Google's Headquarters, photo from stubgroup.com

Average to Googler in Four Weeks: A Study Plan

Published on May 30, 2017

Milad Naseri   **4 articles**   ✓ Following
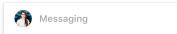Software Engineer at Google

TL;DR

I was successfully able to plan for and achieve a successful interview at Google. Here is my routine, the subject matter I studied, and some other tips.

Introduction

I have already written a number of posts on my first attempt at landing a job at Google. Though that attempt didn't get me the job, it helped me understand myself, my career path, and my shortcomings a great deal better. That is why I advise that everyone at least try their hand at one such extensive interview process -- if not Google's, then somebody else's.

I held off writing this "guide" until I was at the job for a little while, because I wanted to know more about the correlation between the interview process and the job itself. A lot of people want to know how much of their times will be spent using "algorithms" or "data

👤 Messaging

preparation process. I aim to answer that question at the end of this post.

Before getting into the meat of the post, I will first clarify some points and then
bit about who I consider to be an average software engineer. Then, I will talk ab
routine I picked, why I picked it, what are some important things to note about i
replicate it. After that, I will go into what I chose to study and why, what I actua
study, and what I reprioritize based on my times. Finally, I will talk about the int
process and how it relates to the job itself, and why it is designed the way it is (k
own experience, and not that of anyone else at Google). So, feel free to skip any
sections you feel are not interesting to you, though I would be happier knowing
have read all of this rather long post.

Disclaimer

This post or anything else on my blog is not affiliated with Google in any way, n
reflect Google's position, opinions, or stance on any matter. Everything is the re
own experience, and should be read as the philosophizing and musings of an eng
than the position of a company.

How Much Time to Really Put In

I think it bears mentioning that if you have MORE than four weeks, then by all means, you
should put more in. The most important part of this article is not the study plan for four
weeks. Rather, among other things, it is:

- The material you should cover,

- The importance of problem solving skills,

- How to be methodical and establish a routine

Clarifications

First of all, let me clarify that I didn't go through the phone interviews this time around. For
more on that, you can read my entry on the interview process I went through. So, you might
not actually need to limit yourself to four weeks.

Also, I was already pretty invested in taking the chance and going for the interview. If you
aren't, find a reason to be. If you can't, don't bother. That is because this process is tiresome
and extremely time-consuming, and if you aren't fully committed, you will soon find
yourself with excuses and reasons to not do this part or not read that chapter.

A very important point to mention is that my wife went all in with me. She was as invested
in making this move as I was. This meant that she helped me with distractions, she helped

interview questions by asking me the questions and reading me the hints.

Another clarification I would like to make is that this process is by no means a c
meant to convert every average Joe to Google employees; rather, it is a framewo
connecting with the subject matter deeply enough that you would feel comfortab
the problems in an interview setting.

It should also be noted that I prepared for my interview without taking any days
for the couple of days leading up to the interview, and the day of the interview it

Lastly, I would like to note that it has been years since I got out of university, an
years since I formally studied the subject matter I introduce here. Therefore, if y
recent brushes with this material and are already familiar/comfortable enough w
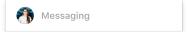you think you can immediately jump into problem-solving, don't bother with thi

Who is Average

Alright, so the title of the post suggests that an "average" engineer should be abl
for the interview in about four weeks.

The average person, in my mind's eye, has these attributes:

- She is comfortable writing code in at least one programming language -- preferably a higher level language such as Java or C, rather than a purely scripting language such as JavaScript, though that is absolutely fine.

- Understands how a computer works. I am talking about basic understanding of operating systems, hardware architecture, and the intrinsic architecture of the language itself. Know what it means to write a while loop within a for loop. Do your homework and know how much memory you are going to use in your "super-fast" implementation of Dijkstra's algorithm.

- She knows what data structures and algorithms are, at least at a very high level.

- She understands why we need to write code that is readable, maintainable, and performant.

- She loves her job already. You can't be a real software engineer if you aren't in it for the love of the job. You must love the creativity, and be excited about the repetitive parts as opportunities for improvement. You must get excited at the prospect of being given a problem to solve. If you aren't, you will end up a miserable 60 years-old that is waiting impatiently for retirement to kick in and be rid of it all.

It's All in the Routine

👤 Messaging

you missed it, you have got to be a hundred percent committed. You must really
good work, from moment zero to the moment you are out of the interview, and e
while after that (until you start working at the new place, at which point, you wi
always at your best, otherwise, why bother?).

Therefore, it is absolutely paramount that you create a routine and stick to it. My
consisted of getting home at about 5-6pm, getting a super quick shower, studyin
and a half to three hours, giving myself a one hour break, and studying for anoth
half to three hours. Then, I would rest for half an hour to clear my mind and get
sleep. Then I would go to sleep for however much I could, aiming to be as close
as possible (really, mostly 5 hours :D), wake up at 7am, go to work, and repeat t
I'd supplement the lack of sleep with short naps during the day and in my rest tir

The aim of the routine is manifold:

- Reducing decision making overhead time

- Increasing productivity by allowing for a habit to grow on you

- Creating a measurable path towards your goal

- Creating a repeatable experience

Therefore, be prepared to be harsh to yourself. Scold yourself if you stray from the routine.

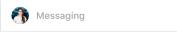Let's break this down a little bit more.

Study Time

In your study periods, study for exactly 45 minutes. Keep a stopwatch going to measure how
long you are studying each day. The goal is to start slow, at 2 hours of useful study time and
ramp it up as much as possible.

Do not extend your study time. Do not go over the 45 minutes marker at any cost. Even if at
minute 42 you realize that by sitting down for 10 more minutes you might finish the current
section or topic, you must not sacrifice your rest. This seemingly benign deviation will be
detrimental to your focus and precision in the long run.

It is absolutely okay to not achieve your goal 100%. In fact, if you are consistently hitting
your goal, you are probably not planning ambitiously enough.

Make sure that your study corner is solely dedicated to your study. If you live in a study, or
share a room with a roommate, this can be an unused corner of your room, facing the wall. If

👤 Messaging

is up, get out of that spot and don't go there again until you intend to study.

Make sure that when you are studying, you are 100% committed to your study t
bothered by distractions or fatigue. This is much more easily said than done. An
me, as a person suffering from mild OCD and a very short attention span, I can
understand you if at this point you have your hands in the air and are about to cl
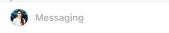browser window. So, let's get into that.

Dealing with Distractions

Distractions are usually stray thoughts that derail you from your focus of attentic
you are already aware that they are fleeting, momentary thoughts, questions, and
inspirations. As such, you feel an urge to follow them immediately, lest you forg
on, thus letting go of your current task. Therefore, the first step is to reassure you
you are not going to forget them. This is simply done by the expediency of using
"distraction list". A distraction list can be anything: a piece of paper, a Notes do
file, a blog entry. Whatever makes you feel like you are less likely to lose it. I cr
list on Notes on my MacBook, synced it with iCloud, and then I felt safe that eve
my MacBook be stolen or lost, my distraction list will not follow suit (I sure hav
priorities straight, right?).

Having created a distraction list, whenever something comes to your mind that demands
your immediate attention, instead of following it up, you jot it down. Remember that the
purpose of this exercise is to be more productive, so, don't spend more than a few seconds
writing the idea down. Usually, you can summarize these fleeting ideas in a few words:
"white buffalo woman", "monad theory", "iPhone gyroscope accuracy", "find out how you
should hold a fountain pen without staining your fingers", etc.

Next, comes the trust in this distraction list, since only a part of the problem stems from your
fear of losing the idea. The next part is fearing that you would never follow it up. As such,
your next task is to appraise your distraction list during your hour-long break (or other long
stretches of break time you are giving yourself, if your routine is different than mine). Pick
at least one item and do it. Pick something that is not completely out there (e.g. "book a
Hawaii vacation") or physically exhausting (e.g. "try a triathlon"). Don't cross out anything.
Keep these distraction items around and even if you don't ever get to do them, they will be a
source of inspiration for later on.

Your smartphone can be your enemy, and it can also be your ally. Delete your social
accounts from your phone, or at least disable their notifications. Delete addictive games that
tempt you. Don't worry, you are going to restore all of that in a month and get back to
smashing monsters and annihilating neighboring tribes and mega-liking your friends'
awesome pictures. Instead, use your phone as a tool to boost your productivity. I did so, and
have included links to some apps I used at the very end. I don't think posting pictures of your
study routine is going to help you, since you will most likely be tempted to check your

Messaging

Dealing with Fatigue

It is completely natural to feel exhausted after extended periods of study, especia
the first few days. So, make the best of your break time. Choose a mentally relax
beforehand. Remember that your breaks are precious and you don't want to spen
minutes each time figuring out what you want to do. I chose to watch "The Adve
Sherlock Holmes" in 15-minute increments. I governed my break time (as well a
time) with countdowns that would alert me when time was up.

Find a posture that helps your body relax during this time. I laid down with a cu
front of the TV and made sure that my back was resting, and I wasn't straining n
or my neck.

If possible, pick an activity that lets you close your eyes a little bit, though make
you don't doze off.

Once your 15 minutes is over, **sharpen your mind**. You must have a test of som
to make sure your mind is functioning at a good pace before going back to study
used the Quick Math app on my iPhone, and made sure that I would not start studying until I
had hit three stars in the "solve" mode on beginner difficulty. I didn't want to exhaust myself,
I just wanted to focus my mind with something repetitive, while attention-demanding. If
after about a few minutes I was still not doing much better, I would rest for another fifteen
minutes, and repeat the process.

There are tons of ways to do this instead. You can try doing the alphabet in reverse order five
times without any mistakes. You can try to rewrite a simple sentence with synonymous
words. I chose math, because doing math exercises engages the same parts of the brain as
studying algorithms and other abstractions [citation needed] (don't cite me, I'm relying on
my memory here; but it doesn't sound illogical, so there.).

Study Comfort Level

Measure your comfort level in your study environment. Make adjustments. Try to be
ergonomic. If you can't have an ergonomic environment at home, go to a coffee shop, a
library, or even a park. See what's comfortable for you, but don't spend too much time. Settle
on a "comfortable enough" study environment that lets you get to rest and back quickly
enough. Pay special attention to lighting and distraction levels.

Exercising and Your Physique

Make sure that your body gets enough physical activity. Otherwise, you are going to cramp
up and lose your concentration, or be unable to perform at your best. You aren't doing
anyone any favors by cramping yourself up in a corner and tightening your neck muscles all
day long.

👤 Messaging

this without hurting yourself. I chose this one. Also make sure that during your
at the beginning of your day of study, and before going to bed, you are doing str
forget to shower, use the bathroom regularly, and inhale enough air to keep your
healthy.

The Paper Trail

If your studying doesn't leave any physical evidence, it is as if you didn't study.
Do not highlight the actual text, do not write in the margins. Take up a dedicated
and write in that notebook. Be organized. You are going to thank yourself later o
are neck-deep in nerves and want to remember that thing you read two and a hal

Don't overdo it though. Your notes should remind you of what the textbook said
solution was all about. You don't want a replica of the textbook in your pocket. I
opinion, every hour of study should produce between 3 to 6 pages in a pocket-si
and no more than that. Pick a pen and stick to it. Don't use multiple colors. Thes
notes. These are intended for the *very* short term. Also, you can always come ba
color it up, which supposedly increases your memory retention.

Use your own words, pictures, diagrams, and keywords. After all, your notes should reflect
your understanding of the text, not just the text itself.

Eating, Sleeping, and the Rest

Make sure you eat meals at regular intervals. Again, this is all about the commitment and
sticking to the routine. Make sure your meals are nutritious, and also filling, but not
overwhelmingly so. You don't want to feel your bowels jiggling around whenever you lean
this way and that (and yes, I did have to paint that picture).

If you can, keep your meals light and instead snack frequently during your breaks.

Don't miss a bedtime. Even if your planned 45 minutes study period isn't done yet, don't
miss your bedtime. Think of your bedtime as a the super alarm that overrides and supersedes
all other times. The police are going to come and arrest you and the FBI will lock you up
forever in jail without letting anyone know and they will let a Koala bear take over all your
social accounts if you do (though in my case, that would probably be an improvement).

I realize that this might result in less pure study time, since up until this point we have added
so many triggers for shortening the study periods (rest alarm, focus trigger, etc.), but believe
me, being able to study for half the time with full attention will pay higher dividends in the
long run.

Also, don't kill off your relationship with your family members, your significant other, and
your roommates by being obsessively annoying, intolerable, and intolerant. Remember that
you might get the job, but end up losing your closest allies in life.

👤 Messaging

Now that all of that has been said and done, let's get to what I actually planned to
divided my studying into three portions:

- Review everything on my resume.

- Focus on filling the gaps in my CS knowledge as well as remember things I
  in a while

- Focus on being able to come up with answers quickly.

The first item is obvious. You absolutely must have a great command of everyth
included on your resume. Anything you put on your resume is fair game, so you
prepare for it. That being said, I do encourage you to read the section "Why the
Books?" to have a better understanding of what *type* of questions are being aske
that would correlate with your current resume.

The second item seems very similar to the third; but it really isn't. In the second
approach was very academic. I focused on reading the textbooks and material th
being taught in universities, as well as other books suggested by people in the in
contrast, in the last portion of my studies, I mainly focused on having a very pragmatic
approach to the subject matter. I didn't focus on learning the material very deeply. That was
for the previous portion. Rather, I tried to make myself adept at figuring out which technical
subject a problem was related to, what were some of the gotchas and loopholes that would
be problematic, and expose myself to as many problems as possible.

Before talking about the plan itself, I am going to talk a little bit about the topics I covered,
why I chose to do it, and how I went about it. If you don't care, skip to the plan.

Academic Material

I planned to read the academic material for about three weeks, and reserve a week or so for
solving interview-related problems. To that end, I first selected subjects I would like to
invest in:

- Algorithms

- Data Structures

- Problem Solving

- Systems Design

- Operating Systems

- Networks

- [Introduction to Algorithms, 3rd Edition (MIT Press) (a.k.a CLRS)](#)

- [The Algorithm Design Manual, 2nd Edition (by Steven Skiena)](#)

- [Programming Pearls, 2nd Edition (by Jon Bently)](#)

- [How to Solve It (by George Polya and John H. Conway)](#)

- [Algorithm Design, 1st Edition (by Jon Kleinberg and Eva Tardos)](#)

- [Introduction to Information Retrieval, 1st Edition (by Manning et al)](#)

- [Operating System Concepts, 9th Edition (by Silberschatz et al)](#)

- [Computer Networking: A Top-Down Approach, 7th Edition (by James Kurc
  Ross)](#)

As I have mentioned before, I didn't consider a section finished until I had physi
and an organized paper trail leading to that section. So, naturally I realized that I
able to finish all of these in three weeks. Therefore, I decided to only focus on th
subjects:

- **Algorithms & Data Structures**: I decided that since algorithms and data structures
  were an important part of this, I should read CLRS, and supplement it with Skiena's
  book, but not read Algorithm Design, as that is a more in-depth discussion of the
  philosophies and approaches to algorithms rather than a primer on the theoretics.

- **Problem Solving**: Programming Pearls looked like an excellent source of applicable
  knowledge, applying the basics of computer science to real-world problem from a
  hardened, experienced professional. I also loved the little anecdotes in the book. So, I
  decided to read this book, instead of George Polya's great book on problem solving as a
  pure skill.

- **Systems Design**: I remembered from my brush with the book that "Introduction to
  Information Retrieval" touched on many different subjects and opened up a great
  discussion about highly scalable information processing systems and their design. That
  is one reason I chose to prefer this book. Another is that the subject matter closely
  resonates with what Google is at its core. It even dedicates a good portion of the book to
  the discussion of a good search engine and tries to unravel the secret algorithms used at
  Google.

Having fleshed out a plan for these three subjects, I realized that I wouldn't have time to read
the rest of the subjects in any meaningful way. Since these topics were more important than
the others, I chose to not compromise these in any way, and instead forewent the study of the
other subjects altogether.

👤 Messaging

I will get to how I studied each topic.

Algorithms and Data Structures

For this topic, I had to choose a foundational book, and between Robert Sedgew
book and CLRS, I decided to read CLRS first. I don't like Robert Sedgewick's bo
algorithms for two reasons:

1. I don't like the code

2. I don't like the order in which the subject is explained

NOTE: That is by no means meant to disrespect the author or his work. I'm pure
from the point of view of someone short on time and out of touch with the basic

Skiena's book is more of a supplementary book, since it glosses over a lot of the
algorithms and data structures, but offers a lot of great insight when it comes to
world applications of these algorithms. I particularly love its graph section, dyna
programming chapter, and its study of heuristic approaches.

So, I decided that I would first read a section thoroughly, then, I would go back to it, reread
it, and take notes, and finally, I would sit down and implement the algorithms and data
structures I learned in Java. It should be noted that I had first started to implement a bunch of
these algorithms back in 2015. So, this time around, instead of implementing them, I would
read through my code, annotate them with comments, and read the tests I had written for
them to make sure I remembered their implementations.

Why Java? Because that is a language I am very comfortable in. My focus was doing some
real world experimentations with the code, not to make it super beautiful or very performant.
It was meant to represent the concepts introduced in the book very clearly.

I created a GitHub repository and started to upload my code there. You can find this
repository (now made public) here: https://github.com/mmnaseri/cs-review

It has a separate module for CLRS (cs-clrs) and another one for Skiena's book (cs-skiena). I
haven't taken notes from or implemented anything in Skiena's book that I had already
covered in CLRS, so, reading Skiena's book was much quicker for me.

Problem Solving

Reading Programming Pearls is an interesting experience, especially after reading something
as rigorous and mathematically correct and stringent as CLRS and Skiena. It is like reading
a story book interspersed with problems and challenges. So, I decided to treat it that way. I
would read a complete column, go back, take notes, and then go to the problems section. I
would select a few of the problems that looked more interesting to me and would dedicate a
study session to solving those and writing down my findings.

perform back-of-the-envelope calculations, being able to deal with extreme con...
knowing when to do optimizations that look ridiculous is a very important skill.

Obviously, there are sections in this book that overlap with what is in CLRS and
books, so, I happily skipped over them.

Systems Design

For systems design, I focused on Introduction to Information Retrieval, which g...
insight into the design and implementation of a search engine, both from an aca...
perspective and looking at the theories, as well as an engineering outlook.

There is an online reading, free version of the book available from the authors' ...
which is actually more up-to-date than the one available on Amazon.

Since this book covers a very wide variety of subjects, I would find myself looki...
additional information for more context on what I was reading. My reading of th...
a little different: I would first read a section, annotate it with the information I h...
up, then I would go back, reread, take notes, and move on to the next section.

The book does have chapter and section problems, but they are not very helpful in preparing
you for the interview process and are also very time-consuming on average.

Preparatory Material

After I finished reading the academic material listed above, I proceeded to focus on applying
the material I had learned to problem solving. For this portion of my preparation, I used the
following material:

- DZone RefCardz (which are small cheatsheets and one-pagers that talk about various
  technologies, theories, and practices); I consulted this mainly to quickly remember
  design patterns, enterprise application integration patterns, and the like.

- Programming Interviews Exposed, 3rd Edition (by Morgan et al); This is what I read
  before moving to "Cracking the Coding Interview", mainly because it didn't have as
  many problems, and also because it was a tutorial+exercise kind of book, which was in
  sync with what I was reading before. I usually like to manage my transitions so that they
  aren't too sudden.

- Cracking the Coding Interview, 6th Edition (by Gayle Laakmann McDowell); The **sixth
  edition** here is the keyword. This edition not only has more problems, but more
  importantly, has interview-style hints that are a great asset for preparing yourself for
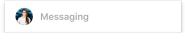 interviews in which you are completely stumped and need some help from the
  interviewer.

👤 Messaging

make any concrete plans around them.

Right after my academic studies, I jumped into reading *Programming Intervie*
Exposed. I first would read a chapter, solve the exercises before the book gave t
and then would jump to chapter problems and rigorously solve all of them on pa
use any computer coding for this. I always coded on paper, using a pencil, and c
myself to using as little paper as possible. This is a great exercise for helping yo
your real estate, because in an actual interview you are most likely going to be c
whiteboard with limited space, and you cannot erase the things you have previou
on the whiteboard.

One particular part of this book I found very useful was the section on bit manip
gives you a very good idea of how to do bit manipulation without getting lost in
of the binary arithmetic involved.

Once I was done with this book, I moved to *Cracking the Coding Interview (*
spent about half a day on everything in the book preceding the interview questio
are legitimate parts of the book that cover things like how to get ahold of a recru
properly comport yourself in the interview, behavioral questions, talking number
on. Then, I proceeded to solve interview questions.

I would first skim through the subject matter in the chapter. I say skim, because most of the
material is what you should already be more than familiar with at this point. Then, I would
give myself 20 minutes to solve each question. For each question, I would do the following:

1. Repeat the question out loud in my own words.

2. Say out loud what the general approach would look like (e.g. "for this question the brute
   force algorithm would be to scan the linked list and then cache all the values").

3. Start by writing code for the **brute force** version.

4. Write the solution on a small whiteboard or a piece of paper.

5. Demonstrate why it works using a small and general sample. Choosing test inputs is a
   great part of this process. Don't choose samples that would take 10 minutes to go
   through. But don't bake assumptions into your tests. Don't worry. You will get better at
   this as you solve more questions.

6. Avoid writing parameter checks for your methods. Just say out loud things like "we
   should check for nulls here". In a real interview, some interviewers might opt to ask you
   to write those checks, while most wouldn't. The important thing is to be aware that these
   checks should be performed.

7. Think of edge cases and fix any errors.

9. Consult the book for hints, in case you have missed the answer.

I would repeat this process continuously and improve my approach. One thing y
really good at is figuring out which problems are hard. Also, don't worry if at fir
is closer to 40 minutes than 20. Just make sure that you unstuck yourself early o
consulting the hints in the book. Keep looking at the clock and improving your t
will start solving problems in way less than 20 minutes the more you go through
even hit 5 minutes for some of the problems.

Here are some general hints and tips for doing the exercises:

1. Don't overoptimize. As Donald Knuth would have it: "Premature optimizatie
   of all evil". Don't try to come up with the efficient, beautiful solution. Make
   solution is legible, modularized, and most importantly, fully functional.

2. Try to come up with ways the answer could be improved.

3. Calculate the memory and time complexity of your answers. Don't be overly
   just say $O(n^2)$ or $O(nlgn)$. You don't need to prove it, just make reasonable
   see you are right. Remember that by this point, you are past the academic phase of your
   studies.

4. If your tests work and you are sure of your answer, you don't necessarily need to check
   the answer against the book, but seeing how a third person would do it is not a bad
   thing. This all depends on how you want to manage your time.

5. Once all of this is done, think of how you can improve the solution (e.g. caching,
   remodeling, alternate solutions, etc.)

As I have mentioned before, I concentrated the problem solving at the very end of my
schedule. This was intentional. First of all, the sense of urgency helped me manage my time
better. I am one of those people who functions well under a deadline. If you aren't, maybe
you should give yourself more breathing room. Secondly, this concentration means that you
will soon find a very good rhythm in solving problems. You will move past the initial
awkwardness of getting the meaning of the question to the business end of writing code very
quickly. Thirdly, this will put you in the right state of mind right before your interviews.

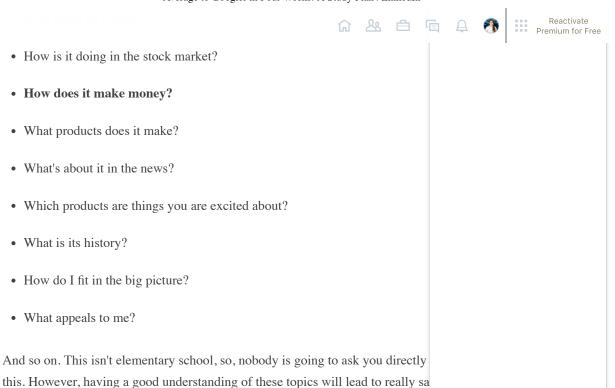## Learn About the Company

Needless to say, another preparatory piece of reading for you is learning about the company.

Some of the questions you should ask yourself are:

- What is its mission statement?

Messaging

- How is it doing in the stock market?

- **How does it make money?**

- What products does it make?

- What's about it in the news?

- Which products are things you are excited about?

- What is its history?

- How do I fit in the big picture?

- What appeals to me?

And so on. This isn't elementary school, so, nobody is going to ask you directly this. However, having a good understanding of these topics will lead to really sa conversations with the people you meet.

What About Mock Interviews?

Mock interviews are great, but not very helpful for short-term plans. Imagine going for a mock interview a week before your actual interview. If you did well, it would reaffirm your plan, but also waste a whole day. If you did poorly, you wouldn't really have the time to course-correct. But that's just my opinion, and I am sure that many people would disagree. Even I ended up doing mock interviews of a sort, with my wife stating questions, letting me rephrase, observing me while I solved, giving out hints from the book when I was stuck, and finally helping me compare the solution to that of the book.

The Plan

Finally, here is my study plan for the whole four weeks. Compare this to the list at the end of this section which explores how well I stuck to this plan and how much of it I missed.

The code words here are:

- CLRS: Introduction to Algorithms

- Skiena: The Algorithm Design Manual

- Pearls: Programming Pearls

- MIR: Introduction to Information Retrieval

- PIE: Programming Interviews Exposed

Messaging

I will present the plan week by week.

Week 1

- Day 1: CLRS chapter 1 to 5; 6 hours of study

- Day 2: CLRS chapter 6 to 9; 6 hours of study

- Day 3: CLRS chapter 10 to 12; 6 hours of study

- Day 4: CLRS chapter 13 to 14; 6 hours of study

- Day 5: CLRS chapter 15 to 17; 6 hours of study

- Day 6: CLRS chapter 18 to 21; 10 hours of study

- Day 7: CLRS chapter 22 to 26; 12 hours of study

Week 2

- Day 8: CLRS chapter 27 to 29; 6 hours of study

- Day 9: CLRS chapter 30 to 32; 6 hours of study

- Day 10: CLRS chapter 33 to 35, CLRS review; 6 hours of study

- Day 11: Skiena chapter 1 to 4; 6 hours of study

- Day 12: Skiena chapter 5 to 7; 6 hours of study

- Day 13: Skiena chapters 8 and 9, the catalog; 12 hours of study

- Day 14: Pearls, column 1 to 10; 12 hours of study

Week 3

👍 Like   💬 Comment   ↗ Share   🔵💡🟢 1,044 · 98 Comments

- Day 16: MIR, chapter 1 to 6; 6 hours of study

- Day 17: MIR, chapter 7 to 12; 6 hours of study

- Day 18: MIR, chapter 13 to 17; 6 hours of study

- Day 19: MIR, chapter 18 to 21; 6 hours of study

- Day 20: PIE, chapter 1 to 6; 12 hours of study

- Day 21: PIE, chapter 7 to 12; 12 hours of study

💬 Messaging

- Day 22: PIE, chapter 13 to 15; 6 hours of study

- Day 23: PIE, chapter 16 to 17; 6 hours of study

- Day 24: CtCI, beginning to interview questions; 6 hours of study

- Day 25: CtCI, Questions, 1 to 6; 6 hours of study

- Day 26: CtCI, Questions, 7 to 10; 6 hours of study

- Day 27: CtCI, Questions, 11 to 17; 12 hour of study

- Day 28: CtCI, Questions, 18 and Advanced Topics; Review resume and note
  of study

Here's what I actually ended up doing. At first, I didn't stick to the plan, and qui
that I am getting derailed, so I hastened to catch up in the following days. An im
thing to note is that because I was writing down everything and measuring my ti
able to quickly realize how much difference there was between my planned time
my actual time usage.

Secondly, I needed more time to study the MIR book than I had originally imagined. It
turned out to be a very heavy read for me, so, I had to put in more hours. This led to me
putting in some time to review the notes I had taken, up to that point. Finally, I ended up
solving the problems from Cracking the Coding Interviews at a much faster pace, because 1)
at that point I had a really great momentum, and 2) I really needed to.

Week 1

- Day 1: Review resume, read RefCardz, look online for questions and answers and tips; 4
  hours of study

- Day 2: CLRS, chapter 1 to 7; 7 hours of study

- Day 3: CLRS, chapter 8 to 10; 6 hours of study

- Day 4: CLRS, chapter 11 to 14; 7 hours of study

- Day 5: CLRS, chapter 15 to 17; 6 hours of study

- Day 6: CLRS, chapter 18 to 21; 10 hours of study

- Day 7: CLRS, chapter 22 to 26; 12 hours of study

Week 2

- Day 8: CLRS, chapter 27 to 30; 6 hours of study

- Day 10: CLRS, chapter 33 to 34; 6 hours of study

- Day 11: CLRS, chapter 35 and CLRS review; 4 hours of study

- Day 12: Skiena, chapter 1 to 6; 7 hours of study

- Day 13: Skiena, chapter 7 to 9, the catalog; 12 hours of study

- Day 14: Pearls, column 1 to 11; 12 hours of study

Week 3

- Day 15: Pearls, column 12 to 15; MIR, chapter 1 to 3; 6 hours of study

- Day 16: MIR, chapter 4 to 7; 6 hours of study

- Day 17: MIR, chapter 8 to 10; 6 hours of study

- Day 18: MIR, chapter 11 to 16; 6 hours of study

- Day 19: MIR, chapter 17 to 18; 6 hours of study

- Day 20: MIR, chapter 19 to 21, MIR review; 12 hours of study

- Day 21: Review notes, PIE chapter 1 to 3; 10 hours of study

Week 4

- Day 22: PIE chapter 4 to 6; 6 hours of study

- Day 23: PIE chapter 7 to 15; 6 hours of study

- Day 24: PIE chapter 16 to 17; 4 hours of study

- Day 25: CtCI first part; 5 hours of study

- Day 26: CtCI questions 1 to 9; 7 hours of study

- Day 27: CtCI questions 10 to 17; 11 hours of study

- Day 28: CtCI questions 18, advanced topics, review; 12 hours of study

The Day of the Interview

The day of the interview is extremely important (duh!). Try to wake up a little earlier the day before, so that you have ample time to rest the following night after finishing your exercises. When you wake up, do stretches, meditate, or do whatever it takes to relax your mind. Bring your mind to focus using one of the methods I mentioned in the section "*It's All in the*

Messaging

the CtCI or PIE books.

Try to remember the solution or rethink it. It doesn't matter. What matters is that
your mind into a problem-solving mode. It shouldn't be too difficult if you have
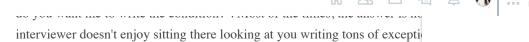majority of your free time for the past week and a half focusing on solving inter
problems.

Eat a light and nutritious breakfast. Fruit, vegetables, or anything else that work

If possible, arrive there about 30 minutes earlier. This gives you time to get past
nervousness of getting to a new place. Then, look around, get comfortable, and
minute review of your notes. This is where the notes being in pocket size format
handy. I was able to cover all of CLRS in the 20 minutes I was sitting there.

Finally, these tidbits might help you:

- As my wife kindly pointed out to me, the interviewers aren't superior human
  They are (if you do this right) going to be your colleagues one day soon, and
  want to have a friendly chat about the problems you are given.

- Be humble. Don't ever refuse answering a technical question. If the problem seems too
  easy, you probably have missed a detail. Have you forgotten to ask clarifying questions?
  If the problem is too difficult, don't worry, reiterate the question and make sure you
  understand the intent. Then ask for hints if none is offered.

- The interviewers know that you are nervous and they don't expect you to have the
  answer completely ready. Rather, they want to hear you think your way through it. Don't
  ever let silence stretch. Think out loud. At best, you are going to hit upon a great idea
  and hear encouragement, at worst, you are going to learn that your line of reasoning is
  flawed. Both are positive outcomes.

- Don't immediately jump into writing code. Usually, when an interviewer says "Write a
  function that ..." the last thing they want is for you to actually start coding. Restate the
  problem first. It buys you time and lets you know if you have misunderstood. Then,
  explain your proposed approach, and once the interviewer is satisfied, go for coding.
  Most of the time, they will just say "let's see some code" or something similar.

- Write clean code. Some interviewers don't care about slight syntactical mistakes while
  others do. Just check with them.

- Once you are given the question, write down the keywords and important bits of it on
  the whiteboard. This helps you focus on those important parts, tells your interviewer
  what *you* think to be important (which they might potentially refute and correct), and
  also helps you remember your line of reasoning while you are solving the problems.

do you want me to write the condition? Most of the times, the answer is no. The interviewer doesn't enjoy sitting there looking at you writing tons of exceptions conditionals.

- Use functions. They modularize your code, let you fill them in later, and help outline your answer.

- Don't EVER try to write the best answer first. Work with the best *working* s comes to your mind. Find its flaws, and figure out how you can improve on

- The interviewers aren't there to be your friends. Remember that if they don't as particularly friendly or happy-faced, it has nothing to do with you or your performance.

- If you have seen the question you are being asked, let the interviewer know. This is specially true if the question has been asked of you on the same day.

- Remember that you start each interview with a completely clean slate. If you one interview, relax, clear up your mind, and do your best at the next. I mess first, and got my act together for the rest of them. It worked for me. It has w tons of people. It will work for you.

- Most interviewers will let you ask questions from them at the very end, for about 5 to 10 minutes. Don't come up with questions on the spot. Have something ready. Some good questions to ask Googlers are what they are currently doing, what 20% projects they are involved in, if any, and how do they like it there. Also, you can ask about company culture, things that make it unique for them, fun facts about the company, and most importantly, their own professional daily lives.

Why the Text Books?

Now, I need to address the all-important question of why do recruiters, interviewers, and advocates constantly badger us about sticking to the good old textbooks, instead of having us do what engineers do best: use technology to solve our problems. At most companies, having something like "Spring" or "Apache Storm" on your resume is a great boost, and sometimes even mandatory. You are sometimes even asked very specific and pedantic questions about these technologies. Companies like Google, though, simply don't care.

What they are there to evaluate is how well can you solve problems, in a very contained environment, using the most basic (albeit extremely powerful) tools at your disposal, namely, algorithms, data structures, and systems design.

Therefore, focusing on these textbooks will hone your skills in applying these tools of trade to the problems in their most abstract form: you are not going to bother about thinking whether or not a particular technology is suitable to solving a given problem; rather, you will

Messaging

algorithmic form, applying the algorithms you have learned, and finally testing

You will be evaluated on your problem-solving skills, which to be honest, is the
your experience that you will take with you to Google (or any other big compan
The CI tools you are used to, the technologies, the conventions, and even the tec
will for the most part be irrelevant to how things are done at Google.

They either have similar tools and technologies to solve their problems, or they
the issue in some other way. For instance, it is no secret that Google internally u
instead of Spring for dependency injection.

So, the aim of the interview process is to make sure the best problem solvers are
through, and that these people are well-suited for thinking about their solution, s
with others, accepting criticism and guidance from more senior people, and final
implementing it.

You will be expected to know the things on your resume and be able to speak fo
unless the role you are applying to doesn't specifically mention a given technolo
are you are not going to be asked detailed and specific questions.

The process might seem like it is aimed to filter out people based on their educational
background or age, but in truth it isn't. In my Noogler team there were people of all ages,
backgrounds, and experience levels. If you are ready to commit yourself to preparing for an
intense study period, and then a fun and yet very challenging process, this job is right for
you.

Conclusion

I have tried to be very thorough in my explanation of what went in my head from the
moment I decided to give the interview to the moment I actually went in. The material I
have written about in here is by no means a conclusive list. It is just what I ended up doing.

If you want to hear on the same matter from a more experienced person, Steve Yegge has a
great blog post titled "Get that job at Google" which is a great post and also a great source of
inspiration for this article.

This post was written with the intent of getting you ready for a generic SWE job at Google.
Don't forget to ask your recruiter for more specific material pertaining to the actual role you
are applying for.

Also, I would be very happy to know if you have benefited from this post, think something
is missing, want to know more about something, or generally have a comment. Let me know
and get in touch!

Send Me Your Resume

and whether or not this is the right move.

To quote a colleague of mine: every software engineer should experience workin̶
at least once.

Resources and Assets

iPhone apps I used to convert my phone from a liability to an asset:

- Quick Math (for sharpening my mind)

- Timer+ (for keeping track of my study, break, and aggregate time)

- Notes (for keeping a running distraction list)

- Books: as mentioned in the reading material section (see *What to Study*)

Videos:

- How to: Work at Google — Example Coding/Engineering Interview

- Cracking the Coding Interview (Video Preview)

- 4-Minute Neck and Shoulders Stretch at Your Desk | Yoga | Gaiam

- Interviewing at Google - YouTube

Other online resources

- Get that job at Google

- How to get a job at Google, interview questions, hiring process

- Google's hiring process

- http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index (though to be honest, I personally didn't find much parity or correlation between TopCoder questions and actual interview questions).

- My GitHub repository for this review process: https://github.com/mmnaseri/cs-review

Update

You might also be interested in the other article I just published, which contains tips on taking notes while studying, as well as links to download my own study notes.

Messaging

wider audience. For the original, and other posts in the same vein, see my blog.

If you find this article useful, please share it with your network.

Published by

Milad Naseri   **4 articles**
Software Engineer at Google
Published • 3y

Reactions

98 Comments

Most Relevant ▾

Add a comment…

Hemant kurmi • 2nd
SDE II | Full Stack Java Developer | Machine Learning

Hi Milad, Thanks for posting this , Did you coded each and every problem from CLRS. Ans then did you used Leetcode too ?

👍 · 1 Like   💬 · 3 Replies

Load previous replies

> **Hemant kurmi** • 2nd   1y ⋯
> SDE II | Full Stack Java Developer | Machine Learning
>
> **Milad Naseri** hi thanks for reply , yes i have seen your repo today , i read this article 2 weeks before and started following the schedule but i was just reading the books not coding all the questions, i coded only few , then today i seen the link to your repo and got confused. I was planning to first finish reading then leetcode . Is this good idea?
>
> 👍 · 1 Like   💬

> **Milad Naseri** • Following   1y ⋯
> Software Engineer at Google
>
> **Hemant kurmi** yes. I'd say laying the foundation before going on to practice coding is a good idea.
>
> 👍 | 💬

Brian Madunezim • 3rd+   11mo ⋯
Software Quality Assurance Engineer

Good stuff Milad, must say! I had a quick question. I was looking to see if there's a book that simply takes out the coding and explains to you how to work out certain problems using logic. For example, in order to get the reverse of a number you simply divide by ten, then add it to the remainder, so forth and so on. Usually these questions are done in code, but we all know that stripping the code and just understanding how to work with the algorithm is key. I see that you have recommended Algori …see more

👍 · 1 Like   💬 · 1 Reply

> **Milad Naseri** • Following   11mo ⋯
> Software Engineer at Google
>
> Hey! I'm glad you found this useful. No. CLRS does not contain anything like that. That's mostly problem solving + math. For problem solving, I'd say you can read the George Polya book. For math, there are tons of resources. Another book that does sort of talk about the thought process that goes into such algorithms is the Programming Pearls book.
>
> 👍 | 💬

👤 Messaging

Milad Naseri

Software Engineer at Google

✓ Following

More from Milad Naseri

Note-taking for Interviews

Milad Naseri on LinkedIn

My Google Interview

Milad Naseri on LinkedIn

Using Spring in the Modern World

Milad Naseri on LinkedIn

Messaging