Interview Prep Notes

Algo · Lower Level (C++/OS/HW) · Quant Dev

About these notes

Answers (where they exist) are abridged and may be inaccurate. Each section contains:

- **Conceptual questions** - questions I was explicitly asked.
- **Coding questions** - questions I was explicitly asked.
- **Extras** - additional questions / coverage checks used during prep.

Questions may not be representative of generic interviews - they're likely tailored to my background.

Quant

Coding Questions

Case-study style questions

Obscured algo questions, lots of prompting for requirements and context before implementation.

- **(Many places) Implement a simplified matching engine.** Similar to LeetCode 1801 - Number of Orders in the Backlog, often with twists.
- **k-sorted array.**
- **Optimal allocation of batched fills.** Roughly: you have internal buy/sell client orders that are executed externally every N seconds. On each fill, you want to distribute the filled quantities to the internal clients such that it is "fair" and their orders are maximally filled. This was an open-ended, case-study style question - lots of discussion about what constitutes "fairness", constraints, etc. The best place to start is via implementing an orderbook, after which you can take a deeper dive into orders by testing your ability to identify iceberg orders.
- **Implement a (very) simplified backtest system.** You get a stream of TOB data (with capture and transact time). Strategy only has one outstanding order at a time. I can't find the exact question, but doing this one will give you the background you need to understand how to handle incremental and snapshot updates.
- **Auction matching system.** Effectively: given a list of bids/asks at different prices, choose some price that maximizes execution volume.
- **Travelling salesman problem**. A common problem in CS, it was this one exactly.

Pure algo style questions

(Misc places, 2024):

- Maximum Number of Visible Points - simplified version upon prompting where we did not need to handle polar coordinate coordination, and we could assume our position is centered. You can also do this one, which I was asked about at the CME, to solidify your knowledge.
- Sliding Window Maximum.
- Extended / modified versions of:
  - Design a Food Rating System.
  - Stock Price Fluctuation.

(Misc places, 2019):

- Find Median from Data Stream.
- Modified versions of:
  - Insert Delete GetRandom O(1).
  - Insert Delete GetRandom O(1) -Duplicates Allowed.
- Jump Game IV.
- Range Sum Query -Mutable.
- Implement LRU cache.

Lower Level (C++ / OS / HW)

Conceptual Questions

Virtual inheritance (asked at many places)

- What is virtual inheritance?
- How is it implemented?
- What is a vtable / vptr?
- What is done at compile time vs. run time?
- How is the code laid out
- How is it implemented in practice?

Extension: How does C++ solve the "diamond problem"? How are the variables and methods laid out in memory for parent and child classes? What if you want to avoid using virtual in your diamond problem solution (how do you disambiguate duplicate member functions)?

Honestly, all of these can be learned if you have a basic understanding of C++, and more questions (that I wasn't asked) and answers can be found in the Quant Dev 250 learning path.

OS grab bag (asked at many places)

Most of these questions don't come up explicitly on their own. They're part of a larger discussion / sequence of questions to see if you know at a high level what the OS is doing, how it interacts with HW components, and whether you know enough to write performant code. You can learn all of this information relatively quickly using the Operating Systems learning path and C++ Concurrency learning path.

- What is the difference between **concurrency and parallelism**?
- What is the difference between a **process and a thread**? How are they related?
- When should you use processes vs. threads? What are the differences in communication and overhead? Know about **IPC** (inter-process communication), **SHM** (shared memory), etc.
- What is **user space vs. kernel space**? What is **kernel bypass**?
- What is **virtualization**?
- What is a **page** in the context of the OS? What is a **TLB**? How are they related?
- Different synchronization mechanisms.
- **(HRT 2024):** "Explain to me, in as much detail as possible, how modern computers achieve parallelism." Basically a brain-dump and follow-up questions for ~10 minutes -covers many of the above points.

Open question: Why is C++ faster than Python?

Ended up being a 15–20 minute discussion that touched on garbage collection, compiler optimizations, interpreted vs. compiled languages, memory access patterns, OS, etc.

(Cit 2024) C++ trivia grab bag

- Types of casting (`reinterpret_cast`, `const_cast`, `static_cast`, `dynamic_cast`, `c-style cast`). I was provided code snippets to test my understanding of edge cases. Can't remember the snippets -they were rapid-fire. One was effectively: "How can we reassign a `const` variable?" I used `const_cast` and mentioned that the behavior is technically undefined, although it did compile and work correctly.
- The `volatile` keyword.
- Why pass in `const string&` instead of `string&` or `string`?
- A lot of these multiple-choice type questions you can grind through on getcracked.

When can templates be slower than runtime polymorphism in C++?

Templates increase binary / code size. If you go too crazy with templates it's possible to ruin the instruction cache by having to load in different paths of execution. A "softer" point: templated code can be harder to implement and debug, so from a team perspective it can slow down iteration if done poorly or by inexperienced engineers.

Even without prefetching, processing contiguous sequences of memory may be faster. Why?

This was part of a series of questions about **advantages of Parquet vs. CSV** (deep-dive, open discussion). When we read from memory, we're not reading a single element or bit - we fill up a cache line entry.

Resume-driven follow-ups

Likely asked because of points on my resume:

- Tell me, in as much detail as possible, what's happening when we're performing atomic operations in C++ (e.g. using `std::atomic`).
- Profiling methods: different types (instrumentation, sampling, HW counters, etc.), pros and cons, what software I used.
- Favorite feature of C++ beyond C++14? Biggest changes in C++11?
- Difference between lambdas and `std::function`. What is **type erasure**?

Other one-offs

- **(Some startup, 2019):** Do you know what the **copy-and-swap idiom** is? Plus follow-up questions.
- **(Citsec, 2019):** How could you measure cold read/write latencies for RAM? Interactive question -the answer was effectively: try to negate most of the optimizations your OS or compiler may do, e.g.:
  - Disable CPU caching and prefetching.
  - Isolate the CPU core.
  - Control DRAM refresh and other interferences.
- **(Citsec, 2019) Open discussion:** Tell me about optimizations modern processors and compilers do that may result in unintuitive results or behavior (open-ended, follow-up format).
  - **Processor:** prefetching, branch prediction, out-of-order execution, speculative execution.
  - **Compiler:** loop fusion, vectorization, inline expansion, dead code elimination, constant folding and propagation.

Coding Questions

- **(Multiple places, 2024):** Implement logic to continuously read from a fixed-size buffer (HEADER/DATA format), parse the data, and write it into another buffer.
- **(Citsec, 2019):** Implement a simplified `malloc`.

Extras

- Describe how `shared_ptr` is implemented. Is there any possibility of memory leak when using `shared_ptr`? Which functions of `shared_ptr` are thread-safe and which are not?
- Explain the compilation process.

- Idioms: what is **CRTP**, what are **mixins**, **RAII** (implement a simplified `lock_guard`), **PIMPL**.
- Examples of things that wreck performance: cache thrashing, interrupt storms, lock contention, memory fragmentation.
- Tell me about the common containers in the C++ STL, what data structures / algorithms they use under the hood, whether they support random / bidirectional / etc. iteration, and how iterators behave (when are they invalidated on erase / insert).
- What are **open addressing** hash tables?
- **Byte alignment** -what happens here?

```
struct A {
    char c;
    char d;
    int  i;
};
struct B {
    char c;
    int  i;
    char d;
};

std::cout << sizeof(A) << std::endl;
std::cout << sizeof(B) << std::endl;
```

- Tell me about loading, linking, and shared objects in the context of C++ programs.
  - Static vs. dynamic linking.
  - Does dynamic linking only incur a cost at program startup, or is there a penalty throughout the program's execution?
- **TCP vs. UDP**
  - Relatively basic here, but interviewer wanted me to go deeper.
  - Asked about a bit more complicated concepts like the sliding window and SACKs, explained via the 'SACK it' problem contained in this quiz.