COMP0005 Algorithms
Academic Year 2021-22

Group Coursework: Stock Trading

Released: 11 February 2022
Submission deadline: 16 March 2022 @ 4pm GMT

Background

A stock trading website specialises in the buying/selling of stocks from the following twelve FTSE-100 companies operating in the financial sector: Barclays, HSBA, Lloyds Banking Group, NatWest Group, Standard Chartered, 3i, Abrdn, Hargreaves Lansdown, London Stock Exchange Group, Pershing Square Holdings, Schroders, and St. James's Place plc. The website processes hundreds of thousands of requests to buy / sell stocks from these companies every day. Every time a request to buy/sell a stock is successfully completed, a new transaction record is created and logged, containing the following 4 pieces of information:

Stock name (a string)
Price per stock in GBP (a float)
Quantity (a positive integer)
Timestamp (formatted as day/month/year HH:MM:SS)

For example, the following record:

Stock name = Barclays
Price per stock in GBP = 10.80
Quantity = 1000
11/02/2022 16:00:00

represents a transaction to trade 1000 stocks of Barclays, priced at 10.80GBP each; the transaction was successfully executed and logged on February 11th, 2022 at 4pm.

The task.

Your task is to design and implement data structures and algorithms to represent all the information contained in successfully completed transaction records, and to efficiently support the following operations on them:

- `logTransaction(record)`: adds a newly completed transaction `record` to those logged so far.
- `sortedTransactions(stockName)`: returns all transactions of a given `stockName` completed so far, sorted by increasing trade value. The trade value of a transaction is defined as the price per stock multiplied by the quantity of stocks traded in such transaction (e.g., the trade value in the example above would be 10,800.00 GBP; the same value would be associated to the trade of 900 Barclays' stocks at 12.00 GBP each).
- `minTransactions(stockName)`: returns the transaction(s) of a given `stockName` with minimum trade value seen so far.
- `maxTransactions(stockName)`: returns the transaction(s) of a given `stockName` with maximum trade value seen so far.

- `floorTransactions(stockName, thresholdValue)`: returns the transaction(s) of a given `stockName` with largest trade value seen so far below a given `thresholdValue`.
- `ceilingTransactions(stockName, thresholdValue)`: returns the transaction(s) of a given `stockName` with smallest trade value seen so far above a given `thresholdValue`.
- `rangeTransactions(stockName, fromValue, roValue)`: returns the transactions of a given `stockName` seen so far whose trade value is within the range `[fromValue, toValue]`.

New transactions are continuously logged; you may assume that transactions, once recorded, are never deleted.

Several data structures and algorithms have been studied that could be used to implement the above API, but with different performance guarantees. Your task is to design, implement and evaluate a solution that is suitable for the above application scenario. Evaluation should be conducted both *theoretically* (in terms of space and time complexity), and *experimentally*. For experimentation, you are expected to design and implement your own experimental framework, with the aim to measure the performance of the various API operations under different conditions (for you to determine).

Constraints. For this coursework, you are expected to implement your own algorithms and data structures. You are NOT allowed to use (import) python libraries, with the exception of `timeit` and `random` (that you may use as part of your experimental framework). If in doubt about what you can and cannot use, ask in Moodle "Ask a question" forum. Your code must successfully run using **Jupyter Notebook with Python 3.8**.

Submission instructions.

This is a group-based coursework and only one submission per group is required. Using the Moodle course website, submit exactly two files:

- A Jupyter notebook `StockTrading.ipynb` containing your implementation of all data structures and algorithms required for this coursework (including your experimental framework). This notebook MUST follow the structure of the skeleton code provided. Do not submit the synthetic data you created for testing purposes; rather, your notebook should contain code to generate it.

- A PDF document `report.pdf` of maximum 4 pages (font style: Ariel or Times New Roman, font size: 12pt), where you describe:

  - your choice of data structures and algorithms to realise the above API;
  - theoretical analysis of your proposed solution (cover both space and time complexity, both in the average and worst-case scenarios);
  - experimental analysis: first, briefly describe the way your experimentation framework operates, then critically and thoroughly discuss your results. Include graphs to illustrate these results (for example, consider using scatter plots, with the number of logged

transactions on the `x`-axis and the measured execution time of your operations on the `y`-axis). Conclude the report with a summary of the main pros and cons of your solution and highlight what scenario(s) it is best suited for.

You must use the skeleton Jupyter Notebook code provided. Your solution must successfully run using Python 3.8.

Submission deadline: Wednesday, March 16th 2022, 4pm GMT.

Assessment.

This coursework represents 40% of the final mark for COMP0005 Algorithms. Submissions will be evaluated in terms of the following marking criteria:

- Quality of your code (i.e., is your implementation *correct*? Is your code *readable* and *well documented*? Is it time and space *efficient*?) [40 points]
- Theoretical analysis (e.g., is your theoretical analysis *correct*? Is it *complete*?) [20 points]
- Experimental analysis (e.g., is your experimental framework comprehensive? Are the results of your experimentation correct? Are they thoroughly discussed?) [40 points]

Marks for each criterion will be assigned following the UCL CS Grade Descriptor (criteria 4, 5 and 6).

Note: students within the same group should not expect to receive the same mark. Marks will also depend on each student's individual contribution to the work of the group (determined based on active engagement during the weekly TA lab sessions).

Academic Integrity.

UCL has a very clear position about academic integrity – what it is, why it is important, and what happens if you breach it. Make sure you have read UCL position on this matter before attempting this coursework.