[redacted]

Order Book Assignment

In this assignment you are provided with three files:

- a .ipynb notebook that has incomplete code with 11 places (look for the word "Problem") where you are asked to fill in the details,
- a file "initial_customer_data" for 100 customers giving their cash and number of shares of a specific company they hold before the start of the trading day
- a file "requests.csv" contains a series of order book requests by these customers buying and selling shares of a specific company at times during a trading day.

The "requests.csv" file is comma-delimited and consists of the following 7 fields:

timestamp: hour:minute:second:millisecond
customerid: an 8 character string
activate/deactive: determines if new contract or contract cancellation
ask/bid: determines whether the contract is a bid or an ask
contractid: unique 10 character id assigned to a contact when the order is made
price: trading price requested by customer
quantity: number of shares to purchase/sell

The "initial_customer_data.csv" file is comma-delimited and consists of the following 4 fields:

cid:customerid: an 8 character string
special_status: a special status indicator (boolean)
nshares: number of shares held before trading day starts
cash: cash held before trading states

A bid order is interpreted as meaning that a customer (the buyer/bidder) is declaring interest in purchasing a specified quantity of shares and is willing to pay a specified price per share for those shares. An ask order is interpreted as meaning that a customer (the asker/seller) is declaring interest in selling a specified quantity of shares and is willing to pay accept a specified price per share for those shares.

Assume that at the moment before the the orders begin to arrive

- the order book has no current contracts
- the customers cash holdings and shares are given as in the "initial_customer_data.csv" file.

All customers are not created equally. Certain customers pay for the right to special status, giving them special privileges. A customer's status is determined by a `special_status` indicator, which can be True or False. This affects the priority of their orders as described below. The customer's special status is also given in the "initial_customer_data.csv" file.

As the orders arrive they need to be processed. Bid orders are ultimately placed in a bid queue and ask orders are placed in an ask queue.

- When aded to the queue the associated contracts become "active".
- A bidder or asker might decide to *deactivate* a existing active contract before trading on it takes place, or after a portion of it has traded. This bid or ask should no longer lead to a trade taking place.
- Top priority [1] is given to the bidder willing to pay the highest price and the asker willing to accept the lowest price. Things get complicated when the two bidders offer the same price or two askers ask for the same price. So in case of ties a decision has to be made to determine which bid or ask gets highest priority.
- Priority of a bid in the case of a tie in the offering price is determined as follows:
  - if the two bidders have the same status then whichever one submitted their order first gets the higher priority.
  - if one bidder has special status and the other does not, then the one with special status is given the higher priority.
- Priority of an ask in the case of a tie in the asking price is determined as follows:
  - if the two askers have the same special status, then whichever one submitted their order first gets the higher priority.
  - if one asker has special status and the other does not, then the one with special status is given the higher priority.
- When the top priority bidder's price is at least as high as the and top priority asker's price, a trade will take place.
- The price they trade at is determined as follows:
  - if the bidder and the asker have the same status then the price is determined by whichever order was placed earliest i.e. whichever order has the earliest timestamp.

[1] If bid #1 has higher priority than bid #1, that means that bid #1 should be processed before bid #2, and similarly for an ask. Note that higher priority means *lower— in the ordering we use for the priority queue*

  - if the bidder has special status and the asker does not, the price they trade at is the asker's (lower) price.
  - if the asker has special status and the bidder does not, then the price they trade at is the bidder's (higher) price.

- Once it is determined which bid and ask contracts get highest priority and a trade should take place, the number of shares traded is the minimum of the two quantities specified in their respective contracts.
- If the buyer purchases fewer shares than specified in their contract, after the trade, the quantity in their bid order is modified to account for the fact that part of their bid order has been met. So despite the trade, their contract might remain active.
- Similarly, if the seller sells fewer shares than specified in their contract, after the trade, the quantity in their ask order is modified to account for the fact that part of their ask order has been met, and despite the trade, their contract can remain active.
- When the quantity specified in a contract is reached by a bidder or asker, their contract becomes inactive.

The Jupyter notebook `ProcessRequests.ipynb` file contains code that should perform the above tasks once you fill in the missing details. You will see that various classes thar are pre-defined (request, bid, ask, timeofday, customer, etc.) that you should not need to modify unless told you need to. You should supply the missing code wherever you see `Problem` (there are 11 "Problems") in the notebook file that, once completed, running the file should accomplish the above tasks.

When the program completes running, assuming the problems have been solved as required, it should output to a file `final_customer_data.csv` giving the final customer balances.

Here are some more details about how the code should work.

A new "activate" order is processed as follows:

- it is put into (`bid_vector` or `ask_vector`) that stores all bids or asks,
- the bid/ask is made "active" (each bid/ask has a data structure with a boolean field called `active` - this is made true
- the `bid_dictionary` is available to look up a bid from its contract id, and the `ask_dictionary` is available to look up an ask from its contract id, of a contract in `bid_vector` or `ask_vector` is updated.
- the dictionaries can be used to check the active/inactive status of a bid/ask,
- the customer who places the order has their vector of active bids `active_bidids` or active asks `active_askids` updated to reflect the new active contract

- the contract is placed in the appropriate priority queue (`bid_queue`/`ask_queue`)

A a deactivate order is processed as follows

- the bid/ask in the `bid_vector` or `ask_vector` is made inactive by changing the `active` field to false
- the customer who places the deactivate order has the corresponding contract removed from their vector of active bids `active_bidids` or active asks `active_askids`
- when an order is deactivated, any quantity and price information passed in the order is ignored when the order is processed.

Each time a new order is processed as described above, the `uptate_queue()` function is called. This function needs to do the following in a `while` loop

- remove top inactive bids from the bid queue until an active bid is at the top or the queue is empty
- remove top inactive asks from the ask queue until we active ask is at the top or the queue is empty
- if either queue is now empty we're done processing until a new order arrives, so we return to the calling function
- otherwise, get the top priority bid and top priority ask and process them as follows
  - determine whether a trade should take place by comparing bid price and ask price and if there is no trade, return to the calling function
  - determine the price at which trade should occur (see below) and quantity traded
  - look up the customers associated with the contracts and update their cash and number of shares holdings
  - modify the bid and ask contract quantities to reflect that some shares have traded
  - if a contract has quantities that are zero, change the "active" status of the contract, adjust the customer's active contract list, and remove the contract from the queue

Suggestion: There are various bits of "DEBUGING" code that have been provided to check that the queue is updated correctly. It allows for tracking trades, the order book, and customer data.

Scoring: For full credit you should get the correct customer balances after the trading day. If incorrect answers are given you may recive partial credit for writing some of the 11 missing codes correctly.

Submission: You should submit (there are two parts to the assignment):

Part 1) Your completed Jupyter notebook `ProcessRequests.ipynb` file with working code. This code should run efficiently and without errors on the entire dataset - my code takes under 30 seconds to run on the whole dataset). **The code you provide should all be in a single cell.**

Part 2) You should submit the output file final `final_customer_data.csv`. **Important:** The functions at the end of the notebook that print out the final customer data not be modified in any way. Any formatting changes in the output file will be penalized.