An **Order book** is used by an exchange to maintain and match a list of orders from buyers and sellers. Order books for different instruments are separate.

Each order book consists of two sides – bid and ask. The bid side is the set of orders from buyers and the ask side is the set of orders from sellers.

Each side consists of price levels, where orders from each side of the book with the same price are grouped together.

The levels within the book side are sorted by price in such way that the top level for the bid side contains orders with the highest price, and the top level for the ask side contains orders with the lowest price.

Inside each level, orders are sorted in the order as they arrive.

Possible operations on the order book:

- Add new order. New order specifies instrument, side (either buy or sell), requested quantity and price. Additionally, each order is identified by a unique order id;
- Modify existing order. Only modification of an order's quantity is allowed. You need to specify the unique order id of an existing order and the new quantity. If the quantity is decreased, the order maintains its position in the queue of orders for the relevant price level. If the quantity is increased, the order is put at the end of the queue of orders for the relevant price level;
- Delete existing order. You need to specify the unique order id of an existing order. The order is removed from the order book completely.

Additionally, at any time, an order book should provide the following information:

- Best bid and best ask prices;
- Number of orders on a level and side of the book;
- Total tradeable quantity on a level and side of the book;
- Total tradeable volume, i.e. sum of price*quantity for all orders, on a level and side of the book;
- List of orders on a level and side of the book, in the correct order

Your task is to implement OrderBookManager, which maintains order books for different instruments. A skeleton OrderBookManagerImpl class is provided.

An order book should be kept in the described order at all times, i.e. the described order should be maintained after any new/modify/delete command, not only after receiving all orders.

For this test assignment, you don’t need to implement matching of orders, so, potentially, order book could be in situation when the best bid price is higher or equal to the best ask price.

Additionally, we expect you to implement as many unit tests as you find reasonable for this problem. A sample test is provided.

Your solution will be assessed for clarity of implementation, simplicity and readability of the code, proper usage of data structures and test coverage. At this moment, we don’t care much about low level optimization for CPU or memory usage, however we do care about the spatial and temporal complexity of the native data structures you choose and why you chose them.

P.S. We will use single core PC to run your tests.