BFA3 - Algo Trading - Problem 1

October 2025

The goal of this challenge is to write a program performing an equity trading auction i.e. determining the cross price given a set of buyers and sellers.

1 Input file

The program shall read the list of bids from an ASCII text file written using the following convention:

- One order per line;
- No empty or ill-formed line;
- Fields are comma separated;
- Each line contains the following field in order:
  - Timestamps in nanoseconds since Jan 1st, 1970;
  - Symbol name (for example AAPL if the auction is for Apple shares), symbols are uppercase letters only;
  - Side of the order i.e. B for a buy order and S for a sell order;
  - Number of shares to bid (always an unsigned integer);
  - Price of the bid (a floating point number) with zero meaning market price i.e. buy at infinity or sell at zero.

2 Expected results

The program shall produce the following data:

- Price of the auction (if any);
- Volume crossed (unsigned integer)
- Remaining imbalance (signed integer, either B for a buy surplus or S for a sell surplus or N if no imbalance)

You are expected to submit this program by email. It shall have a readme file with relevant information, including how to execute your program.

3 Auction algorithm

1. Any buy orders can potentially be matched with any sell order provided their respective prices are compatible i.e. the buy price is greater than or equal to the sell price.
2. The auction price (or cross price) must be one of the order prices (notwithstanding market orders).
3. Considering all the buy and sell orders, find the price at which the maximum number of shares are matched.
4. For any potential cross price, eligible orders are the orders compatible with said cross price i.e. all sell orders cheaper than the potential cross price and all buy orders dearer than the potential cross price are eligible. All other orders are deemed non eligible.
5. If there is more than one price, choose the one which minimizes the imbalance i.e. which leaves the minimum number of shares from all eligible orders unfilled. Note that this quantity is signed (either B or S, or ideally N for nil).
6. If there still are multiple possible prices chose the one closer to the reference price (which is provided before the algorithm is run).
7. Finally if there still are multiple prices a tie breaker is applied: The lowest price is used if the oldest eligible order is a buy and the highest price is used if the oldest eligible order is a sell.

4 Input example

File:
1527604196773077003,AAPL,S,500,270.5700
1527604199695788161,AAPL,B,100,270.3900

1527604199397997988,AAPL,S,100,0
1527604199974781594,AAPL,S,900,278.00
1527604200211637272,AAPL,B,100,0

Reference price: 275.99

5 python code

Create a folder problem1.
Create a file auction.py and a main function def compute_auction.
You must implement in auction.py a function:

def compute_auction(orders, reference_price) -¿ dict

orders: list of dicts, each with keys:
"ts", "symbol", "side", "qty", "px"
All orders belong to the same symbol.

The function must return a dict:

"symbol": str,
"cross_price": float or None,
"crossed_volume": int,
"imbalance_side": "B" — "S" — "N",
"imbalance_qty": int

6 Test and validation

After implementing your function compute_auction in the script auction.py, copy the file test_auction in your folder problem1.
To run the test and validate your code, run the following line in the command prompt (having your current folder: problem1): **pytest -v** (you may need to install the python library pytest)

The scenarii tested are:

1. **Basic single-asset example (AAPL)** Reproduces exactly the example provided in the Challenge PDF. Verifies correct eligibility logic, matched volume computation, and imbalance. *Expected:* price = 270.39, volume = 100, imbalance = B 100.

2. **Multiple symbols handled independently (AAPL + MSFT)** Ensures that the auction runs independently per symbol, with distinct reference prices. One symbol crosses perfectly (AAPL), the other has no crossing (MSFT).

3. **Only market orders (no limit prices)** All orders have a price of 0. Checks that when there are no valid (non-zero) candidate prices, the algorithm returns *no cross* (`cross_price = None`, `volume = 0`, `imbalance = N 0`).

4. **Non-overlapping prices (zero volume for all candidates)** Highest buy price $<$ lowest sell price. Confirms that the algorithm correctly reports no auction.

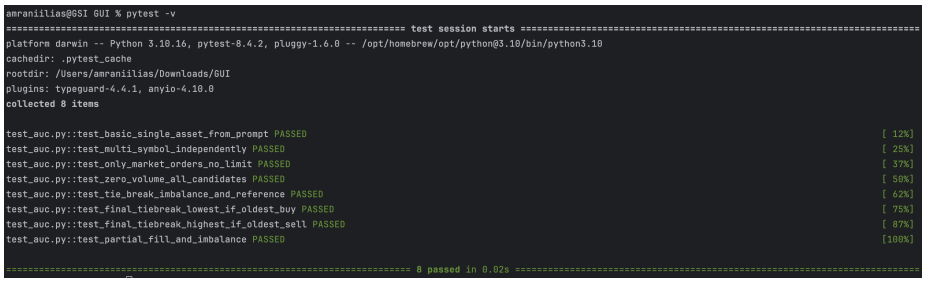
5. **Tie-break on imbalance, then reference price** Several candidate prices yield the same maximum matched volume. Verifies that the algorithm selects first the price with the smallest absolute imbalance, and if still tied, the one closest to the reference price.

6. **Final tie-break: oldest order is a BUY (choose lowest price)** When all previous criteria tie, the oldest eligible order is a buy. Confirms that the algorithm chooses the lowest price among the tied candidates.

7. **Final tie-break: oldest order is a SELL (choose highest price)** Same situation as above but with a sell order being the oldest eligible one. Checks that the algorithm chooses the highest price among the tied candidates.

8. **Partial fill and imbalance sign check** Tests a situation where one side has a larger total quantity (e.g. B 80 @10 vs S 50 @9). Confirms that only min(buy, sell) is crossed, that the imbalance sign is correct (B 30), and that the chosen price is the one closest to the reference value.

If your code is correct you should see something like:



7 Hints

The article **Autcion_article.pdf** may help finding the correct matching idea.

Pytest is an industry standard for unit testing and ensuring code robustness. In our case, it allows us to validate the implementation against the different scenarios and edge cases. for more info please visit: Pytest