**Implementation:**

As was requested - implemented without any additional libraries.

The main business logic of solution covered with unit test(JUnit).

The executable jar built up with Maven.

The **jar** can be loaded from:

<https://github.com/lakoza/CIBOrderBook/blob/master/target/CIBOrderBook-1.0.jar>

(click View Raw and jar will be saved to yours downloads folder)

The **source code** can be checked here:

<https://github.com/lakoza/CIBOrderBook/tree/master/src>

In case if you don’t want to use git – archive with jar and src attached to email.

**Run solution:**

**java –jar ${path\_to\_the\_jar\_file}/CIBOrderBook-1.0.jar ${path\_to\_xml}**

You have to provide path to the file(like C:\tmp\orders.xml).

Path added as argument because of flexibility and not to include huge test file in solution as a resource.

**Structure:**

Used standard TreeSet collection for each bid and ask orders. TreeSet supports sorted order based on custom Comparators and provides in average the best time(comparing with other standard data structures) to required operations.

Additional structure HashMap – stores orders for future effective deletion orders from TreeSet.

**Time complexity:**

Add order: O(log n)

Delete order: O(log n)

Access to first order: O(1) (but there is no exact information about implementation, could be done by O(Log n))

Access order: O(log n)

**Order book merging algorithm:**

Used the simplest strategy – **FIFO** – ordering by Price and Time priority. the first order at a price level is the first order matched. Orders with the same price matches by time priority. By time I assume the orderId sequence( file parsing sequence accordingly).

**Speed up solution:**

* Use SAX parser instead of StaX(which is used in my solution) parser for XML processing as SAX performs better according some benchmarks.
* Implement custom balanced search tree – delete could be done by O(1).
* If HashMap for deleting objects would not perform well(worst case could be O(n)) -improve hash function.
* If solution supposed to be multithreading – think more carefully about acquiring the locks(synchronized methods),
* Make the smallest parts of code synchronized.
* If there supposed to be a much output information(like frequent order book printing) – should get rid of System.out and use logging instruments which are perform faster.

**Improve solution:**

* Good to have the order book **depth** parameter in case when there is a lot orders in a book and we are not interested in whole orders in book.
* Extend Order to have Time parameter as in real life. Add to merging strategy Time parameter.
* Extend solution to have more merging algorithms for Order Book processing, e.g. implement Pro-Rata strategy.

**Simplify solution:**

As solution is pretty simple itself I don’t see any ways to simplify it.