Coding
(1)
https://leetcode.com/discuss/interview-question/5461047/Robinhood-Phonescreen-or-L4-or-July-2024

https://leetcode.com/discuss/interview-question/1052406/robinhood-telephonic-interviewreject

https://www.1point3acres.com/bbs/thread-812843-1-1.html

https://www.1point3acres.com/bbs/thread-793893-1-1.html

https://www.1point3acres.com/bbs/thread-763813-2-1.html

https://www.1point3acres.com/bbs/thread-577581-1-1.html

https://www.1point3acres.com/bbs/thread-680114-1-1.html

https://www.1point3acres.com/bbs/thread-566534-1-1.html
https://www.1point3acres.com/bbs/thread-560154-1-1.html

https://www.1point3acres.com/bbs/thread-787966-1-1.html

(4) Coding - Referal Count
https://www.1point3acres.com/bbs/thread-704860-1-1.html

https://leetcode.com/discuss/interview-experience/1702776/Robinhood-or-L2-or-New-York-or-Dec-2021-Reject

(5) 1656 https://www.1point3acres.com/bbs/thread-977087-1-1.html 不太懂题目意思。

(6) Margin Call 第二问有个case过不去 debug太久了 以至于没啥时间做第三问

https://www.1point3acres.com/bbs/thread-975068-1-1.html
https://www.1point3acres.com/bbs/thread-963155-1-1.html
https://www.1point3acres.com/bbs/thread-799337-1-1.html
https://www.1point3acres.com/bbs/thread-793893-1-1.html

(7) https://www.1point3acres.com/bbs/thread-901876-1-1.html

(8) LC1249

(9) 1801
　最后要求的是总共交易的total share。用两个heap或者priorityqueue就能解决。算是所有onsite的面经题中最简单、代码最少的一题了。很快就写完了，剩的时间跟面试官聊了半天有的没的。

(10）House & Street Trades Match
https://leetcode.com/playground/QhdXFrFc

https://www.1point3acres.com/bbs/thread-763813-1-1.html

https://leetcode.com/discuss/interview-question/882324/robinhood-phone-screen

（11）438
（12）295
（13）480, too hard

system design

一个是compute ads campaign ROI, 一个是job scheduler, 还有一个是trading system to submit limit order to external service.

job scheduler，stock exchange system，realtime stock price

今天hr也跟我说了一个distributed storage system, 请问楼主最后考的是job scheduler吗！！

(1) 加面: limit order sd。

1. support placing limited orders: during market hours, expire at each trading day
2. support accounting logic: to ensure the order is valid and consistent
3. support multiple/concurrent orders

https://www.1point3acres.com/bbs/thread-823704-1-1.html

Foundation：问past project，具体问了trade off，cross team collaboration，mentorship

(2) SD - Trading Broker
老题，设计一个能够查看实时股票价格和历史价格的api。后端链接两个纳斯达克tcp link获取数据，我被问了需不需要dedupe，failover的具体操作细节，问得挺详细的。

System Design: Design stock exchange system for limit order，重点是data model和fault tolerance

https://www.1point3acres.com/bbs/thread-963155-1-1.html

一个robinhood的交易系统。需要提供两个API, liveQuery and historical query。另一端和3rd party provider联系，接受股票的实时Update，主要是write operation

https://www.1point3acres.com/bbs/thread-943606-1-1.html

https://www.1point3acres.com/bbs/thread-884355-1-1.html

https://www.1point3acres.com/bbs/thread-856955-1-1.html

https://www.1point3acres.com/bbs/thread-852187-1-1.html

stockSymbol/Live -> return the live stock price by given stock symbal
/stockSymbol/Historical?range {1d,1m, 6m, 1y} -> return the historical stock price data by range.

https://www.1point3acres.com/bbs/thread-736807-1-1.html

https://www.1point3acres.com/bbs/thread-727142-1-1.html

* end-to-end设计绿林好汉。重点问了order的数据库设计，还有order从自家服务器到交易所的过程。

https://www.1point3acres.com/bbs/thread-577581-1-1.html

只handle服务器和NYSE之间的buy order。

需要写下database table的schema。例如：
| order id | symbol | quantity | timestamp |
| user id | balance | 等
（只是举略下，还有一些其他的column）

有4种failure cases：
1. error happens at server side
2. error happens when sending request from server to NYSE
3. error happens at NYSE side
4. error happens when receiving response from NYSE to server
How to identify each failure case?

compute ads campaign ROI

(1) https://www.1point3acres.com/bbs/thread-856955-1-1.html
(2) https://www.1point3acres.com/bbs/thread-758253-1-1.html
(3)

Behavioral:
A project that you're proud of.
What parts succeeded?
What parts failed?
What was the feedback from your coworkers?
Describe a bottleneck in the design and how you fixed it.
Describe how you worked across teams and across companies.
What would you do differently if you could do it over again?

On our journey to democratize finance for all, we’ve created the concept of fractional shares. Fractional shares are pieces, or fractions, of whole shares of a company or ETF.

However, exchanges only trade in whole shares. Robinhood is required to manage the fractional portion of each trade.

If Robinhood has 0 shares of AAPL and then a customer wishes to purchase 1.5 AAPL shares, Robinhood will need to request 2 shares from the exchange and hold on to the remaining 0.5 shares.
If another customer requests to purchase 0.4 shares of AAPL, Robinhood can use its inventory (0.5 shares) instead of going out to the exchange and will have 0.1 shares of AAPL remaining.
If the third customer requests 0.5 shares, Robinhood can fill 0.1 shares out of inventory but will need to go to the exchange for an additional share leaving Robinhood's inventory at 0.6 shares.
If a customer requests a dollar based order, we need to convert it to the relevant number of shares and run through the above steps.
Always ensure the firm has a positive quantity in inventory and has under one share after handling an order. There's no need for us to hold onto whole shares!
Steps:

Handle buying fractional shares.
Handle selling fractional shares.
Ensure inventory is less than 1 after each order.
e.g. Customer sells AAPL for 0.75 and then another sells AAPL for 0.50 -- we have 1.25 inventory. We can sell 1 share to the market and keep our inventory small at 0.25.
Ensure inventory is always non-negative after each order.
e.g. Inventory is 0.2 and the customer buys 0.5 shares: ensure we end up with 0.7 shares in inventory.
Always “flatten”! (steps 3+4)
The final 2 digits of every integer is the decimal. e.g. 1000 = 10.00, 20 = 0.20, 100 = 1.

Example scenario:

Input:
// One AAPL buy order for 0.42 shares. AAPL is currently worth $1.
orders: [["AAPL","B","42","100"]]

// Inventory for AAPL is currently 0.99 shares.
inventory: [["AAPL","99"]]

Expected Output:
// The users buys 0.42 shares from inventory, leaving us with 0.57 shares.
[["AAPL","57"]]
Another example scenario:

Input:
// One AAPL buy order for $0.42. AAPL is currently worth $1, so that's 0.42 shares.
orders: [["AAPL","B","$42","100"]]
// Existing AAPL inventory is 0.50 shares.
inventory: [["AAPL","50"]]

Expected Output:
// 0.50 - 0.42 = 0.08 shares leftover.
[["AAPL","8"]]

[execution time limit] 3 seconds (java)

[memory limit] 1 GB

[input] array.array.string orders

A list of orders in the format of [$SYMBOL, $BUY_OR_SELL, $QUANTITY, $CURRENT_PRICE]. Each parameter is a string.

$SYMBOL: Can be "AAPL", "GOOGL", "MEOOOOOW" or anything really.
$BUY_OR_SELL: "B" or "S". B for BUY, S for SELL.
$QUANTITY: Can be a number or a dollar amount (prefixed with $). e.g. "100" for 1 quantity or "$150" for $1.50.
$CURRENT_PRICE: Current price of the symbol with no $ sign. e.g. "1000" for $10.

** All numbers are multiplied by 100 to store two significant digits. e.g. 100 = 1.00, 150 = 1.50, 1025 = 10.25 **

[input] array.array.string inventory

Inventory is a list of the inventory of each symbol. Each element in the list a 2 item list of [$SYMBOL, $QUANTITY] (remember quantity is multiplied by 100!).

An example for AAPL of 0.50 shares and GOOGL of 0.75 shares would be:

[["AAPL","50"],
 ["GOOG","75"]]
[output] array.array.string

The output is the final inventory of each symbol after iterating through each trade. This is expected to be in the same order and format as the inventory parameter.

e.g.

["AAPL","58"],
 ["GOOG","50"]]