Gleb Chuvpilo
Knight Capital
Order Book Problem
November 15, 2012

Running the solution using a market data file:

```
$ make clean && make && make run
```

Running the solution using standard input (output in bold):

```
$./Pricer 200
28800538 A b S 44.26 100
28800562 A c B 44.10 100
28800744 R b 100
28800758 A d B 44.18 157
28800758 S 8832.56
28800773 A e S 44.38 100
28800796 R d 157
28800796 S NA
28800812 A f B 44.18 157
28800812 S 8832.56
28800974 A g S 44.27 100
28800974 B 8865.00
28800975 R e 100
28800975 B NA
28812071 R f 100
28812071 S NA
28813129 A h B 43.68 50
```

```
28813129 S 8806.50
28813300 R f 57
28813300 S NA
28813830 A i S 44.18 100
28813830 B 8845.00
28814087 A j S 44.18 1000
28814087 B 8836.00
28814834 R c 100
28814864 A k B 44.09 100
28815774 R k 100
28815804 A l B 44.07 175
28815804 S 8804.25
28815937 R j 1000
28815937 B 8845.00
28816245 A m S 44.22 100
28816245 B 8840.00
```

Files/line count:

```
$wc -l *.h *.cpp
      89 Exceptions.h
     186 Log.h
      75 MarketDataProvider.h
      58 MarketOrder.h
     167 OrderBook.h
      87 Parser.h
      42 Utils.h
     168 Main.cpp
      30 MarketDataProvider.cpp
     109 OrderBook.cpp
      41 Utils.cpp
    1052 total
```

What is the time complexity for processing an Add Order message?

I am using an STL multimap to store orders using price as the key. STL multimap is implemented as a binary search tree. Therefore, Add Order messages are processed O(log n).

What is the time complexity for processing a Reduce Order message?

I am using an additional data structure – STL unordered_map – to go from an order ID to an iterator into the multimap above. The STL unordered_map is implemented as a hash map, therefore lookup is O(1). I pass this iterator to the multimap's erase method, which runs O(1). Thus, total time complexity of Reduce Order is O(1).

If your implementation were put into production and found to be too slow, what ideas would you try out to improve its performance?

- Increase the size of the hash
- Improve the hashing function
- Use a custom multimap implementation
- Use a cache-aware algorithm
- Inline more methods (most are inlined already, though)
- Use better message parsing
- Use an GPU or an FPGA (preferable)
- Co-locate
- Use a faster/fatter optical link to the exchange
- Use better network cards and routers
- Timestamp packets on arrival and on departure and figure out the bottleneck
- Use valgrind for memory optimization