

## Bart Wronski

Technology, programming, art, machine learning, image and signal processing

---

## Insider guide to tech interviews

Posted on [January 4, 2022](#) by [bartwronski](#)

I’ve been meaning to write this post for over a year... an unofficial insider guide to tech interviews!

This is the essence of the advice I give my friends (minus personal circumstances and preferences) all the time, and I figured more people could find it useful.

The advice is about both the interview itself – preparing for it and maximizing opportunity to present your skills and strengths adequately, but also the other aspects of the job finding and interviewing process – talking with the recruiter, negotiating salary, making sure you find a great team.

A few disclaimers here:

**This post is my opinion only.** It’s based on my experiences both as an interviewer (I gave ~110 interviews at Google, and probably over a 100 before that – yep, that’s a lot, had weeks of 2-3 interviews!), someone who always reads other interviewer’s feedback, and an interviewee (had offers from Google, Facebook, Apple – in addition to my gamedev past with many more). **But it doesn’t represent any official policies or guidelines of any of my past or current employers.** Any take on those practices is my opinion only. And I could be wrong about some things – being experienced doesn’t guarantee being great at interviewing.

**All companies have different processes.** From “big tech” Google, Facebook, and Amazon do “coding” (problem solving / algorithm) interviews based on standard cross company procedures, Microsoft, Apple, and nVidia have more custom / each team does their own interviewing. A lot of my advice about problem solving interviews applies more to the former. But most of the other general advice is applicable at most tech companies in general.

**Things change over time and evolve.** Some of this info might be outdated – I will highlight parts that have already changed. Similarly, **every interviewer is different**, so you might encounter someone giving you exactly the opposite type of interview – and this randomness is always expected.

**This is not a guide on how to prepare for the coding interviews.** There are some books out there (“[Cracking the Coding Interview](#)” is bit outdated and has too many “puzzles”, but it was enough for me to learn) and competitive programming websites (I used [Leetcode](#), but it was 5y ago) that focus on it and I don’t plan to “compete” in a blog post with hundreds of pages and thousands of hours others put into comprehensive guidelines. This is just a set of practical advice and common pitfalls that I see candidates fall into – plus focus on things like chats with the recruiter and negotiating the salary.

**The guide is targeted mostly at candidates who already have a few years of industry experience.** I am sorry about this – and I know it’s stressful and challenging getting a job when starting out in a field. But this guide is based on my experience; I’ve been interviewing mostly senior candidates (not by choice, but because of shortage of interviewers) and I was starting over a decade ago and in Poland, so my personal experience would not matter either. Still, hopefully even people who look to land their first job or even an internship can learn something.

## Most important advice

The most important thing that I’d want to convey and emphasize – if you are not in a financial position that desperately needs money, then **expect respect from the recruiter and interviewers** and **don’t be**

## desperate when looking for a job and interviewing.

Even if you think some position and company is a dream job, be willing and able to walk away at any stage of the process – and similarly, if you don’t succeed, don’t stress over it.

If you are desperate, you are more likely to fall victim to manipulations (through denial and biases) and make bad decisions. You would be more likely to ignore some strong red flags, get underleveled, get a bad offer, or even make some bad life choices.

I don’t mean this advice as treating hiring as some kind of sick poker game, hiding your excitement or whatever – the opposite; if you are excited, then it’s fantastic; genuine positive emotions resonate well with everyone!

But **treat yourself with respect and be willing to say no and draw clear boundaries.**

Kind of... like in any healthy human relationship?

For example – if you don’t want to relocate, state it very clearly and watch out for things like sunken cost fallacy. If you don’t want any take-home tasks (I personally would not take any that’s longer than 2h of work), then don’t.

At the same time, I hope it goes without saying – **expecting respect doesn’t mean entitlement to a specific interview outcome, or even worse being unpleasant or rude** even if you think you are not treated well. Remember that for example your interviewers are not responsible for the recruiter’s behavior and vice versa.

## Starting a discussion with the recruiter

Unless applying directly to a very small company / a start-up, your interview process is going to be overseen and coordinated by a recruiter. This applies both when you apply (whether to a generic position, or a specific team), as well as when you are approached by someone. Such recruiters are not technical people, and will not conduct your interviews, but clear communication with them is essential.

*I generally discourage friends from using external recruiting agencies. I won’t go deeper into it here (some personal very bad experience – a recruiter threatening to call my current employer at that time because I wanted to withdraw from the process with them) – and yes, they can be helpful in negotiating some salaries and knowing the market, and there are for sure some helpful external recruiters and you’ll most likely some positive stories from colleagues. But remember that their incentives are not helping you or even the company, but earning the commission, and to do it, some are dishonest and will try to manipulate the both sides!*

## Understand recruiter’s role

At large tech companies your “case” (application) will be reviewed by a lot of people who don’t know each other – interviewers, hiring committees, hiring managers, compensation committees, up to VP or CEO approval. Many of them will not understand your past experience, most likely many will not even look at your CV. But they all need access to some information like this – plus feedback from interviews, talks with internal teams, managers, understand your compensation expectations, and what are your interests. There are internal systems to organize this information and make it as automatically available, but **the person who collects it, organizes it, and relays is the recruiter.** Think of them as a law attorney who builds a case.

...But they are not your attorney, and not your “friend”.

Job of a recruiter is to find a candidate who satisfies internal hiring requirements, walk them through the interviewing process, and make them sign an offer. I have definitely heard of recruiters lying to candidates and manipulating them to make it happen (most often through lying about under-leveling or putting some artificial time pressure on candidates – more on it later). Sometimes they are tasked to fill slots in a very specific team or

geographic location and will try to pressure a candidate to pick such a team, despite other, better options being available. **Recruiters often have specific hire targets/quotas as part of their performance evaluations and salary/bonus structure** – and some might try to convince or even manipulate you to accept a bad deal hit those targets.

But they are not your “enemy” either! After all, they want to see you hired, and you want to see yourself hired as well?

It’s exactly like signing any business deal – **you share a common goal** (signing this deal and you getting employed, and them finding the right employee), and some other goals are contradictory, but through negotiations you can hopefully find a compromise that both sides will feel happy about.

But what it means in practice – be as clear with communication with the recruiter as possible and actively help them build your case. Express very quickly your expectations and any caveats you might have.

At the same time, **watch out for any type of “bullshit”, manipulation, or pressure practices**. Watch out for some specific agenda like trying to push you towards a specific team or relocating if you don’t want it. If you have contacts inside of the company, you can try verifying some information that the recruiter gives you and seems questionable.

## Do you need a recommendation / referral?

I get often asked by friends or acquaintances to refer them as they believe it will make it easier to get hired. Generally, this is not the case. I still happily do that – to help them; and there is a few thousand dollar referral bonus if a referral is hired.

*Worth noting that hiring in tech is extremely costly. Flying candidates, the amount of interviews and committee time, this adds up. There is also a huge cost of having teams that need more employees, and cannot find them. Interviewing obviously costs also candidates (their own time, which is valuable; plus any potential other missed opportunities). But how costly is it in practice in dollars? External hiring agencies typically take 3 monthly candidate salaries as commission – so this goes into multiple tens of thousands of dollars. And this is just for sourcing; in addition to internal hiring costs! So a few thousands for an internal referral – especially that such a referral can be much higher quality than external sourcing – is not a lot.*

But **in practice referrals of someone I didn’t work with seem to not matter and are immediately rejected by recruiters**. In a referral form, there is an explicit question asking to explain if I worked with someone, how long etc. And it seems that referrals of people one didn’t work with are not enough to get past even the recruiter. Which seems expected.

On the other hand, **if you can find a hiring manager** inside the company to get you referred and want to get you hired on their team, this **can increase your chances a lot**. This almost guarantees going past the recruiter and directly towards phone screens. Such a hiring manager can help you get targeted for a higher level, and add some weight for the final hiring committee decision – but not a guarantee. Some hiring managers try to “push” for a hand-picked panel of interviewers and convince them to ask easier questions, and it’s obviously questionable ethically and only sometimes works, but sometimes backfires.

Just don’t ask your friends to find such a hiring manager for you, it’s not really possible (especially in a huge company), and puts them in an uncomfortable position...

**...but it’s totally ok for you to reach out to people managing teams asking them directly to apply to work with them**, instead of recruiters or official hiring forms. But it’s also ok for managers to ignore you, don’t be upset if that happens.

## Do not get underleveled

This might be the most important advice before starting the interviewing – understand company level structure and the level you are aiming for and what the recruiter envisions for you (typically based on a shallow read of your CV and simple metrics like career tenure and obtained education degree).

All companies have job hierarchy ladders; in small companies it's just something like “junior – regular – senior”, in large companies it will be a number + title. See something like [levels.fyi](https://levels.fyi) for levels salaries and equivalences between different companies.

Why do you need to do this before interviews start? Because **interviewers are interviewing you for a target level and position!**

A very common story is someone starting a hiring process, talking with a recruiter, doing phone screens, on-site interviews, and around the time of getting an offer... “I want to be hired as a senior programmer” and the recruiter says with a sadness in their voice *“but senior is L5, and you were interviewed for L4. I am sorry. You would need to go through the whole interviewing process again. And we don't have time. But don't worry. If you are really, truly an L5 candidate, you will quickly get promoted to L5!”*.

**This is misleading at best and a lie at worst and a trap that numerous people fall into.** Most people I know fell into it. I was also one that got definitely underleveled (both based on feedback from peers plus it got “fixed” with a promotion a year later). I have seen many brilliant, high performance people with many years of past career experience on ridiculously low levels only because they didn't negotiate it.

“Quickly get promoted” is such an understatement of how messy promotion can be (this is a topic beyond this post) and will cost you demonstrable “objective” achievements including product launches (it's not enough that you're a high performer), getting recommendations and support from numerous high level peers that you need to get to know and demonstrate your abilities to, your case passing through some committees – and will take at least a year since you got hired.

But what's even worse, after your promotion, it's almost impossible to get another one for two more years -> so **by agreeing to a lower level, you are most likely removing those two years from your career at that company.**

Is level important in practice? Yes and no. **Your compensation and bonuses depend on it.** In the case of equity, it can be significantly better on a higher level – and your initial grant is given you typically for the whole first four (!) years. Some internal position availability might depend on it (internal mobility job postings with strict L5/L6/L7 requirements). On the other hand, even a low level doesn't prevent you from leading projects, or doing high impact, cool work.

But my general advice would be – rather don't agree to a lower level; even if you don't care too much about compensation and stuff like that (for me it's secondary, we are extremely privileged in tech that even “average” paying salaries are still fantastic), it's possible you would later be bitter about it and less happy because of something so silly and not related to your actual, daily work.

**So explain your expectations right away, before the interviews start** – if you want to be a “senior” don't agree to signing a contract for another position with a “promise of future re-evaluation”; it doesn't mean anything.

Yes it might mean ending the hiring process earlier, but remember the advice about being non-desperate and respecting yourself? It's ok to walk away.

*Some digression and side note – I think under-leveling is one of the many reasons for gender salary disparity. All big corps claim they pay all genders equally for the same work, and I don't have a reason to question their claims, but most likely this means per “level” – but it doesn't take*

*into account that a gender might get under-leveled compared to another one. Whether from recruiter's bias, or cultural gender norms that make some people less likely to fight for proper, just treatment – but under-leveling is in my opinion one of the main reasons for gender (and probably other categories) pay gap. Obviously not the fault of its victims, but the system that tries to under-level “everyone”, yet some are affected disproportionately. Discrimination is often setting the same rules for everyone, but applying them differently.*

## How long does the whole process take?

Interviews can drag for months. This sucks a lot – and I experienced it both from some giants, as well as some small companies (at my first job, it took me 1.5 years from interviewing to getting an offer... though you can imagine it was due to some special circumstances). Or it can be super fast – I have also seen some giants moving very fast, or smaller companies giving me the offer the same day I interviewed and while I was still in the building.

But take this into account – especially around any holidays, vacations season etc. it can take multiple months and plan accordingly. Typically after each stage (talking with the recruiter, phone screen, on-site interviews, getting an offer) there are 1-15 days of delay / wait time... **At Google it took me roughly ~5 months from my first chat with a recruiter and agreeing for interviews to signing my offer.** And I have heard some worse timing stories.

This is especially important when interviewing at multiple places – worth letting recruiters know about it to avoid getting completely out of sync. Similarly important if you are under financial pressure.

Note that if you need to obtain a visa, everything might take even a year or a few longer. I won't be discussing it in this guide – but immigration complicates a lot and can be super stressful (I know it from first hand experience...).

## Why are the whiteboard “coding” interviews the way they are?

Before going toward some interviewing advice, it's worth thinking about why tech coding (often “whiteboard” pre-pandemic) interviews that get so much hatred online are the way they are. Are companies acting against their own interest for some irrational reason? Do they “torture” candidates for some weird sense of fun?

(Note: I am not defending all of the practices. But I think they are reasonable for the companies once you consider the context and understand they are for them, and not for candidates.)

Historically, companies like Facebook or Google were looking for “brilliant hackers”. People who might lack speciality, but are **super fast to adapt to any new problem**, quickly write a new algorithm, or learn a new programming language if needed. People who are not assigned to a specific specialty / domain / even team, but are encouraged to move around the company every two years, adding fresh new insights, learning new things, and spreading the knowledge around. So one year a person might be working on a key-value store, another year on its infrastructure, another year on message parsing library, and then another year developing a new frontend framework, or maybe a compiler. And it makes sense to me – one year XML was “hot”, another one JSON, and another one proto buffers – knowing such a technology or not is irrelevant.

A lot has changed since then and now those companies don't have hundreds of engineers, but hundreds of thousands, but I think **the expectation of “smart generalists”** mostly stayed until recently and it's changing only now.

In such a view, you don't care for coding experience, knowing well any particular domain, programming languages or frameworks; instead **you care for the ability to learn those**; quickly jump onto new abstract problems that were not solved before, and hack a solution before moving to new ones.



Then testing for “problem solving” (where problems are coding algorithmic problems) makes the most sense. Similarly, asking someone to spend 2 weeks catching up on algorithms (most of us have not touched it since college...) seems like a proxy for quick learning ability.

Those companies also did decades of research into hiring and what correlates with future employee performance and found that yes, such types of interviews they believe represents “general cognitive ability” (I think it’s an unfortunate term) for software engineers correlates well with future performance.

Another finding was that it doesn’t really matter if the process is perfect, but the more interviewers there are and they are in consensus, the better the outcomes. This shouldn’t be surprising from just a mathematical / statistical point of view. So **you’d have 4-5 problem solving interviews with algorithms and coding**, and even if some are a bit bs, if interviewers agree that you should be hired, you would.

The world has changed and the process is changing as well – luckily. 5-6y ago, you would get barely any domain knowledge interviews – now you get some, especially for niche domains. At Google, there were no behavioral interviews testing *emotional intelligence* (who cares about hiring an asshole, if they are brilliant, right? This can end badly...) – now luckily they are mandatory.

But those 6y ago I heard stories of someone targeting a graphics related team / position, having purely graphics experience, and failing the interview because of a “design server infrastructure of [large popular service]” systems design question (as I am sure an interviewer believed that every smart person should be able to do it on the spot; somehow I am also sure they worked on it...).

I think this unfortunate reality is mostly gone, but if you are unlucky – it’s not your fault.

## Coding interviews are not algorithms knowledge / trivia interviews

One most common misconception I see on the internet are outraged people angry at “I have been coding for 10 years and now they tell me to learn all these useless algorithms! Who cares how to code a B-tree, this is simple to find on the internet, and btw I am a frontend developer!”.

Let me emphasize – generally nobody expects you to memorize how to code a B-tree, or know Kruskal’s algorithm. Nobody will ask you about implementing a specific variant of sorting without explaining it to you first.

### **Those interviews are not algorithm knowledge interviews.**

They are not trivia / puzzles either. If you spend weeks learning some obscure interval trees, you are wasting your time.

You are expected to use some most simple data structures to solve a simple coding problem.

I will write my recommendations on all that I think is needed in a further section, but the crux of the interview is – having some defined input/output and possibly an API, can you figure out how to use simple array loops, maybe some sorting, maybe some graph traversal to solve the problem and analyze its complexity? If you can discuss trade-offs of a given solution, propose a few other ones, discuss their advantages, and code it up concisely – this is exactly what is expected! Not citing some obscure kd-Tree balancing strategies.

And yes, every programmer should care about the big-O complexity of their solutions, otherwise you end up with accidentally quadratic scaling in production code.

So while you don’t need to memorize or practice any exotic algorithms, you should be able to competently operate on very simple ones.

This might get some hate, but I think every programmer should be able to figure out how to traverse a binary tree in a certain order after a minute or two of thinking (and yes, I know that it’s different in a stressful interview

environment; but this applies to any kind of “thinking”! Interviews are stressful, but it doesn’t mean one should not have to “think” during one) – this is not something anyone should memorize.

Similarly, any competent programmer should be able to “find elements of the array B that are also present in an array A”.

And we’re really talking mostly about this level of complexity of problems, just wrapped up in some longer “story” that obscures it.

## Randomness of the interview process

Worth mentioning that those large companies have so many candidates that they consider acceptable to have multiple false negatives; or even a false negative ratio order of magnitude higher than false positives. It was especially true in their earlier growth stages when they just became famous, but were not adding tens of thousands people, but just tens a year. In other words – if they had one hiring slot, and 10 truly brilliant candidates in a pool of 1000 applicants, it used to be “ok” to reject 5 of them prematurely – they were still left with 5 people to choose from.

When interviewing for a job, you will get a panel of different interviewers. In small companies, those people know each other. In huge companies, most likely they don’t and could be random engineers from across the company.

*Being interviewed by random engineers is supposed to be “unbiased”, but in my opinion it sucks for the candidate – in smaller companies or when being interviewed by the team directly, the interview goes both ways; they are interviewing you and you are interviewing them and deciding if you would like to work with those people, chatting with them, sensing personalities, and what they might work on based on the questions. If the size of a company is over 100k people and you are assigned random interviewers, this is random and useless for judging – as a bad interviewer is someone you are likely to never even meet...*

For some of those interviewers, it might be even the first interview they are doing (sadly the interviewer training is very short and inadequate due to general interviewer shortage). Some interviewers might even ask you questions that they are not supposed to ask (like some “math puzzles” or knowledge trivia that you either know or not – those are explicitly banned at most big tech corps, and still some interviewers might ask them by their mistake). **So if you get some unreasonable questions – it’s just the “randomness” of the process and I feel sorry for you.** Luckily, it’s possible that it will not matter for the outcome, so don’t worry – if you ace the other interviews, a hiring committee that reviews all interviewer feedback will ignore one that is an outlier from a bad interviewer or if they see that a bs question were asked.

But if you don’t get an offer and your application gets rejected – it can also be expected and not your fault; it doesn’t say anything about your skills or value.

The interview process is not designed for judging your “true” value, it’s designed for the company goals – efficiently and heavily filtering out too many candidates for too few open positions. Other world class people were rejected before you – it happens.

So don’t worry about it, don’t take it personally (it’s business!), look at other job opportunities or simply reapply at some later stage.

## Shared advice

Regardless of the interview type (problem solving/coding, knowledge, design, behavioral) there is some advice that applies across the board.

## Time management

It's up to both you and your interviewer to do the time management of the interview. Generally, interviewers will try to manage time, but sometimes they can't do much if you actively (even unknowingly!) obstruct this goal.

**What is the goal of an interview?** For you it's to show as many strengths as possible to support a positive hiring decision. For the interviewer is to poke in many directions to find both your strengths and the limits of your skills/knowledge/experience that would determine your fit for the company/team.

Nobody knows everything to the highest level of detail. Everyone's knowledge has holes – and it's fine and expected, especially with the more and more complicated and specialized tech industry.

The interviews are short. In a 45min interview, you have realistically 35mins of interview time – and a bit better 50min in a 1h interview. Any wasted 5mins is time that you are not giving any evidence to support a hire.

Therefore, **be concise and to the point!**

Don't go off the tangent. Avoid “fluff” talk, especially if you don't know something. This is a natural human reaction, trying to say “something” to not show weakness. But this “fluff” will not convince an expert of any strength (might even be considered negative).

Don't be afraid to ask clarifying questions (to not waste time on going in the wrong direction) or saying you have not seen much of a given concept, but can try figuring out the answer – typically the interviewer will appreciate your try to challenge yourself, but change the topic to something more productive.

**Humility and being able to say “I don't know” also make for great coworkers.**

If someone tries to bullshit and pretend they know something they don't is a huge red flag and a reason for rejection on its own.

## Communication

**Make your answers interactive and conversational.**

This is expected from you both at work as well as during the interview process. It is explicitly evaluated by your interviewers.

Explain your decisions and rationale.

Justify the trade-offs (even if the justification is simple and pragmatic “here I will use a variable name ‘count’ to make the code more concise and fit on the whiteboard, in practice would name it more verbosely.”).

Explain your assumptions – and ask if they are correct.

Ask many clarifying questions for things that are vague or unclear – just like at work you would gather the requirements.

If you didn't understand the question, express it (helping the interviewer by asking “did you mean X or Y?”).

As mentioned above, don't add fluff and flowery words and sentences.

Generally, be friendly and treat the interviewer like a coworker – and they might become one.

## Being respectful

I mentioned that you should expect to be treated respectfully. Any signs of belittling, proving you are less worthy, ridiculing your experience or takes are huge red flags.



It's ok to even express that and speak up.

But **you also absolutely have to be polite and respectful** – even if you don't like something.

Remember that your interviewers are people; and often operate within company interviewing rules/guidelines.

Don't like the interview question, think it's contrived, doesn't test any relevant skills?

It happened to me many times. But keep this opinion to yourself. It's possible that you misunderstood the question, its context, or why the interviewer asked it (might be an intro and segway to another one).

Similarly, **nobody cares about your “hot takes” “well actually” that disagree with the interviewer on opinions** (not facts) – keep to yourself if you think that linked lists are not useful in practice (they are), that C++ is a garbage language (no, it's not), it's not necessary to write tests (hmm...), or that complexity analysis is useless (what?).

What purpose would it serve, to show “how smart you are”?

Instead it would show that you are a pain in the ass to work with if even during an interview you cannot resist an urge to argue and bikeshed – then how bad it has to be in your daily work?

## Problem solving (“coding”) interviews

Let me focus first on the most dreaded type of the interviews that gets most of the bad reputation. I have already explained that they are not algorithm knowledge interviews, and here is some more practical advice.

### Picking interview programming language

Most big companies nowadays let you use any common language during the interviews – and it makes sense, as remember, they don't look for specialty / expertise and believe that a “smart” candidate can learn a new language quickly; plus at work we use numerous languages (I wrote code in 6 different ones at work in 2021).

So you can pick almost any, but I think some choices are better than the others. I can give you some conflicting advice, use your judgment to balance it out:

- **Something concise like Python gives you a huge advantage.** If you can code a solution in 10 lines instead of 70, you are going to be evaluated better on interview efficiency and get a chance to move further ahead in the process. You are likely to make less bugs. And it's going to be much easier to read it multiple times and make sure it works correctly. If you know Python relatively well, pick it.
- **You do not want to pick a language you don't feel confident in**, or are likely to use incorrectly. So if you write mostly C/C++ code or Java, I would not recommend picking Python only because it's more concise. It can backfire spectacularly.

### Does coding style matter?

Separate point worth mentioning – generally interviewers should not pay too much attention to things that can be googled easily – like library function names etc. It's ok to tell the interviewer that you don't remember and propose to use the name “x” and if you are consistent, it should be fine.

But I have seen some interviewers being picky about coding style, idiomatic constructs etc. and giving lower recommendations. I personally don't approve of it and I never pay attention to it, but because of that I cannot give advice to ignore the coding style – it might be a good idea to ask your interviewer what their expectations are.

And it's yet another reason to not pick a language you don't feel very confident in.

## Must-learn data structures / algorithms

While you are not expected to spend too much time on learning algorithms, below are some basics that I'd consider CS101 and something that you could “have to” know to solve some of the interview problems.

Note that by “know” I don't mean read the wikipedia page, but to know how to use them in practice and have actually done it. It's like riding a bike – you don't learn it from reading or watching youtube tutorials. So if you have not used something for a while, **I highly recommend practicing using those simple algorithms in your language of choice** with something like [advent of code](#) or [leetcode](#).

### Hash maps / dictionaries / sets

**Most problem solving coding interview problems can be solved with loops and just a simple unordered hash map.** I am serious.

And rightfully so; this is one of the most common data structures and whole programming languages, databases (no-sql-like) and/or infrastructures are based on those! Know *roughly* how ordered / unordered one works and their complexity (amortized). Extra points for understanding things like hash collisions and hash functions, but no reasonable interviewer should ask you to write a good one.

### Sorting

**Don't learn different sorting algorithms, this is a waste of time.**

Know +/- how to implement one or two, their complexity (you want amortized  $O(N \log N)$ ), but also understand trade-offs like in-place sorting vs one that allocates memory.

Only sorting-like algorithm that I think is worth singling out as something different is “[counting sort](#)” (though you might not even think of it as a sorting algorithm!) – a simpler variant similar to radix sort that is only  $O(N)$  – but works for specialized problems only; and some of problems I have seen can take advantage of counting sort + potentially a hash map.

Having a sorted array, **be able to immediately code up binary search**, those come up relatively often in phone screens – watch out for off-by-one errors!

### Graphs

Do not memorize all kinds of exotic graph algorithms.

But be sure to know how to implement graph construction and traversal in your language of choice without thinking too much about it.

**For traversal, know and be able to immediately implement BFS and DFS**, and maybe how to expand BFS to a simple Dijkstra algorithm.

If you don't have too much time, don't waste it on more advanced algorithms, spanning trees, cycle detection etc. I personally spent weeks on those and it was useless from the perspective of interviews, and at Google I have not seen any interviewer asking those. This wasn't wasted time for me personally, I learned a lot and enjoyed it, but was not useful preparing for the interviews.

### Trees

Know how to write and traverse binary tree; how to do simple operations there like adding/removing. Know that trees are specialized graphs.

The most sophisticated tree structure that will pop up in some interviews (though it was so common that most companies “banned” it as a question) is a trie for spell checks – but I think it’s also possible to derive / figure it out manually, and it might not be an optimal solution to the problem!

## Converting recursion to iteration

It’s worth understanding how to convert between recursion based approaches and iteration (loop) based. This is something that rather rarely comes up at work, but might during the interview, so play around with for example DFS implementation alternatives.

## Dynamic programming

This one is a bit unfortunate I think – as dynamic programming is extremely rare in practice, I think through 12y of my professional career and solving algorithmic problems almost every day I had only a single use for it during some cost pre computations – but dynamic programming used to be interview favorites...

I was stressed about it, and luckily spent some time learning those – and 6 out of 10 of my interviewers in 2017 asked me problems that could be solved optimally with dynamic programming in an iterative form.

Now I think it has luckily changed, I haven’t seen those in wide use for a while, but some interviewers still ask those questions.

## Understanding the problem

I do only on-site interviews (which for the past two years were virtual, blurring the lines... but are still after “phone screens”), so I get to interview good candidates and generally, I like interviewing – it’s super pleasant to see someone blast through a problem you posed and have a productive discussion and a brainstorming session together.

One of the common reasons even such good candidates can get lost in the weeds and fail is if they didn’t understand the problem properly.

Sometimes they understand it at the beginning, but then forget what problem they were solving during the interview.

If the interviewer doesn’t give you an example, **ALWAYS write a simple example and input/expected output in the shared document or on the whiteboard.**

I am surprised that maybe less than a third of the candidates actually do that.

This will not only help you if you get stuck (might notice some patterns easier), but also an interviewer could help clarify misunderstanding, and it can prevent getting lost and forgetting the original goals later.

It can also serve as a simple test case.

## Testing

After you propose and code a solution, very quickly run it through the most simple (but not degenerate) example and test case. In my experience, **if a candidate has a bug in their code and they try to test it, 90% of them will find those bugs** – which is always a huge plus.

But testing doesn’t stop there.

Code testing is an ubiquitous coding practice now, and big tech companies expect tests written for all code. (And yes, you can test even graphics code)

So be sure to explain briefly how you would test your code – what kind of tests make sense (correctness, edge cases, performance/memory, regression?) and why.

## Behavioral interviews

Behavioral interviews are one of the older types of the interviews; they come with different names, but in general it's the type of the interview where your interviewer looks at your CV, and asks you to walk them through some past projects; often asking questions like “tell me about a situation, when you...” and then questions about collaboration, conflicts, receiving and giving feedback, project management, gathering requirements, in the recent years diversity/inclusion... basically almost anything.

**The name “behavioral” comes from the focus on your (past) behaviors**, and not your statements.

This is the type of the interviews that are the most difficult to judge – as they attempt to judge **emotional intelligence** among other things.

On the one hand, I think we absolutely need such interviews – they are necessary to filter out some toxic candidates, people who do not show enough maturity or EQ for a given role, types of personalities that won't work well with the rest of your team, or maybe even people with different passions/interests who would not thrive there.

On the other hand, they are extremely prone to bias, like interviewers looking for candidates similar to themselves (infamous “culture fit” leading to toxic tech bro cliques).

Also the vast majority of interviewers are totally under qualified and not experienced enough to judge interpersonal skills and traits from such an interview (hunting for subtle patterns and cues) – experienced good managers or senior folks who have collaborated with many people often are, but assuming that a random engineer can do it is just weird and wrong. Sadly, not much you can do about it.

Google didn't do those interviews until I think 2019 because of those reasons – but it was also bad.

**Not trying to filter out assholes leads to assholes being hired.**

Or someone who is a brilliant coder or scientist, but a terrible manager getting hired into a managing position.

Note that some companies also mix behavioral interview with knowledge interviews and judge your competence based on them (this was true at every gamedev company I worked for) – while some others treat it only as a behavioral filter.

I suggest for you as an interviewee treating it as both – showing both your competence (whether expertise, engineering, managerial, or technical leadership) and confidence, as well as personality traits.

### There are no “perfect” answers

From such a broad description it might seem that such an interview is about figuring out the “right” answers to some questions.

I think this is not true.

**There are some clearly bad and wrong answers** – examples of repeated, sustained toxic behaviors, bullying, arrogance – **but I don't think there are perfect ones**, as this depends a lot on the team.

Example – teamwork. In some teams, you are expected to collaborate a lot with the whole team, having your designs reviewed almost daily, splitting and readjusting tasks constantly. Some personalities thrive in such an environment, some don't and need more quiet, focused time.

On the other hand, on some teams there is a need for deep experts in some niche domain who can go on their own for a long period of time – driving the whole topic and involving others only when it’s much more mature. Some other personalities might thrive in this environment, while it might burn out the first type.

Another example – management. A relatively junior team needs a manager that is fairly involved and helps not only mentor, but also with some actual work and code reviews, while a more mature team needs a manager who is much more hands-off, less technically involved, and mostly deals with helping with team dynamics and organizational problems.

So “bullshitting” in such an interview (which is a very bad idea, see below), even if it works, can lead to being assigned to some role that won’t make you happy...

## Prepare

This might be a vague and “everything goes” type of interview, but it doesn’t mean you cannot prepare for it. You don’t want to be put on the spot being asked about something that you have never thought about and having to come up with an answer that won’t be very deep or representative.

Two ways to prepare – the first one is to **simply look online what are the typical questions asked in such an interview**, note down ones that surprised you, are tricky, or you don’t know the answer.

The second one, more important, is to **go through your CV and all projects** – and think how they went, what you did, what did you learn, what went wrong and could have been better. It’s easy to recall our successes, but also think about unpleasant interactions, conflicts, and what has motivated the person you didn’t agree with.

Spend some time thinking about your career, skills, trajectory, motivations – and be honest with yourself. You can do this also while thinking about the typical behavioral interview questions you found online.

## Focus on behaviors, not opinions

When asked about how you gather requirements for a new task, don’t give opinions on how it should be done. This is something anyone can have opinions on or read online.

Anchor the answer around your past project, explain how you approached it, what worked, and what didn’t.

Generally **opinions are mostly useless for interviewers** – and if you simply don’t have experience with something, explain it – remember the time management advice.

## Be honest – no bullshit

Finally, absolutely **don’t try to bullshit or make up things**.

It’s not just about ethics, but also some basic self-preservation.

The industry is much smaller than you might think, and an interviewer could know someone who could verify your story.

Made-up stories also fall apart on details, or could be figured out by things that don’t match up in your CV.

*Two situations have stuck in my memory. One was a “panel” interview a long time ago with two of my colleagues and interviewing a programmer who put in their CV something they didn’t really understand, and it caused one of the interviewers to almost snap and grill the interviewee about technical details to the point that they were about to completely break down. They got stopped by another interviewer (senior manager). This was very unprofessional of my colleague and made*

*everyone uncomfortable – I didn’t speak up as I was pretty junior back then and feel bad about it now. But remember that bluffs might be called.*

*A second one is talking with someone who bragged about their numerous achievements they did personally on a “very small team, maybe 2-3 programmers other than them, mostly juniors”. Not too long time later I met their ex-manager and it turned out that this person did maybe 10% of what they bragged about, and the team was 15 people. This was... awkward.*

Obviously, **you don’t have to be self incriminating** – that one time you snapped and were unprofessional 5y ago? Or that time you were extremely wrong and didn’t change your mind even being faced with the evidence, until it was too late? You don’t have to mention it.

At the same time, if something you did was imperfect or turned out to be a mistake, but you learned your lessons actually makes for a great answer – as it shows introspection, insight, growth, and self-development. **Someone who makes mistakes, but recognizes them, shows constant learning and growth is a perfect candidate**, as their future potential self is even better!

## Be clear about attribution, credit, and contribution

This is expansion of the previous point – when discussing team projects, be very clear what were your personal exact contributions, and what were contributions of the others.

If you vaguely discuss “we did this, we did that”, how is the interviewer supposed to understand if it was your work, your colleagues, who made decisions, and what the dynamics looked like?

**Don’t bloat your contributions and achievements, but also don’t hide them.**

If you made most of the technical decisions, even if you were not officially a lead – explain it.

At the same time, if someone else made some contributions, don’t put it under vague “we”, but mention it was the work of your collaborator. **Give proper credit.**

## Knowledge / domain interviews

Knowledge interviews are one of the oldest and still most common types of interviews. When I was working in video games, every company and team would use knowledge interviews as the main tool. They ranged from legit poking at someone’s expertise – very relevant to their daily work; through some questionable ones (asking a junior programmer candidate fresh from college “what is a vtable”); to some that were just plain bad and not showing almost anything useful (“what does the keyword ‘mutable’ mean?”).

I think this kind of interview can often give super valuable hiring signals (especially when framed around CV/past experience or together with “problem solving” / design).

On the other hand, it can also be not just useless, but also reinforcing some bias and intimidating – when the interviewer grills the interviewee on some irrelevant details that they personally specialize in. Or when they want to hire someone with exactly the same skills like they have – while the team actually needs a different set of skills and experience.

So as always, it depends a lot on the experience and personality of the interviewer.

## What makes a good domain knowledge question?

In my opinion it is an open ended nature and potential for both depth and breadth and allowing candidates of all levels to shine according to their abilities.



So for example for a low level programmer a question about “if” statements can lead to fascinating discussions about jumps, branches, branch prediction, speculative execution, out of order processors, vectorization of if statements to conditional selects, and many many more.

By contrast a lazy, useless domain knowledge question has only a single answer and you either know it or not – and it doesn’t lead to any interesting follow up discussions.

Still, it was quite a surprise to learn that Google used to not do any of those a while ago – I explained the reasons in one of the sections above. Now the times have changed, and Google or Facebook will interview you on those as well – not sure about some introductory roles, but that’s definitely the case in specialized organizations like Research.

**Before the interviews, talk with the recruiter what kind of domain interviews you will have** – which domains should you prepare to interview for and what the interviewers expect. It’s totally normal, ok and you should ask it! Don’t be shy about it and don’t guess.

Even for a gamedev “engine” role you could be interviewed in a vast range of topics from maths, linear algebra, geometry, through low level, multithreading, how OS works, algorithms and data structures, to even being asked graphics (some places use the term “engine” programmer as “systems” programmer, and in some places it means a “graphics” programmer!).

*A fun story from a colleague. They mentioned that straight after college, they applied to a start-up doing quant trading, but there was some miscommunication about the role and the interviews – they thought they applied for an entry level software engineer position, while it was for a junior quant researcher. Whole interview was in statistics and applied math and they mentioned that the interviewer (mathematician) quickly realized the miscommunication and tried to back off from intimidating questions and rescue the situation by finding some kind of simple question “so... tell me what kind of statistical distributions do you know?” “hmm... I don’t know, maybe Gaussian?”. They were totally cool about it, but to avoid such situations, make sure you communicate properly about domain knowledge expectations.*

## Refresh the basics

I’ll mention some of the knowledge worth refreshing before domain interviews I gave or received (graphics, engine/low-level/optimization, machine learning), but something that is easy to miss in any domain interview – **refresh the absolute basics**. You know, undergrad level textbook type of definitions, problems, solutions.

Whether chasing the state of the art or simply solving practical problems, it’s easy to forget some complete basics of your domain and “beginner” knowledge – things you have not used or read about since college.

I think it’s always beneficial to refresh it – you won’t lose too much time (as you probably know those topics well, so a quick skim is enough), but won’t be caught off guard by an innocent, simple question that the interviewer intended just as a warm-up.

Getting stuck there for a minute or two is not something that would make you fail the interview, but it can be pretty stressful and sometimes it’s hard to recover from such a moment of anxiety.

## Graphics domain interviews

Graphics can mean lots of different things (I think of 3 main categories of skills – 1. high level/rendering/math, 2. CPU rendering programming and APIs, 3. low level GPU / shader code writing and optimizations), but here are some example good questions that you should know answers to (within a specialty; if you are an expert on Vulkan CPU code programming, no reasonable interviewer should be asking about BRDF function reciprocity; and it’s ok for you to answer that you don’t know):

- What happens on the CPU when you call the Draw() command and how it gets to the GPU?
- Describe the typical steps of the graphics pipeline.
- Your rendering code outputs a black screen. How do you debug it?
- What are some typical ways of providing different, exclusive shader/material features to an artist? Discuss pros and cons.
- A single mesh rendering takes too much time – 1ms. What are some of the possible performance bottleneck reasons and how to find out?
- Calculate if two moving balls are going to collide.
- Derive ray – triangle intersection (can be suboptimal, focus on clear derivation steps),
- What happens when you sample a texture in a shader? (Follow up: Why do we need mip maps?)
- How does perspective projection work? (Follow up for more senior roles, poke if they understand and can derive perspective correct interpolation)
- What is Z-fighting, what causes it, and how can one improve it?
- What is “culling”, what are some types and why do we need those?
- Why do we need mesh LODs?
- When rasterization can outperform the ray tracing and vice versa? What are the best case scenarios for both?
- What are pros/cons of deferred vs forward lighting? What are some hybrid approaches?
- What is the rendering equation? Describe the role of its sub-components.
- What properties should a BRDF function have?
- How do games commonly render indirect specular reflections? Discuss a few alternatives and compare them to each other.
- Tell me about a recent graphics technique/paper you read that impressed/interested you.

## Engine programming / optimization / low level domain

It’s been a while since I interviewed for such a position, and similarly long time since I interviewed others, but some ideas of topics to know pretty well:

- Cache, memory hierarchies, latency numbers.
- Branches and branch prediction.
- Virtual memory system and its implications on performance.
- What happens when you try to allocate memory?
- Different strategies of object lifetime management (manual allocations, smart pointers, garbage collection, reference counting, pooling, stack allocation).
- What can cause stalls in in-order and out-of-order CPUs?
- Multithreading, synchronization primitives, barriers.
- Some lock-free programming constructs.
- Basic algorithms and their implications on performance (going beyond big-O analysis, but including constants, like analysis of why linked lists are often slow in practice).
- Basics of SIMD programming, vectorization, what prevents auto-vectorization.

Best interviews in this category will have you optimize some kind of code (putting your skills and practical knowledge to test). Ones I have been tasked with typically involved:

- Taking something constant but a function call outside of a loop body.
- Reordering class fields to fit a cache line and/or get rid of padding, and removing unused ones (split class).
- Converting AoS to SoA structures.
- Getting rid of temporary allocations and push\_backs -> pre-reserve outside of the loop, possibly with alloca.
- Removing too many indirections; decomposing some structures.
- Rewriting math / computations to compute less inside loop body and possibly use less arithmetic.
- Vectorizing code or rewriting in a form that allows for auto vectorizing (be careful; up to discussion with the interviewer).
- (Sometimes) adding “\_\_restrict” to function signatures.

- Prefetching (nowadays might be questionable).

**Important note:** Generally do not suggest multithreading until all the other problems are solved. Multithreading bad code is generally a very bad practice.

## General machine learning

This is something relatively recent to me and I haven't done too many interviews about it; but some recurring topics that I have seen from other interviewers and questions to be able to easily answer:

- Bias/variance trade-off in models.
- What is under- and over-fitting, how to check which one occurs and how to combat it?
- Training/test/validation set split – what is its purpose, how does one do it?
- What is the difference between parameters and hyperparameters?
- What can cause an exploding gradient during training? What are typical methods of debugging and preventing it?
- Linear least squares / normal equations and statistical interpretation.
- Discuss different loss functions – L1, L2, convex / non-convex, domain specific loss functions.
- General dimensionality reduction – PCA / SVD, autoencoders, latent spaces.
- How does semi-supervised learning fill in the gap between supervised and unsupervised learning? Pros/cons/applications.
- You have a very small amount of labeled data. How can you improve your results without being able to acquire any new labels? (Lots of cool answers there!)
- Explain why convolutional neural networks train much faster than fully connected ones?
- What are generative models and what are they trying to approximate?
- How would you optimize runtime of a neural network without sacrificing the loss too much?
- When would you want to pick a traditional approach over deep learning methods and vice versa? (Open ended)

## System design interviews

Systems design interviews are my favorite as an interviewer (and I thoroughly enjoyed them as an interviewee). They are required for more senior levels and comprise a design process for larger scale problems. So instead of simple algorithms, this is the type of problems like designing terrain and streaming system for a game; designing a distributed hash map; explaining how you'd approach finding photos containing document scans; maybe designing an aspect of a service like Apple Photos.

This is obviously not something that can be done in a short interview time frame – but what can be done is gathering some requirements, identifying key difficulties, figuring out trade-offs, and proposing some high level solutions.

This is the most similar to your daily work as a senior engineer – and this is why I love those.

I don't have much specific advice about this type of the interviews that would be different from the other ones – but **focus on communication, explaining boundaries of your experience, explaining every decision, asking tons of questions, and avoiding filler talk.**

*An example of one design interview I have partially failed (enough to get a follow-up one but not be rejected). An engineer that spent 10y of his career designing a distributed key-value store asked me to design one. Back then I considered it a fair question as I had focused quite a bit on this type of low-level data structures, though just on a single machine (now I would probably explain that it's not my speciality so my answers would be based mostly on intuition and not actual knowledge/experience – and the interviewer might change their question). At one point of the*

*interview, I made an assumption about prioritizing on-core performance that was a trade-off of deletion performance as well as ability to parallelize it across multiple machines – and I didn’t mention it. This caused me to make a series of decisions that the interviewer considered as showing lack of depth / insight / seniority. Luckily I got both this feedback from the recruiter, as well as had great feedback from other interviews, and was given a chance to re-do the systems design interview with another engineer that I did well on. In hindsight, if I communicated this assumption and asked the interviewer about it, it could have gone better.*

## After the interview

After the interview, don’t waste your time and mental stamina on analyzing what could have been wrong and if you answered everything correctly.

Seeking self-improvement is great, but wait for the feedback from the recruiter.

I remember one of the interviews that I thought I did badly on (as I got stuck and stressed on a very simple question and went silent for two minutes; later recovered, but was generally stressed throughout) and was overthinking it for the following week, but later the recruiter (and the interviewer – I worked with them at some later point) told me I did very well.

**Sometimes if you totally bomb one interview but do very well on others, you will get a chance to re-do a single interview in the following week.** This happened both to me with my design interview I mentioned, and to many of my friends, so it’s an option – don’t stress and overthink.

Also note that if you fail to get an offer, you can try to ask for feedback on how to improve in the future. Sadly, big companies typically won’t give it to you – due to legal reasons and being afraid of lawsuits (if someone disagrees with some judgement and claims discrimination instead).

But small companies often will and I remember how one candidate we interviewed that did well except for one area (normally it would be ok and we’d hire them, but it was also a question of also obtaining a visa) got this as a feedback and in a few months they caught up on everything, emailed us, demonstrated their knowledge, and got the offer.

## Interviewing your future team

At smaller companies and some of the giants (Apple and to my knowledge also Microsoft/nVidia), all of your interviews will be with your potential team members – an opportunity not only for them to interview you, but as importantly for you to interview them, check the vibe, and **decide if you want to work with them.**

At Google and some other giants, this happens in a process called “team matching” after your technical interviews are over and successful. At Facebook it was a month **after** you get hired during a “bootcamp” (totally wrong if you ask me – since so much of your work and your happiness depends on the team, I would not want to sign an offer and work somewhere without knowing that I found a team I really want to work with).

In my opinion, a great team and a great manager is worth much more than a great product. Toxic environments with micromanagers and politics will burn you out quickly.

In any case, at some point you will have an opportunity to chat with potential teammates and potential manager and it’s going to be an interview that goes both ways.

You might be asked both some “soft” and personal motivation/strengths/work preferences questions, as well as more technical questions.

One of my advices is to learn something about your potential future team before chatting with them and be prepared both for questions they might ask, as well as asking them ones that maximize amount of information (spending 10mins on hearing what they are working on is not a productive use of a half an hour you might have – if it's information that can be obtained easily earlier).

## Interview your prospect manager and teammates

Here are some questions that are worth asking (depending if something is important for you or not):

- How many people are on your team? How many team members (and others outside of your team) do you work closely with?
- How many hours a week do you spend coding vs meetings vs design vs reading/learning?
- How much time is spent on maintenance / bug fixing vs developing new features / products?
- Do you work overtime and if so, how often?
- Do people send emails outside of the working hours?
- How much vacation did you take last year?
- Does the manager need to approve vacation dates, and are there any shipping periods when it's not possible?
- Do you have daily or weekly team syncs and stand-ups?
- Do you use some specific project management methodology?
- What does the process of submitting a CL look like – how many hours or days can I expect from finishing writing a CL for it to land to the main repo?
- What are the next big problems for your team to tackle in the next 6/12/24 months?
- If I were to start working with you and get onboarded next week, what task/system would I work on?
- Do you work directly with artists/designers/photographers/x, or are requirements gathered by leads/producers/product managers?
- What are the 2-3 other teams you collaborate most closely with?
- Are there other teams at the company with similar competences and goals?
- How do you decide what gets worked on?
- How do you decide on priorities of tasks?
- How do you evaluate proposed technical solutions?
- When there is a technical disagreement about a solution between two peers, how do you resolve conflicts? Tell me about a particular situation (anonymize if necessary).
- Does the manager need to review/approve docs or CLs?
- How often do you have 1-1s with a skip-manager / director?
- How many people have left / you hired in the last year?
- How many people are you looking to hire now?
- How much time per year do you spend on reducing the tech debt?
- How do you support learning and growth, do you organize paper reading groups, dedicate some time for lectures / presentations, are there internal conferences?
- Do you send people to conferences? Can I go to some fixed number of conferences per year, or is there some process that decides who gets sent?
- Does your team present or publish at conferences? If not, why?
- What is the company's policy towards personal projects? Personal blog, personal articles?

## Watch out for red flags

Just like hiring managers want to watch out for some red flags and avoid hiring toxic, sociopathic employees, **you should watch out for some red flags of toxic, dysfunctional teams** and manipulating, political or control-freak micromanagers.

Some of the questions I wrote above are targeted at this. But it's kind of difficult and you as an interviewee are in a disadvantaged, extremely asymmetric situation.

Think about this – companies do background checks, reference checks, interview you for many hours, review your CV looking for any gaps or short tenures – and you get realistically 15, maximum 30 minutes of your questions where you get to figure out the potential red flags from often much more senior and experienced people (especially difficult if they are manipulative and political).

**One way to help the inherent asymmetry and time constraint is to find people who used to work at the given company / team**, and ask them about your potential manager and the reason they have left. Or find someone who works inside of the company, but not on the team, and ask them anonymously for some feedback. The industry is small and it should be relatively straightforward – even if you don’t know those people personally, you could try reaching out.

A single negative or positive response doesn’t mean much (not everyone needs to get along well and conflicts or disliking something is part of human life!), but **if you see some negative pattern** – run away!

Similarly, one question that is very difficult to manipulate around is the one about **attrition** – if a lot of people are leaving the team, and almost nobody seems to stay for more than one project, it’s almost certainly a toxic place.

Another **red flag is just the attitude** – are they trying to oversell the team? Mention no shortcomings / problems, or hide them under some secrecy? There are legit reasons some places are secretive, but if you don’t hear anything concrete, why would you trust vague promises of it being awesome?

Similarly, **vibes of arrogance or lack of respect towards you are clear reasons to not bother with such a team**. Some managers use undervaluing of the candidates and playing “tough” or “elite” as a strategy to seem more valuable, so if you sense such a vibe, rethink if you’d want to work there.

Finally, **there are some key phrases that are not signs of a healthy functioning place**. You know all those ads that mention that they are “like a family”? This typically means unhealthy work-life balance, no clear structures and decision making process, and making people feel guilty if they don’t want to give 100%. Looking for “rockstars”? Lack of respect towards less experienced people, not valuing any non-shiny work, relying on toxic outspoken individuals. When you ask for a reason to work somewhere and hear “we hire and work with only the best!”, it both means that they don’t have anything else good to say, as well as management that in its hubris will decline changing anything (after all, they think they are “the best”...).

Figure out what “needs to be done”

I mentioned this in the proposed questions, but even if a perfect team works on a perfect product, it doesn’t mean you’d be happy with tasks you are assigned.

**It’s worth asking about what are the next big things that need to get done – basically why they want to hire you.**

Is it for a new large project (lots of room for growth, but also not working on the one you might have picked the team for!), to help someone / as their replacement, or to just clear tech debt and for long term maintenance?

One not-so-secret – whatever system you work on, however temporary, will “stick” to you “forever” – especially if it’s something nobody else wants to do.

I think at every single team my “first” tasks are the things that I had to maintain for as long as they were still in use. And those things pile up and after some time, your full time job will be maintaining your past tasks.

Therefore watch out for those and **don’t treat this as just “temporary” – they might define and pigeonhole you in the new role forever.**



*My favorite funny example is my first job at CD Projekt Red and my second task (the first one was a data “cooker” – though most of it was written by the senior programmer who was mentoring me). It was optimizing memory allocations – as the game and engine were doing 100k memory allocations per frame. 😊 So after a week of work and cutting it to something around ~600 (was surprisingly easy with mostly just pooling, calling reserve on arrays before push\_backs and similar simple optimizations), it was difficult to remove them, and still slow because of some locks and mutexes in the allocator. So I proposed to write a new memory allocator, seems like a perfectly reasonable thing to do for a complete junior straight out of college? 😊 Interestingly it survived Witcher 2 and Witcher 2 X360, so maybe wasn’t actually too bad. Anyway, it was a block list allocator with intrusive pointers – so any memory overwrite on such a pointer would cause an allocator crash at some later, uncorrelated point. For the next 3 years, I was assigned every single memory overwrite bug with comments from testers and programmers “this fucking allocator is crashing again” and them rolling their eyes (you know, positive Polish workplace vibes!). I learned quite a lot about debugging, wrote some debug allocators with guard blocks that were trivial to switch, systems for tracking and debugging allocations etc., but I think there is no need to mention that it was a frustrating experience, especially under 80h work weeks and tons of crunch stress around deadlines.*

## Negotiating the offer

Finally – you went through the interviews, found a perfect team, so it’s time to negotiate your salary!

I wish every company published expected salary ranges in the offer and you knew it before interviewing – but it’s not the case; at least not everywhere.

This is yet another case of extreme asymmetry – as an applicant, you don’t have all the data that a company uses when negotiating salaries – you don’t know their ranges, bonuses, salary structure, competitor’s salaries. And they know all that, and salaries of all the employees they can compare you with.

Before negotiation, do your homework – figure out how much to expect and how much others within a similar position get paid. Example useful salary aggregator website for the US market is [levels.fyi](https://levels.fyi). You can also check salaries (though only base salary, not total compensation) of [every H1B visa company application](https://www.levels.fyi/blog/every-h1b-visa-company-application).

**If you know someone who works or used to work at some place, you can also ask them about expected salary ranges** – sadly this is something that many people treat as a taboo (I understand this, but hope it can change. If you are a friend of mine that I trust, feel free to ask me about my past or current salary. Also note that in Poland or California and many other places it is illegal for employers to ban employees from disclosing their salaries).

Salary negotiation is kind of like a business negotiation – business will (in general; there are obviously exceptions, and some fantastic and fair company owners) want to pay you the lowest salary you agree to / they can offer you. After all, capitalism is all about maximizing profit. Unfortunately, unlike a business deal, I think salary is much more – and not “just” your means of living, but also something that a lot of people associate personal value with and we expect fairness (example: tell someone that their peer at the same company earns 2x more while doing less work and being less skilled and watch their reaction).

*I am pretty bad at negotiations – I was raised by my parents in a culture of “work hard, do great stuff, and eventually they will always reward you properly”. They would also be against credits/mortgages, investments, and generally we’d never talk about money. I love my parents and appreciate everything they gave me, and I understand where their advice was coming from – it might have been good advice in the ‘80s Poland – but this is terrible advice in the US competitive system, where you need to “fight” to be treated right. When changing jobs, most of the time I*

*agreed to a lower salary than the one I had – believe it or not, this includes getting a salary cut when moving from Poland to Canada (!).*

But here are some strategies on how to get a much better offer even if you are as bad at negotiating money as I am.

First is the one I keep repeating – **don't be desperate, respect yourself, be able to walk away at any point if the negotiations don't lead nowhere.**

Always have competing offers / total compensation structure

The second best and most successful advice is simple – **have some competing offers and a few employers fighting for you.**

Tell all companies that you are interviewing at a few other places (obviously only if you actually do; don't bullshit – this is very easy to verify).

Then when you get some offer, present the other offers you get.

And no, it's not as bold as you might think it is and it's definitely not outrageous; it's totally acceptable and normal – when I was interviewing for both Google and Facebook, the Facebook recruiter told me that they will give me an offer only after I present the Google one (!). Google's offer was pretty bad initially, Facebook offered me almost ~2x of total compensation. After I came back to Google, they have equalized the offers...

Similarly, a few years earlier Sony gave me a fantastic offer (especially for gamedev), but it was only after I told them of the Apple offer I had around that time – and I'm sure it has influenced it.

Worth understanding here is the compensation structure. **In big tech corps, the base salary is not very flexible / negotiable.** You can negotiate it somewhat within ranges, but not too much (but it depends on the level! See my advice about under-leveling). Similarly target bonus, it's fixed per-level percentage. But what is negotiable, and you can negotiate a lot is the equity grant. **But as far as equity negotiation goes, sky's the limit.** The huge difference between Google and Facebook offers came purely from the equity offer difference.

Note that I don't condone applying to random places only to get some offers. You'd be wasting some people's time. **Apply to places you genuinely want to work for.** This will help you also not be desperate and have some cool options to pick from based not just on salaries, but many more choice axes!

Don't count on royalties / project shipping bonuses

Game companies sometimes might tell you that the salary might seem underwhelming, but the project bonuses are going to be so fantastic and so much worth it!

**Don't believe in it.** I don't mean they are necessarily lying – everyone wants to be successful and get a gigantic pile of money that they might take a small part of and give to employees.

But unless you have something specific in your contract and guaranteed, this is a subject to change. Additionally, projects get postponed or canceled, and it might be many years until you see any bonus – or might never see it if the studio closes down. Game might underperform. Even if it's successful, you might get laid off after the game ships (yes, happens all the time) and not see any bonus.

There is also a very nasty practice of tying bonus to a specific metacritic score. Why it's nasty? Game execs know how much metascore a game will most likely have (they hire journalists and do pre-reviews a few months ahead!), and set this bar a few points above.

**FWIW my game shipping bonuses were 1-4 monthly salaries.** So quite nice (hey, I got myself a nice used car for one of my bonuses!), but if you work on it for a few years, it's like less than a monthly salary per year of

work – much less than big tech target annual bonuses of 15/20/25% of base salary.

And definitely doesn't compensate for brutal overtime in that industry.

## You have a right to refuse disclosing your current salary

This point depends a lot on the place you live in – check your local laws and regulation, possibly consult a lawyer (some labor lawyers do community consultations even for free) – but **in California, Assembly Bill 168 prohibits an employer from asking you about your salary history**. You can disclose it on your own if you think it benefits you, but you don't have to, and an employer asking for it breaks the law.

## Sign on bonus

Something that I didn't know about before moving to the US are “sign on” bonuses – simply a **one time pile of cash you get for just signing the contract**.

You can always negotiate some, it's typically discretionary and it makes sense in some cases – if by changing jobs you miss out on a bonus at the previous place, or need to pay back relocation at the previous company. Similarly, it makes sense if you start vesting stock grants after some time like a year – it's to have a bit more cash flow the first year.

Companies are eager to offer you some sign on bonus as it doesn't count towards their recurring costs, it's just part of the hiring cost.

But if a recruiter tries to convince you that a lower salary is ok because you get some sign on bonus, think about it – are you going to remember your sign on bonus in a year or two? I don't think so, and you might not get enough raises to compensate for it, so treat this argument as a bit of bullshit. But you could use it during negotiation.

## Relocation package

If the company requires you to relocate (and you are ok with that), they are most likely to offer some relocation package. Note that those are typically also negotiable!

If you are moving to another country, **don't settle for anything less than 2 months of temporary housing (3 months preferred)**.

You will need it – obtaining documents, driving licenses etc will take you a lot of time, and you won't have it to look for a place to rent (or buy), so getting some extra time is very useful.

If you have an option of “cashing out” instead of itemized relocation, it is typically not worth it – especially in most tech locations in the US where monthly apartment / house rental costs are... unreasonable.

## You have a right to review contract with a lawyer

If you are about to decide to sign a contract, you can review it with a lawyer (or the lowest effort option – ask your friends to review some questionable clauses – if they have seen anything similar). For example, according to the California Business and Professions Code Section 16600, “*every contract by which anyone is restrained from engaging in a lawful profession, trade, or business of any kind is to that extent void.*”. So **most non-compete clauses in California are simply bullshit and invalid**.

On the other hand, I have heard of non-poaching agreements to be not just enforceable in general, but of some cases when they did get enforced.

## Never buy into time pressure

Here’s a common trick that recruiters or even hiring managers might pull on you – try to put artificial time pressure and make you commit to a decision almost immediately.

This is a bit disgusting, reminds me of used car sellers or realtors (I remember some rental place in LA that had “this month only” monthly special for 3 years anytime I was driving by; only the deadline date was changing every 2 weeks to something close in time). Your offer will come with some deadline too.

This is human psychology – “losing” something (like a job opportunity) hurts as much even if we don’t have it yet (or might even not have decided for it!).

I know that you might be both excited, as well as afraid of “losing” the offer after so much time and effort put into it – but **it works both ways**.

**Remember that hiring great people is extremely hard right now** in tech, and who would pass on a great hire just because you waited for a week or two more? This can (very rarely) happen in a small company (if they have only 1 hiring slot and need to fill it as fast as possible), but generally won’t happen in big tech.

So yes, offers come with a deadline, but they almost always can write you a new one.

Take some time to calm down your emotions, think everything through, review your offer multiple times, consult friends and family. Or sign right away – **but only if you want**.

*Someone close to me was pressured like this by a recruiter – being told the offer will be void if they don’t decide in a day or two – while they needed just a week more. What makes it even sadder, was that it was an external agency recruiter and apparently the company and hiring manager didn’t know about it and were willing to give as much time as necessary – but the person didn’t take the offer because of this recruiter pressure and unpleasant interactions with them.*

*Or another friend of mine who was told “if you don’t start next month, you won’t get a sign on bonus”.*

*I personally regret starting a job once a few months sooner than I wanted because I got pressured into it by the hiring manager (“but I have a perfect project for you [can’t tell you what exactly], if you don’t start now, it will be done or irrelevant later” – and later their management style was as manipulative; I felt so silly about not seeing through this).*

## Fin

This is it, my longest post so far! I’ll keep updating and expanding it (especially with ideas for questions to ask future team / manager, as well as example interview topics) in the future.

I hope you found it useful and got some useful advice – even if you disagree with some of my opinions.

This entry was posted in [Code / Graphics](#). Bookmark the [permalink](#).

## 6 Responses to *Insider guide to tech interviews*



**Nhbk NHB** says:

January 9, 2022 at 23:41

Thank you so much for the post!

I am currently looking for a position at a tech giant and haven’t started interviewing yet, so this is extremely helpful, especially your own experience.

What I don’t really understand yet is why the “FAANG-sexual” community is so focused on leetcode medium-hard grind, whether such occurrences should be treated as exceptions or not, and if a BS degree is required to be able to work in EU / US.

Could you share your thoughts on these?

Thanks again!

[Reply](#)



**bartwronski** says:

January 10, 2022 at 07:20

I touch lightly upon this grinding in my post – I think people who do this, slightly misunderstand the purpose of the interviews. They fail on a different axis (like clear communication, having some expertise, or properly explaining their design decisions, or just bad luck), and all this preparation is irrelevant – causing understandable, yet misguided anger. I have also interviewed some candidates over-prepared on competitive programming and algorithms, but didn’t recommend hire for some of the other listed reasons. Problem solving interview is NOT a Leetcode competition! It is testing if someone can translate longer question into a simple algorithmic problem, code it up, explain all decisions, analyze complexity (usually very simple), test it, propose tests, and have a productive discussion with the interviewer.

Regarding the BSc: I don’t know. I have worked with people who never went to college, but most of them are very senior and with decades of experience that is much stronger than a degree. I personally don’t pay attention to any kind of degree given someone’s track record, but large companies might have specific rules (that sometimes also relate to immigration laws – harder to obtain a visa without any college degree).

[Reply](#)



**NhbK NHB** says:

January 10, 2022 at 07:38

Thanks again!



**Nils (@geekynils)** says:

January 17, 2022 at 11:48

Thank you for such an elaborate post! Just one question: You recommend Python for interviews because it allows you to write short and concise code. I actually thought that using C++ would be an advantage since most kids these days only know Python, so I could stand out from the crowd. How valuable are C++ skills these days? I know both so I have to chose.

[Reply](#)



**bartwronski** says:

January 17, 2022 at 11:55

At big tech company interviews there is not even a single evaluation rubric that could give extra “points” for such “standing out”!

And in general, the goal of coding interviews is not to test your knowledge of technologies, libraries, languages – but if you can translate your ideas to correct, working, and somewhat elegant code. There is also this assumption that anyone smart enough to pass the interview bar can learn any language, so nobody generally cares. So unless they ask you to write code in C++ / it is part of actual job requirements, I would definitely not bother.

[Reply](#)



**Nils (@geekynils)** says:

January 17, 2022 at 13:30

Very helpful, thanks!

[Reply](#)