Limit Order Book: Documentation

VELAN E — velane929@gmail.com

November 2025

What This Does

This implements a simplified limit order book used in trading engines. Buyers and sellers submit orders with a price or with no price in the case of market orders. The book matches compatible prices and produces trades.

Core Data Structures

Orders and Trades

Two containers capture intent and execution:

```
Order = (id, side, price, qty,
    timestamp)
Trade = (buy_id, sell_id, price, qty
    , timestamp)
```

Side is either buy or sell. A market order uses price None.

The Book

The `OrderBook` class maintains:

- `bids`: map from price to queue of buy orders
- `asks`: map from price to queue of sell orders
- `bids_prices`: descending prices, best first
- `asks_prices`: ascending prices, best first
- `trades`: executed trades

The dict gives constant time access to queues at a price. The sorted lists give direct access to the best level.

Matching Logic

Price Time Priority

Orders at the same price follow FIFO rules. Earlier arrivals match first.

Limit Orders

When a buy limit at price P_b arrives:

1. Check if sell orders exist at any price less than or equal to P_b
2. If they do, match against the lowest sell prices
3. Place leftover quantity at price P_b

Sell limits operate with the symmetric rule: match against bids with price at least P_s.

Market Orders

A market buy consumes sell orders starting from the lowest ask upward. A market sell consumes bids from highest downward. Any remaining quantity disappears once the visible book is consumed.

Implementation Details

Inserting Prices

To keep `bids_prices` sorted in descending order, the implementation inserts negative prices into a list that remains sorted in ascending order. This avoids reversing during lookups.

Insertion uses `bisect.bisect_left` in logarithmic time.

Matching Loop

Example for matching a buy limit at P_b:

```
while order.qty > 0 and asks_prices
   and asks_prices[0] <= P_b:
    best_price = asks_prices[0]
    q = asks[best_price]
    while q and order.qty > 0:
        top = q[0]
        trade_qty = min(order.qty,
            top.qty)
        # create trade and update
            quantities
        ...
```

If a queue at a price becomes empty, remove that price from the lists and maps.

Cleanup

After processing, if a price level holds no orders:

```
if not price_map[price]:
    del price_map[price]
    price_list.remove(price)
```

The book only stores active price levels which keeps memory efficient.

Benchmark Results

A simple benchmark submits random limit and market orders.

Typical performance on modern hardware:

- Ten thousand orders in roughly 0.1 seconds
- Around one hundred thousand orders per second throughput
- Average per order latency near ten microseconds

Real exchange engines reach far higher throughput by using advanced structures such as skip lists or custom heaps. This version keeps the design compact for clarity.

Why This Matters

Limit order books drive price discovery in financial markets. Gaining intuition for how matching works provides insight into:

- Microsecond sensitivity in HFT workflows
- Queue dynamics at each price level
- Tradeoffs between limit orders and market orders

Extensions

Possible directions:

1. Order cancellation inside a queue
2. Triggered stop orders
3. Iceberg orders that only show part of the size
4. Moving core loops to C++ for low latency