degree and a great understanding of big-O for a Google interview, and received a number of complaints for it. Having you say it makes me feel right in my suggestion. Not necessary, but definitely extremely helpful.

I decided I wanted to reapply to Google after a year the instant I was turned down, and I've been working on my skills in preparation for it. Hopefully I'll work up the nerve to try again soon: your post definitely gave me a lot of hope. I had no idea so many people got in on their second or third tries.

Anyway, thanks for the post. It was both helpful and inspiring.

3:39 PM, MARCH 16, 2008

anil said...

Thanks for the post Steve, I'm reading your blog recently and I found them very useful and fun.

11:31 PM, MARCH 16, 2008

Adam Byrtek said...

One potentially obvious tip: when you are having a phone interview, *always* use a hands-free or a speaker for your phone. Having both hands free makes you much more comfortable and allows you to make notes, draw diagrams and/or just scratch your head easily.

6:10 AM, MARCH 17, 2008

Sam said...

Blogger Jurgen said...

"Anyone who misspells the name of one of the greatest thinkers in the history of our Software Engineering discipline must be turned down immediately...."

"I hate it when people talk to long."

How many points do you take off for spelling a word a simple as "too" incorrectly?

How many points do you take off for re-posting your comment trolls on your blog in order to impress the people you think are friends and glean a little spotlight off of someone who is obviously far more knowledgeable and talented than yourself?

Great post, Steve. Thanks for the tips.

Also, finding out that you also play guitar makes me wonder if you do everything I do a honkzillion times better than I do it.

9:19 PM, MARCH 17, 2008

Quoll said...

Following your quote:
"Gosh, we sure wish that obviously smart person had prepared a little better for his or her interviews..."
But if they're obviously smart, shouldn't you be considering them? If they just had to answer questions the questions correctly, then why bother with the interview? Why not just give them a quiz?

I appreciate the argument that their effort in preparing indicates a work ethic that you may be looking for. On the other hand, it ends up being like college: people cram for the "exam" with little hope of long term retention.

Personally, as an interviewer I'd much rather see what a person is capable of day-to-day. If a person *normally* knows the difference between a B+-tree and a B*-tree, then that tells me a lot about the kinds of things they find interesting, and the sorts of things they may be good at. But if they studied up on it in the days before because they knew I might ask, then what does that tell me about them?

Maybe it's naive, but I used this approach when I interviewed at Google a couple of months ago. As an applicant I figured that if they didn't like me as I was, then I didn't want to work for them.

(But they did want me... which made the decision to turn them down a tough one!) :-(

7:43 AM, MARCH 19, 2008

Quoll said...

...and as a follow up, I should point out that I was just commenting on that particular line from your post, and *not* flaming the essay in general. It's a good post.

I suppose I wish that the applicants I see (and even me, to a certain extent) already knew most of what you wrote about without needing to study it.

8:07 AM, MARCH 19, 2008

Sony Mathew said...

Graphs and problem solving questions in general make for an excellent interview by themselves don't you think? Having candidates use various data-structures like Lists, Trees, HashMaps to solve such problems should be sufficient to gauge their skills.

Remembering exactly how every data-structure object or sorting method is implemented seems no-more relevant than knowing how an object and its methods turn into byte-code instructions. Abstraction layers exist so as to focus on solving higher order problems. Only if a problem is suspected in an abstraction layer would one need to dig deeper - a good problem solver would do so without hesitation.

11:43 AM, MARCH 20, 2008

Adi said...

Postback from http://hebrew.dotmad.net/archive/2008/03/18/how-to-get-a-job-at-google.aspx

2:13 PM, MARCH 20, 2008

Paul Tomblin said...

Last summer I had a series of 3 phone screens and a grueling all-day in-person in Manhattan. All in all it was fun and a pretty good learning experience, although it was pretty obvious by the second or third hour of the in-person that they were interviewing me for a position that I wasn't willing to move to Manhattan for, and I said as much.

At the time, I figured I'd blown it big time, especially after I whinged a bit on my blog about the fact that they took 3 months to reimburse me for my expenses. But I got an email yesterday from a Google recruiter who wants to talk to me about a different position. So I guess that means I didn't totally blow it.

The thing is, though, that I'm a 46 year old self-taught programmer. I graduated as a Civil Engineer, but went into computers before I'd even graduated and never looked back. So that means while you guys were taking course on graph theory and discrete programming, I was learning pre-stressed concrete and sanitary sewer design. So I've had to pick up a lot informally. I can tell you that most of the good sort algorithms are O(n log n) in time and some are worse in time and better in space, and that Qucksort can blow up into O(n^2) in the worst case. But I can't name every sort algorithm in Knuth and what it costs in time and space. I can tell you that I know how to look up those things on Google or Wikipedia when I need to know it. Similarly, I've heard of Red-Black and B+ trees, but I couldn't describe how to write one from scratch. There's library classes and Wikipedia articles for that.

Anyway, given that my first phone screen is in a few days, and the book you recommend is out of stock at Amazon, do you have any fall back suggestions for how to prepare for this set of interviews?

5:13 PM, MARCH 20, 2008

George Pikoulas said...

Another good book is Algorithms by Sanjoy Dasgupta , Christos H. Papadimitriou , Umesh Vazirani.Check it out.It is easier to read than the Introduction to Algorithms.

3:23 AM, MARCH 21, 2008

Adam Hupp said...

This advice is very good. I prepared for my Google interview for about 6 months, reading the Corman Algorithms book and then, on Stevey's advice, the Skeina book.

The best thing I did was to spend time implementing various algorithms. I found that algorithms tend to have more tricky edge cases than the average code slog. Working through those helps prepare for the kinds of problems you'll get in an interview even if you don't get asked about those specific algorithms.

3:33 PM, MARCH 23, 2008

graham said...

But I have *people* skills!

For those lamenting the CSiness of this list, this prior post might help.

"They're absolutely right. You can be a good, solid, professional programmer without knowing much math. But hey, you don't really need to know how to program, either ... you might discover you're good at project management, or people management, or UI design, or technical writing, or system administration ..."

11:19 AM, MARCH 24, 2008

Jim said...

No one mentioned:

Algorithmics: The Spirit of Computing by David Harel and Yishai Feldman

This is one of the easiest to read books on Algorithms that I have found and it is written by a master in the field. It is not overly math based, but if you want to understand the concepts before the math is thrown in, it is a good primary (and it is relatively short).

9:29 AM, MARCH 28, 2008

indil said...

Jurgen,
I'm going to have to take 5 points off your comment since Edsger Dijkstra was a computer scientist, not a software engineer, the difference being he only cared about the mathematics of computer science, not requirements or processes. In fact, he didn't even own a computer until late in life because computers have little to do with computer science (read a bibliography of him to verify). And you're right--how could anyone possibly misspell a name with a silent 'J'? ;-)

2:04 AM, MARCH 29, 2008

Fred Ross said...

I am thoroughly amused at the Dijkstra discussion. Have none of you actually read "A Discipline of Programming" or "Predicate Transformers and Program Semantics"? For shame! He was very mathematical, he was very much concerned with the practical problem of developing software, and he had no respect for people who would put ten times as much effort into a hacked up solution instead of thinking for half an hour about what would make the semantics clean.

I'm even more amused that people are complaining about things like complexity classes and asymptotic complexity. I did math and physics. Now I work in a tuberculosis lab, yet I know this stuff. Come on people!

Hey, Steve, does Google need a mathematical physicist who knows how to use source code control and can culture bacteria? For that matter, how would you handle interviewing someone with that weird a background?

12:09 PM, MARCH 30, 2008

Jeff Brown said...

Heh. You forgot to mention that Google interviews are fun!

Nothing like ripping through a handful of gnarly algorithms in half a day with some really smart people.

Unfortunately I turned down the job offer for personal reasons. However, my interview experience was stellar and well worth my time!

Perhaps another day.

Meanwhile, I'll send these tips off to a few junior hopefuls I know here and there...

12:39 AM, APRIL 04, 2008

valentina said...

have you ever hired anyone who was significantly smarter than you?

9:16 AM, APRIL 09, 2008

F. Vargas said...

Too bad this won't help if the morons skip you because your resume doesn't list a school that's "on the map." I've been hacking code since high school in nearly a dozen languages and couldn't even get a phone interview. My startup will never deal kindly with Google for that reason alone, nevermind their Do No Evil policy is fragmenting.

2:33 PM, APRIL 09, 2008

thraxisp said...

I think we need vocabulary to differentiate between CS grads who talk about lambda, and ordinary muggles faced with a repetitive task.

Users are scared away by the concept of "programming," when their day could easily be improved by a 5 line macro or shell script.

10:02 AM, APRIL 12, 2008

Paul Tomblin said...

when their day could easily be improved by a 5 line macro or shell script.

My day could be considerably improved by replacing my boss with a 5 line shell script. Although 3 would probably suffice:

```
while true ; do
echo "get your timesheet in!"
done
```

10:59 AM, APRIL 12, 2008

lappet! said...

Looks really useful...am bookmarking it for future reference!

11:52 PM, APRIL 18, 2008

Steve Yegge said...

Wow... I wasn't planning on commenting, but valentina has asked a surprisingly deep and relevant question:

> have you ever hired anyone who was significantly smarter than you?

And the answer is... yes! all the time! Um, I think!

I _suspect_ they're way smarter than me, but it's often difficult for me to prove, because I'm not smart enough to know if they're bluffing.

This is one of the most fundamental problems with interviewing as part of the hiring process: how do you tell if someone is smarter than you? As a general rule, people *can't* tell, so they wind up hiring people who are subsets of themselves, and a team's bar gradually declines.

They sometimes get lucky and accidentally hire people with astonishing math or CS (or other) skills that nobody on the loop was qualified to gauge. But on the whole, the bar is always in steady decline.

This will be the subject of a talk I'm giving at Stanford soon. I hope it's controversial!

Valentina, thanks for the excellent question.

3:31 AM, APRIL 25, 2008

me22 said...

Regarding the "Big-Oh is useless" comment:

I'll admit, I'm never going to go out and prove the amortized complexity of a splay tree access. But I've done the ACM's ICPC (and similar) a number of times, and it got me into the habit of doing a (very) rought Big-Oh calculation to see whether the solution was plausible for the size of inputs that were expected.

This was actually useful for me just recently. I had reduced it down to a graph problem (and yes, everything's a graph problem), and implemented Dijkstra, but soon needed to extend it to negative weights. That leads to bellman-ford, but the complexity --- O(v*e) --- for it made be think, 'darn, that's too high'. I knew I had a DAG, though, so I researched a bit, and found an applicable variant that got it down to O(v).

The quick rule-of-thumb estimate saved me quite a bit of coding time.

11:46 PM, APRIL 27, 2008

~jess said...

So by "software engineering positions" you are excluding user experience positions (web developers) that mostly focuses on AJAX + Javascript + CSS?

Thank you so much for your article :) Hope to see you on the Google campus someday.... I hope! As for your closing comments "Real-world work makes you rusty." <- remarkably true.

7:34 PM, APRIL 28, 2008

Steven 'lazalong' Gay said...

Why reinvent the wheel again and again?

Said differently: Do you really ask a Chief how to build pans, knifes or an oven? Wouldn't it be better to test if he is able to select appropriate ingredients or use **bad** ingredients to make tasty dishes?

As most of us I studied how sorting, hashtable, trees and other statistics methods where implemented. And quickly realised that only very few mathematical genius would be able to ameliorate them hence I forget how to code them as soon as possible to leave place for useful stuff.

Such as which of the available implementations (Boost::graph, Poco, std,...) is the best in a cross-platform implementation, which one is thread-safe, which one is fastest, etc.

Anyway very informative post.
I wonder how much time you use you write those rants :)

PS: If I made interviews one of my question would be to use so-called "advanced" statistical methods to prove a relationship among data then change the chosen method parameters to *disprove* it!

6:40 PM, APRIL 29, 2008

Sivasankar said...

Hi Steve,
I have been following your blog for close to 3 years now, from the time I graduated with my Master's in CS and had enough time to kill :)
Awesome job! Please Keep writing !!
Am not sure if you would actually see this comment and respond, but, it would be great if you did!
I worked predominantly in Operating System research during grad school and developed an Assembler (Yes! Not a compiler and some folks still do write production code in Assembly Language :) in my first job, but, am a routing protocol developer in my present job.
I've never really learnt C++ even, though I've developed "programs" in C++, Java, what_have_you :) I wouldn't brag about my C++/Java knowledge ;-) To summarize I am reasonably good (maybe, really good)at C , but, it would be too much to claim I am as good at C++ and the likes without actually having done anything productive in the other programming languages. For the kinda jobs I'm interested in, I've never really seen anything other than C being used though archaic(basic??) ideas of OOP do feature in the code base.
Phew! Atlast!! The question I wanted to ask.
Does it make sense to start working (learning)on a programming language like C++ or the likes just to broaden my repertoire ? Or with future jobs in mind ? Given a choice, I would like to learn Lisp the right way and invest my time in it!

Looking forward to your response.

Thanks,
Shiva.

3:56 PM, MAY 14, 2008

Shauna said...

Most positions say degree required. Is this a hard and fast rule or is equivalent experience also considered?

1:17 PM, MAY 30, 2008

dankamongmen said...

I love (and find myself eerily evoked by) your summary of Class A Interviewers; *et tu, Yegge*? Indeed, see a set of interview questions I published yesterday, from the Reflex Security days. LISP, Strassen multiplication, and TCP state machines, all to the rousing beat of the Book of Revelation! Grand ol' times.

Solid general interviewing is just as difficult as solid general education, and just as likely to be implemented piecework by specialists.

As an interesting case study in resumes, here's mine; I've already turned down the big G twice (for SRE), though =].

As always, Steve, thanks for partying like a blog star and bringing the Good Word; your dynamic languages presentation was thrilling, and a fine introduction to modern language issues for folks slacking on their λtU.

9:16 AM, MAY 31, 2008

dankamongmen said...

oh, and:

I see plenty of people have already mentioned *Concrete Mathematics*, so I'll spare that reference; I agree with all of its exceptional (well, for non-Knuths) rigor, dazzling examples and asides and utter unapproachability without some warming-up exercises. For those who enjoy that one, I heartily endorse Gries's A Logical Approach to Discrete Math, from the always-formal, always-authoritative *Monographs in Computer Science* series. It's a strong pair with *CM*, but brings one along a bit more gently.

The vast majority of DM texts seem utter rubbish.

1:34 PM, MAY 31, 2008

devlan said...

How do I ask an employer if they are finished with the hiring process woithout sounding to pushy. I need to move to the new town and get an apartment pronto

8:43 AM, JUNE 12, 2008

user24 said...

same principles apply for oxbridge postgrad interviews, just so you know. (I did terribly at the interview - wish I'd read this blog first!)

5:33 AM, JUNE 16, 2008

Michael Head said...

Question about Google: is there some sort of pecking order among different groups within Google? Are there some groups that are viewed as doing run-of-the mill development vs. those doing maybe deeper stuff? I see things like "Internal Applications", "Google.com Engineering", "Site Reliability Engineering", "Engineering Tools", "Systems Infrastructure", ... and I'm wondering which may be more (or less) challenging.

2:21 PM, JUNE 17, 2008

FF said...

"Most positions say degree required. Is this a hard and fast rule or is equivalent experience also considered?"

Considered, yes, but you can expect degree+no experience to trump no degree+experience. They're doing things that nobody else has ever done, after all...

1:27 AM, JUNE 19, 2008

Cooper said...

"They're doing things that nobody else has ever done, after all..."

Like what? I can't think of anything innovative from google besides pagerank.

7:02 AM, JUNE 19, 2008

Yavar said...

Great Great Great Article. I work for an Internet startup and I very well know the importance of Big- O Notations, Binary Trees, Sorting Algorithms, Hash Tables. Without data structures and algorithms one really cant survive in an Internet Company.

Thanks again for the nice blog.

12:52 PM, JUNE 25, 2008

Rahi Jain said...

very helpful... for me it came just for right time... i m a grad student with one year left for completion. many answers were answered for me and i read the whole 104 comments which i rarely do while reading blogs... very thoughtful and helpful article.

Thanks steve for this...

6:02 AM, JULY 02, 2008

vipin sahu said...

thanks a lot steve for such an awesome post, i am a newbie in the field of data structure and algorithm
and not exactly know which one is for just reading and which one is for coding i always thought that the bigoh notation are not much useful as per as other data structure but

after reading this blog i understood that BigOh is just not for reading :)
thanks a lot one again great job ....

12:10 PM, JULY 03, 2008

lamb of god said...

Hey Steve,
It was a real eye opener to read all of what you had to say. I am not an CS Engineer, i studied English Literature but got hooked on programming, i have been writing code for a couple of years now, its a big shift but i love what i do and i would love to work for Google. I was lucky enough to work as an intern in some of the best research organizations so i got my basics right. I did a course in abstract algebra and most of coding is Math related. I don't know if i stand a chance to work for Google but do let me know if someone with a non engineering background would be considered at all. Thanks

9:32 AM, JULY 09, 2008

FF said...

So, what job at Google are we talking about? Does it mean anything that Google's recruiting rates started dropping dramatically around the time this article was published? :-)

12:05 AM, JULY 18, 2008

Nick said...

Hi Steve

I've just spent the past hour and 1/2 reading your article and all the comments. The article was excellent and the comments were very entertaining. I logged into my library and put a hold on a couple of books that were recommended here. Yes, I would love to work at Google.

Well, I have nothing critical to say, so I'll fall in with the group of people who want to appear intelligent by writing something intelligent.

e = mc^2

:o)
thanks
Nick Fox

2:52 PM, JULY 18, 2008

eric said...

Thanks, it is a helpful post. Do you think Google has a bias about which school you go to ? (for engineering entry-level jobs, at least)

1:48 AM, JULY 24, 2008

SDC said...

Also, be sure at no point during the interview to say that though you've used Google religiously for years, you have yet to click on one of the ads. Man I could have saved myself some heartache.

6:53 PM, AUGUST 09, 2008

pcleddy said...

Not everyone has or cares to have either the memory for, not the intense interest in computer science that you have. Respectfully, some of us just want to have fun doing your job, while we have other interests in life. Believe me, there are things out there to spend a lot of time on. Plenty.

What is striking me funny these days, as I reflect on technical interviews, and after being in the industry for 10 years, is the lack of simplicity: why not just ask people what they have done, look at their code, go over it with them, look at their implemented architectures, go over it with them? Ask them where they were, who they worked with, who they are. Nobody ever asks me these questions, the human ones. Believe me, you are going to locate a lot more smart people that way. As it is, you are going to end up with a lot of extreme geeks. Maybe that works for you.

While I was at Amazon, I was astounded to watch technical interviews done by my peers, and listen to their philosophies, or lack thereof. It was like some big hazing episode. And, yes, I've experienced it first hand too. I've been insulted to have to answer questions that have and won't ever have anything to do with anything I will ever do. Like data structures for multi-dimensional tic-tac-toe, that one wins the cake, from the morons Speakeasy. I think I'm glad I got fazed out.

My claim to fame is I never asked an interview question that did not have to do directly with what the team was doing everyday. And I kept it simple. None of this is very hard really, sorry to disappoint those of you out there that think you're really some hot stuff. My mother could do what I do, if she cared to. Fact it, most people don't want to.

The truth is, you people don't want to look at what people have done, because you don't have the time. Or you just don't want to bother. And you can't think outside the box and be real. So what do you do? What's been done to you: what your professors did. Same old sad story. Btw, there are other teaching and assessment methods out there. You might want to write a paper on that and get your head out of your algobook. No one cares.

2:55 PM, AUGUST 17, 2008

Paul Tomblin said...

pcleddy: I don't know what you think you'll learn by looking at the code somebody claims to have written. When I was younger and more naive, I hired somebody who had great experience, showed me some well written code, seemed to know what she was talking about. Then I had to fire her because it turns out that the code that she told me she'd written was done as a group project, and she knew about it because she'd helped do the presentation on it, but she couldn't code for beans. As a matter of fact, she probably was worse than your grandmother.

Maybe you're happy as a boring code grinder, but I'm a smart guy and I want to work someplace where they'll throw new original problems at me and I'l produce new original solutions.

3:14 PM, AUGUST 17, 2008

Y!avar said...

PCLEDDY: Do you think the Google search and relevance algorithms, gmail spam classification algorithms, google personalization algorithms, google video search can be created by people who are not aware of Big-o Notations, Kanpsack, Travelling Salesman problems, Hamiltonian Cycles or by seeing the code that people have written ? :)

6:28 AM, AUGUST 18, 2008

Y!avar said...

I work for an Internet startup and I think we use all these technologies mentioned by Stevey. For example hash tables, N-ary trees in Boolean Query Evaluation, Discrete Matchs(especialy probability and all) in Bayes Theorem which go in making of spam classification algorithms, tries data structures for so many things like auto completion and so on...dynamic programming in calculating word proximities through longest common substrings.....if u need more example contact me @ yavarhusain@gmail.com. NO OFFENCES MEANT :)

7:00 AM, AUGUST 18, 2008

Dr. Jay said...

I have an interview @Google today, and based on what I read here, I am set to bomb terribly...

Ill be positive anyway

7:53 AM, AUGUST 18, 2008

Paul Tomblin said...

Dr. Jay:
Don't sweat it. I've been through several rounds of interviews at Google, and I found them fun and challenging. I got the impression they don't want to you memorize the algorithms book so much as you can think your way through a problem. Remember to talk aloud as you think through a problem - they'll drop hints if they think you're nearly there.

8:03 AM, AUGUST 18, 2008

Anthony Mowers said...

I share very much the same feelings expressed by Brian Di Croce in his comment.

I was uncertain about whether I'd be a good match for google - or google would a good match for me - and this blog provided me with that extra bit of information.

5:45 PM, AUGUST 19, 2008

Anthony Mowers said...

The article was very useful for knowing and the google interview process. It is very much appreciated. Thanks again Steve.

When I read this blog it prompted me to spin around in my office chair and look at my bookshelves of software engineering books that I've collected over the last 20 years of work.

I have one book on data structures and algorithms. What about the content of all those other books? It makes one wonder. Know what I mean?

How is it that the interviewing process has become so homogenous at Google when the discipline itself is so diverse?

8:40 AM, AUGUST 20, 2008

Preston said...

First thing's first.

Steve, you are to be commended for putting together a thorough and useful article (the comments are good errata, so they help as well) on how to refresh your skills as a software developer so that you can interview well in a day and age where the job market is tight and lots of us have watched as our knowledge in these areas has atrophied. I will personally apply some of this. I'm certain of that.

I think it also has to be said, however, that I find it a bit sad that this is the state of software development today. I've worked in the industry for 11 years. Granted I'm generally solving business problems and often UI problems, so I'm working more with AJAX, Struts and things of that nature rather than complex data storage algorithms.

However, nothing I do on a daily basis, currently, requires me to know what you believe we should know. I just have to come right out and say that. I read constantly. I read fiction, for fun. I read the Wall Street Journal. I read the Economist. I read blogs on economics and world affairs and triathlons and all kinds of subjects. I bike, I skateboard, I do open water swimming and creative writing. I, frankly, don't have enough time for all things I want to do in my life. All the things I do to enrich my life and become healthier in mind and body.

Now I know the intent of this article was to provide a template whereby a software developer, especially one who is rusty at interviewing or maybe skipped the CS part, can be prepared and maybe even learn some things to make them better developers. Once again, you succeeded. However, I doubt I'll be picking up the data algorithms books anytime soon. My life is too short. Too precious. If I were to die tomorrow I would not regret that I'm "merely" a software developer and not an Engineer. I would regret that time I didn't take to talk to and hug my wife. I would regret that I didn't take dancing lessons with her. Or that I didn't finally learn how to play the piano, or learn Spanish and work overseas for a non-profit.

The list of things I want to do in my life that I haven't done is so long that discreet math doesn't even register. It's not even on the list. And maybe that means I'll be marginalized someday. I will have to sort this out for myself. But I felt it was important, amongst all this self-important talk about what a true programmer is, to remember that many of us are doing good work and doing just fine without knowing complex data structures by heart or discrete math. Many of us are working hard to enjoy our precious lives before it's too late.

Hopefully you receive what I'm saying with due respect and compassion. I don't mean in any way to say that my way is the "right" way. I simply mean to say that "Engineering" isn't necessarily a must have for working in this industry. Not always.

Lately, at the behest of a brilliant co-worker of mine, I've found myself reading books about process. Books like "The Five Dysfunctions of a Team". These books are terribly fascinating to me. Because at the end of the day I've never had cause to improve my knowledge of discreet math. Except for in the case of wowing an interviewer. However, becoming a good teammate, that's something I think we all should be working on improving upon every day. Those are the books I wish to rotate through my collection. And if they land me a job someday, then great. If they don't, then that's okay too. I've made my peace with who I am and what makes me valuable.

10:37 PM, AUGUST 21, 2008

Y!avar said...

Preston: You didnt mention the amount of time you wasted in writing this very demoralizing article with intense pessimistic thoughts :) I suggest this time you could have used in reading a data structure/algorithm book.

O My God. Why the hell do you dont think that Steve has written this blog for aspiring developers who want to get into Google and I think it may mean a world to some of the people to say "Been there done that" You can read economics do scating hug your wife etc etc and you can be at google at the same time.

I am sorry but It was not all related to the blog...Please avoid such posts to increase the space complexity of the Steves very nice blog...Do watever you want Preston but dont force others ... Thanks!!!!!

11:37 AM, AUGUST 30, 2008

Retro said...

It appears that there are two types of responses to your post: "its wonderful, thanks" or "why ask discrete math questions?".

A good answer to the second type is in order. Perhaps google does not want to hire a person who is and always will be a coder. May be, just may be, the idea is to hire people who not only can code, but can think beyond what is, to what can be. For this, the only test is whether you can think clearly, understand something hazy and complicated, and come back with a clear solution. Sometimes, this requires skills that are quite different from coding skills. These skills are probably closer to those needed to solve discrete math or other algorithmics questions.

Coders are indispensible. But Google can find coders by the dozen. What Google might be looking for is innovators+coders: people who can do their job during the day, but then in the evening, want to push the state-of-the-art a little further. That takes different skills.

Of course, it does not follow from my argument that what Steve says he asks in interviews are *the right* questions to filter out such talent. My argument supports asking questions that are removed from day to day activities of coders.

Google needs people who are not stuck in today, but who can think of cool new things that will exist tomorrow.

6:54 PM, SEPTEMBER 14, 2008

terminals-blocks said...

A Foundation for Computer Science" by Graham, Knuth, and Patashni

1:29 AM, SEPTEMBER 18, 2008

Pragmattica said...

Does anyone know any good examples of "disguised" NP-Complete problems? I've been reading about the most famous NP-Complete problems, and I would like to get some practice at picking them out in situations that I might encounter while programming.

9:16 AM, SEPTEMBER 22, 2008

PancharaKutti:-) said...

Thanks Steve for this wonderful post.your post was mostly focussed on getting into google as a software developer . I would greatly appreciate if you could let me know what all stuff we need to prepare when we are preparing for an interview with google in software testing?

simplebuzz@gmail.com

8:26 AM, OCTOBER 07, 2008

Benjii said...

Great article, thanks for putting it together!

Anyone know what the response time is like for applying to Google?

3:52 PM, DECEMBER 10, 2008

Gary said...

Thanks Steve, on your advice I looked up the The Algorithm Design Manual and there is now a second edition available.

9:36 AM, JUNE 24, 2009

Pamela Mishra said...

Wow. That was the most comprehensive 'How-to' for Google Interview. I dont have a CS background but I have been preparing for something that comes up in Google that suits my profile.

4:36 PM, JUNE 27, 2009

Jahanzeb Farooq said...

Thanks Steve for this very useful post. While I completely agree on the importance of data structures and algorithms, I still wonder why Google gives so much importance to ONLY these two areas. Software engineering is a very vast dicipline. How about other areas such as requirement analysis, design, writing high quality and human readable code, documentation, software development models and methodologies, managing complexity, object oriented methodology, testing, debugging, etc etc etc etc?

I recenlty finished reading Code Complete, one of the prominent books on software develpment since past some years. If I take it as an example, it does not even remotely discuss algorithms and data structures. How about the topics it covers in its 35 chapters, spanning 800+ pages? Doesn't Google think those topics are also important in software engineering? (if not equally important).
Anyways thanks for the valuable post.

5:54 AM, JULY 15, 2009

Kes said...

Thanks Steve!

I remember ignoring these advice b4 my 1st phone interview. Needless to say, I nearly went up in flames.

I focused a significant amount of effort on these areas in preparation for the next round.

Guess what? I'm currently on a summer internship with Google :)

3:03 AM, AUGUST 23, 2009

4lp]-[4 said...

Thanks Steve, that was very informative and helpful. I would like to get on it right now. So much to study ....

amresh2k at gmail dot com

PS: captcha image: calcul ....
mere coincidence???

9:49 PM, SEPTEMBER 08, 2009

Ed Smiley said...

Actually, the DCL question is JVM dependent. See the latest revision of Effective Java by Joshua Bloch for the gory details.

TIP:
Very typical though being an "expert" and getting it wrong. I strongly recommend testing any solution or in a standard text you find on those trivia and trick question sites. I found typos and misleading information on them. Think science. Test your hypothesis.

5:43 PM, OCTOBER 02, 2009

Akshay said...

Steve what would you recommend as the preparation outline for a testing position at Google?

9:38 PM, OCTOBER 07, 2009

crazzybouy said...

It was really amazing to read this whole blog and the comments. I have always wanted to be in a job which will give me lots of opportunities to solve original problems. But till now i have landed in jobs which are not very challenging. Especially in the systems kinda jobs, you do not get to solve algorithmic problems unless you are writing an OS on your own. At least thats what i have felt till now. Probably google is one such place. I have no idea. Never interviewed with them. May be if i can read up and digest some of the stuff advised by stevey, i can give a shot.

I do agree that algorithms, data structures are really the foundations for designing developing good software. But my disappointment is it is not always easy to find these challenges in the job you do. So it is also a good idea to read/think/solve problems in your free time (if you have), even if you dont go for google interview. Provided one is very much interested in computer science, it acts like food for your intellectual hunger.

11:13 AM, OCTOBER 08, 2009

Max said...

Steve, thanks so much for writing this. Used it as a prep guide for my interviews - algorithm design manual rocks, I used it with the author's other book 'Programming Challenges'.

Reading this post really helped me deserialize knowledge that was stored away in my brain from college, and made all the difference in the world.

I interviewed in Manhattan, and due to a flight delay arrived 4 hours before my interview instead of the night before. This was a nightmare, sleep deprived and barely operating on fumes thanks to copious amounts of coffee, without using this article to manage my preparation - I surely would have gone down in flames. (esp with regards to the pointers on long term and short term prep. Warming my cache 30 min beforehand with some graph problems really paid off.)

I write this on the same day I accepted my offer, thanks for providing me advice on how best to focus my efforts. Looking forward to engineering orientation!

11:37 AM, OCTOBER 12, 2009

ruvinitl said...

Thanks a lot stevey for your extremely important tips!
I'm a girl and a fresh graduate from South east Asia (Sri Lanka). I love coding and tomorrow I'm supposed to face an interview in a leading S/W company. So, I will keep these tips in mind.
Btw.. I badly want to go through "the algorithm design manual" book. But couldn't find it anywhere here. Somebody please tell me a link to free download a copy... :(

3:35 AM, OCTOBER 15, 2009

Su said...

Hi,
This is really very helpful.
Can you give me your mail address where i can forward you my resume !!
I really want to get into Google as a Software developer and i am trying since 5 months to

get that 1st interview call.

12:32 PM, OCTOBER 19, 2009

Sumin said...

I got a job interview next week and this was a great help! I just pulled out my algorithm book that I used last semester ;-) Thanks for posting this.

3:27 AM, OCTOBER 22, 2009

Saurabh said...

That was a great post. Thank you for sharing your insights on the process.

For someone wanting to get a hold of the basics, Schaumm series book on Discrete Maths can be a good starting point. For developers in India Graph Theory by Narsingh Deo is pretty good.

Can anyone point out a good book for practical applications of binary, n-ary trees, graphs?

10:59 PM, NOVEMBER 29, 2009

Gaurav said...

Hey Stevey,

I had applied to google long back, but still didnt get a call. Is google really choosy while calling in candidates or is it, there are no vacancies now.

Gaurav
(Mumbai, India)

11:27 AM, DECEMBER 02, 2009

Colt said...

Steve - great article! Someone needs to speak with someone in your Finance Department to write the same... just submitted my resume last week, keep your fingers crossed!

5:02 PM, DECEMBER 24, 2009

hans said...

Read your post, got the job. Thanks man :)

8:04 AM, FEBRUARY 15, 2010

mangaroo said...

FYI, I was told by the recruiter turnaround time from interview to getting hired (if hired) would be like 6 weeks.

I wonder if the test automation/tools/engineering positions cover this much territory as well. On the one hand, to very effectively test you need to know more than the developers or the application being developed, so that makes sense. But on the other hand, to also effectively test you have to be "not a developer" with a different frame of mind, which would be knowing a different set of things, so that would seem like overkill to know so much of both worlds. And to add to the "not be a developer" mindset, if you think like one for testing, you may miss the same bugs developers miss.

10:53 PM, FEBRUARY 16, 2010

Lauren said...

Thanks Steve - I'm an IT recruiter and this really helped me prep my candidates. I have a number of clients who use similar strategies. I'd be happy to speak with anyone looking for non-technical interview tips or developers looking for new opportunities in the Boston area. I wish you hadn't stopped blogging - I enjoyed your others as well.

12:49 PM, MARCH 15, 2010

Vlado said...

"If you don't know why you're doing what you're doing, you're just an Eclipse macro that happens to breathe. Educate yourself."

Love the statement. Though Albert Einstein said "If you know what you are doing it is not research."
Are all jobs in google just hard core engineering? Working nights to meet deadlines?
Reading the blog was fun, reminded me days when I was lecturing that stuff at university (except exams, too much reading). But for all practical purposes I prefer not to remember implementation details. Library is not far, as long as I recall class of the problem.

But what about black magic, unsolvable problems solved in real time, NP complete solutions in real time?
The crazy science and research part? Is there some place for that kind of people in Google?

Graph coloring is used for register allocation in compilers for better part of 30 years, and it is still NP complete problem, just some clever heuristic is used to find close to optimal solution. And that one did not grow on a tree, it was found out by someone (Chaitin, Briggs).
Is Google interested in those kind of people, that do not know what they are doing most of time, but occasionally, the results influences whole industry for few more decades?
Current job offers seems to be mostly publicity and management. Surely fun in it's own right, but not a job for everyone.

2:34 PM, MARCH 26, 2010

like_indigo said...

It is really helpful post!
But how about interaction designers? Do they ask about the same things like algorithms and math for UX researchers or designers?
Do you know?

10:26 PM, MAY 02, 2010

MFM said...

Well, as for WHY the interview process is flawed in some companies...

If there is a person needed to do certain tasks, the HEAD (management) decides what is important. They are the ones who are responsible how the person performs and pay to this person. If every interviewer (who's not responsible for the outcome) puts their own criteria, it means the head is missing. Big companies can afford to hire on a random basis, something will eventually work out, while small companies can be killed by one wrong hire. Now, imagine a big company hiring a new CEO based on the same system of whatever every employee feels like.

12:34 PM, MAY 13, 2010

hvete said...

Hardehardehar... even your Grandmother could do that. Tell me it isn't a liability to be old and female in this industry.

I forgive you, because you're just so transcendently cool.;-)

So here’s the thing. If my daughter so decided, I could be a grandmother this time next year. Wouldn't that be exciting? Until yesterday, I'd never heard of Big-O. Most of my degrees are in linguistics. My last CS course was Mr. A's 8th grade intro to BASIC. And I was sitting quietly minding my own business, knitting socks or baking strudel or data mining or whatever it is grandmas do, and Google contacted ME. Go figure.

I taught myself C over 30 years ago, then C++. I implemented my first hash table in 1979 and my first trie in 1982. And I've evolved my own method of approaching NLP applications that has enjoyed some success. So, the interviewer recommended I check out this blog, and read a bunch of books on algorithms, and I'm thinking I might just do that.

So clearly Google doesn’t throw you out automatically just because you don’t have a Cal Tech degree in CS. I told the interviewer the unadulterated truth, and he was really nice. He didn’t say, “Oh. (long pause) Wrong number,” and hang up. Furthermore, I think Google’s success is undeniable, and I think it is clearly rooted essentially in competence, not marketing, and so clearly something about their hiring process works. I think it’s probably fair to ask that I get my BS in CS now after 30+ years in the industry. I probably would have been a better programmer had I done this long ago.

That being said, and having spent many years in many different venues (both high-powered, testosterone-infused and not so much), I’d be curious to know what my chances are of rising above the worker bee in Google. What percentage of your great programmers are female? What’s the male/female salary ratio? Why is that?

That’s an open question. I feel Google has a right to select their employees in accordance with any reasonable criteria, and this is reasonable. There’s probably a bit of that hazing thing going on, but that’s not the whole story. I have brothers, and I think the reason they became math/physics PhDs and I went soft and linguistical is GENETIC. I’m just a girl. I think my brothers would have an easier time acing the Google interview, not only because they are better trained in math, but for the same reason they’re likely to beat me at racquetball. There. I said it. Are any of you recent MIT grads out there as courageous as the grandma? Care to comment? Didn’t think so.

But I’m not going to curl up and die just yet... My brothers are the first to admit that they couldn’t have written my best code. And they would also admit if prssed that if they got their hands on my code now that its written, they could make it smaller and faster. For some reason, that particular problem doesn’t fascinate me nearly as much as it does them. I think the reason is genetic.

I’ve lived a creative life, and I don’t want to be reduced to a worker bee, even a very well paid one. But it remains an open question for me how we best make use of everyone’s talents. Google advertises the open, free, egalitarian, collegial atmosphere, where everyone has an fighting chance at being understood. Is it really so? If it is, then Google is even more remarkable than I thought.

2:06 PM, JUNE 19, 2010

Guennadi Vanine said...

I am shocked to see that so many people are reading the article of person who confuses:
- the interview with the exam on the unknown before it topics, or, even, with interrogation;
- skills with knowledge,
- working abilities with experience in taking tests
- work experience with having free time for training in puzzle and crossword solving,
- the finding of suitable worker according to the requirements of work/employer with personal ego satisfaction

Even more, the author alludes people that career success (employment) depends on buying unnecessary for work books, burning the time on training in puzzles, school math exercises and test passing, sorry, algorithms memorizing, instead of achieving the working results, successful projects, which Google interviewers ignore in hiring process.

All specialists in Google are of such intelligence? and with such writing abilities?
Or it is some kind of a competitors spoofing attack on Google?

7:10 PM, JULY 05, 2010

vi5in said...

Just ran into this post! Thanks a lot. I interviewed with Google about a 2.5 years ago at their office in Tempe (which, incidentally closed down). I didn't get in and I was disappointed. They told me that they were impressed with my skills and that I interviewed well but that they didn't have a position that fit my skill-set. I thought they were just trying to make me feel good until I talked to my buddy who works at Google - he said that you guys try to match up people with the jobs you have available and if you don't think the skill-set or area of concentration matches, then you don't offer the job.

But yeah, you were spot on. I refreshed my algorithms and data-structures stuff and that really came in handy during the interview.

1:33 PM, JULY 13, 2010

NVRAM said...

Stevey,

Not sure if you still read these posts, but I just wanted to say thanks for encouraging (me to) study before interviewing w/Google.

I actually ended up having a long weekend off just before the interview, so I studied and played in Java (most of the last year has been in C# and JS).

Anyway, it paid off. I start next week.

So thanks!
NVRAM

10:09 PM, JULY 14, 2010

NVRAM said...

Stevey,

I just wanted to say thanks for encouraging (me to) study before interviewing w/Google.

I actually ended up having a couple of days off (plus a weekend) just before the interview, so I studied and played in Java (most of the last year has been in C# and JS).

Anyway, it paid off. I start next week.

So thanks!
NVRAM

10:28 PM, JULY 14, 2010

Gaathi said...

Thank you very much for your tips. According to me, each and every tip of yours is important and useful for the interview. Seriously I like the way of interview which gives the candidate an other chance to brush up some important concepts. Thanks once again.

3:59 AM, JULY 21, 2010

Uncle Mikey said...

Steve -- just a note to say thanks for this article. Regardless of how my own upcoming Google Interview Adventure turns out, Skiena's algorithm book is already turning out to be "the book I should have read years ago" and will almost certainly make me better at my job!

8:35 PM, JULY 21, 2010

Edward said...

Thanks Steve! This is a really great post.

Does this apply to the Technical Account Manager position at Google? I'm wondering if Google will test TAMs with CS-centric questions as you've outlined here.

Good news is that I've got an upcoming on-site interview in Mountain View.

7:27 AM, AUGUST 17, 2010

Nitish said...

Thanks for the summary of everything. Definitely an eye opener for someone like me...

7:59 AM, AUGUST 25, 2010

Atul said...

Hi Steve,
I really want to work at Google.
I went for the on-site interview. Because of the good feedback of phone interview and a great first 1:1 interview, I took it easy for the 2nd and 3rd round of interviews and lost attention. I tried to catch up on 4th and 5th interview (and the 5th interviewer was saying I finished his question ahead of time). But I didn't get the job :(How to beg them for one more round of interview :(I am asking this because you said you did the same.
By the way your thin whiteboard marker rocks :)

10:38 PM, SEPTEMBER 02, 2010

Atul said...

@Jahanzeb Farooq
You are right that those software engineering skills are required. And those can also be asked as interview question.(e.g. project management need project scheduling with deadline, write an algorithm which maximize the number of releases in a month with the various constraints and limited resources(box, employee etc). Also reserving a meeting rooms is a form of graph coloring problem :).

So your code complete is not tough because it has no exercise. Get a good software engineering book with exercise in it. you can find everything is falling back to good data structure and algorithm selection.

So Google is asking the right question. If you have any doubt please read the previous sentence :)

10:52 PM, SEPTEMBER 02, 2010

Phillip J. Birmingham said...

My only quibble with your advice is that I only saw it Friday, and my phone screen is Thursday.

I suppose you can't be blamed for that, though.

1:32 PM, OCTOBER 25, 2010

A.J.S. said...

"Crap" - that's exactly what I am feeling like as I am writing this piece. I am not a God-sent CS major having the sole objective of saving this earth from people who don't know CS concepts.

While I did graduate with honors from a "Top 10" school, majoring in CS - I feel I am no where close (in CS terms, algorithms, Big Os) to most people who have put their comments here.

I have been working with C++ for last 5 years, but at a much higher level - mainly doing application development in C++ and VC++. Also, I have always worked in telecommunications domain, thus never got into situations where I had to deal with tons of data, search through that data, sort the data and do other "cool" things with it.

I have worked in 4 different companies - ranging from startups to mid-sized to Fortune 20 - in last 6 years (by choice, and willingly), and I must admit I have always been the "Rockstar" in the teams I have worked in. I don't know much in CS aspect of thing, really, but I am the guy who gets things done - by hook or by crook, or sometimes by fluke.

Now, I don't know what a certain Google recruiter saw in my resume, but I now have a phone screen scheduled with G next week. And I know already, I am totally going to get screwed. I am more of a "Software Developer" than a "Software Engineer" or "Computer Scientist". I understand and use object-oriented paradigm, all common design patterns, solid networking concepts - but hardly any core data structures like Graphs or Trees. Maybe Maps\Hashtables at times. So reading all this, I am already shitting in my pants, and I feel as if I am going to get raped come next Monday. Maybe I should just inform the recruiter that I am not in a position to take the interview, I don't know if that's rude or whether wasting someone's 45mins-1 hr is rude"r".

I don't blame Google for their interview process, the fact is they can afford it right now so why the heck not. All my life I wanted to work for Google, but looks like I just got a reality check... Pray for me guys, at least I can get a graceful denial...much better than people at Google thinking "what a pile of $hit this dude was"

8:06 PM, NOVEMBER 02, 2010

A.J.S. said...

Second Part:

I have worked in 4 different companies - ranging from startups to mid-sized to Fortune 20 - in last 6 years (by choice, and on my own terms). I must admit I have always been the "Rockstar" in the teams I have worked in. I don't know much in CS aspect of things, really, but I am the guy who gets things done - by hook or by crook, or sometimes by fluke.

Now, I don't know what a certain Google recruiter saw in my resume, but I now have a phone screen scheduled with G next week. And I know already, I am totally going to get screwed. I am more of a "Software Developer" than a "Software Engineer" or "Computer Scientist". I understand and use object-oriented paradigm, all common design patterns, solid networking concepts - but hardly any core data structures like Graphs or Trees. Maybe Maps\Hashtables at times. So reading all this, I am already shitting in my pants, and I feel as if I am going to get raped come next Monday. Maybe I should just inform the recruiter that I am not in a position to take the interview, I don't know if that's rude or whether wasting someone's 45mins-1 hr is rude"r".

I don't blame Google for their interview process, the fact is they can afford it right now so why the heck not. All my life I wanted to work for Google, but looks like I just got a reality check... Pray for me guys, at least I can get a graceful denial...much better than people at Google thinking "what a pile of $hit this dude was"

8:07 PM, NOVEMBER 02, 2010

Pherdnut said...

"Especially if you're Alan Turing, in fact, since it means you obviously don't know C++."

Funniest damn thing I've read in months.

10:10 AM, NOVEMBER 26, 2010

none said...

Awesome article. I was thinking about trying my luck at Google (about to get my Bachelors of Science in Computer Science). While I know C, C++ and Java, my main language is Python (pretty proficient in it too).

I was wondering if Google still places a heavy emphasis on Java, and whether or not I should do/learn more of it before I apply and get interviewed. Would I be able to make it through with just Python knowledge?

8:44 AM, DECEMBER 13, 2010

Abhishek Ghose said...

I realize this is a very old post, and it isn't unreasonable to believe I won't see a reply- but, here goes anyway: What is Google's usual approach to the hiring procedure for a specialist? I am a MS, with research experience in NLP and Machine Learning - should I be expecting a different sort of interview?

12:13 AM, DECEMBER 29, 2010

Saurabh S said...

Of course the blog is enormously helpful and entertaining! But for me it had these moments of deep inspiration! Gracias!

8:50 AM, JANUARY 22, 2011

sayasree said...

I studied the following books for Discrete Mathematics about 25 years ago:

Discrete Mathematics For Computer Scientists And Mathematicians
(Paperback)
by Joe L. Mott,Abraham Kandel,And Theodore P. Baker

The book is easy to read (does not scare the reader with lots of symbols and equations aka pure mathematics).
At the end of the each chapter, you will find good number of exercises marked from one star (*) to to five stars (*****) indicating the level of difficulty. I think is the one of the best books are Discrete Mathematics. Indians can buy the book online on FlipKart for mere INR 315 (about US$8).

URL: http://www.flipkart.com/discrete-mathematics-computer-scientists-mathematicians-book-8120315022

The books covers counting problems (Combinatorics), Reccurrence relations, and Graph

Theory.

Good luck with your Google Interview.
--Saya Sreenivasulu
Software Engineer, India/UK.

10:54 AM, FEBRUARY 13, 2011

sami said...

Hi Steve

"For God's sake, don't try sorting a linked list during the interview."

Why not? Don't get it...

12:57 AM, FEBRUARY 22, 2011

Kompila said...

These are the reasons Google is where it is right now and why it will continue to be above all....(even if some of the people can't understand it...)

I am preparing for my first phone screen and i am chichek shit...I'll post later to let you know how it goes. Peace!

10:19 AM, APRIL 06, 2011

Chankey Pathak said...

Very useful post.
Thanks a lot :-)

12:55 PM, APRIL 21, 2011

Evan said...

Hey, Steve,

I'm a little late to the party, but just wanted to say thanks because your advice helped me focus on what to study for round 2 with Google (I interviewed a year ago and got turned down). I'll be starting a front-end software engineering position in July.

10:20 AM, MAY 06, 2011

WebHrushi said...

Thanks Steve...this was really very helpful.I recently had a phone screen with Google and this really helped me for preparation.I am a Mechanical Engineer who is working as Software professional and almost illiterate about the Algorithms,data structures(But I know how to use Java Collections) and overall software engineering.
After reading all the recommended algo and ds books in 5 days,somehow I survived the 35 mins of phone screen but I am not sure if I can survice in next round (if any).
I am doing great as a Java Programmer without these things.But after I discovered the magic world of algorithms my perspective of looking at the problems really changed.Now I can imagine why Google interviews are around algorithms and data structures.But as I am not a native computer science/engineer...there is still lot to learn for me and it might

not be possible in a short span of 2-3 weeks.Do you think there is any place for me in Google? Could you please suggest me some books I can refer.Right now I have "Introduction to Algorithms",MIT and "The Algorithm Design Manual" (For graphs)

9:36 PM, MAY 31, 2011

Chicken McLovin said...

Very helpful Stevey! I actually did some coding for your game Wyvern, and have been tossing around the idea of getting out of the world of Defense and looking at moving on to something that actually challenges me as a software developer (rather than an Systems type Engineer). Look forward to applying and one day working for a company like Google (... like there's any other!)

5:15 AM, JULY 19, 2011

Mauricio said...

"don't say choo-choo-choo while you are thinking".
So THAT's what I did wrong...

8:03 AM, JULY 20, 2011

Anand said...

Hi Stevey. May I know your email ID want to forward my resume

8:26 AM, JULY 28, 2011

POST A COMMENT

<< Home