Background

From the following user story:

- As Steven, a crypto trader, I want transactions in the mempool to be prioritized by fee, so that I can pay more fees to ensure my transactions get included in a block quickly, and less fees when the transaction is not urgent.

Note: A *mempool* is a group of unconfirmed transactions that are waiting to be included in a block. For an introduction to the concept as it relates to Ethereum: see https://www.blocknative.com/blog/mempool-intro

Technical details

For this exercise, you are tasked with implementing a priority mempool where transactions entering the mempool are ordered based on the fee that they pay. You can use any programming language you like. Note that the mempool should have a maximum size of 5000 transactions. Past 5000 transactions, inserting a new transaction into the mempool should cause the lowest priority transaction to be dropped.

After you have implemented the mempool, you will need to read the included *transactions.txt* file, parse the transactions, and insert them into the mempool. Then, create an output file, *prioritized-transactions.txt*, that lists the transactions as they are ordered in the mempool.

The transactions in *transactions.txt* are formatted as follows, with one transaction per line.

TxHash=0x54030E30503453949230403 Gas=300000 FeePerGas=0.001 Signature=0x54030E30503453949230403

Note that `FeePerGas` refers to the amount of fees that will be paid *per unit* of gas (0.001 * 300000) in the example above.

Your solution file, *prioritized-transactions.txt,* should have the same format as above

To download transactions.txt:
https://gist.github.com/karzak/9d4c25e2c92fc64a98ddf28d833308a1/raw/1b5ca87a14ace69b0de4f801983108e868d4f140/transactions.txt

Considerations

This exercise is designed to take 2-4 hours to complete.
Please host this code on Github, Gitlab, or Bitbucket AND paste the link into the submission link in the email you received for this assignment.

In evaluation, we will be looking for:

1. Correctness of solution
2. Soundness of approach - how efficient is the mempool implementation, have edge cases been properly handled
3. Code readability / testability / maintainability
4. Automated Tests