**Binance Orderbook Implementation Test**

Please submit your code along with instructions on how to run it.

**Objective**  
Implement a simple orderbook using Rust that processes Binance Individual Symbol Book Ticker and Partial Book Depth streams.

**Requirements**

Create a struct `OrderBook` that represents the orderbook.  
2. Implement methods to update the orderbook based on Book Ticker and Partial Book Depth data.  
3. Implement a method to get the current best bid and ask prices and quantities.  
4. Implement a method to get the total volume at a given price level.  
5. Use appropriate data structures for efficient updates and queries.

**Input Data Format**

Individual Symbol Book Ticker:  
```json  
{  
  "u":400900217,     // order book updateId  
  "s":"BNBUSDT",     // symbol  
  "b":"25.35190000", // best bid price  
  "B":"31.21000000", // best bid qty  
  "a":"25.36520000", // best ask price  
  "A":"40.66000000"  // best ask qty  
}  
```Partial Book Depth (20 levels):  
```json  
{  
  "lastUpdateId": 160,  // Last update ID  
  "bids": [             // Bids to be updated  
    [  
      "0.0024",         // Price level to be updated  
      "10"              // Quantity  
    ]  
  ],  
  "asks": [             // Asks to be updated  
    [  
      "0.0026",         // Price level to be updated  
      "100"             // Quantity  
    ]  
  ]  
}  
```

**Tasks**

1. Implement the `OrderBook` struct with necessary fields.
2. Implement the following methods for `OrderBook`:  
      - `new(symbol: String) -> OrderBook`: Create a new empty orderbook for the given symbol.  
      - `update\_book\_ticker(&mut self, data: &BookTickerUpdate)`: Update the orderbook with Book Ticker data.  
      - `update\_depth(&mut self, data: &DepthUpdate)`: Update the orderbook with Partial Book Depth data.  
      - `get\_best\_bid\_ask(&self) -> Option<((f64, f64), (f64, f64))>`: Return the best bid and ask prices and quantities.  
      - `get\_volume\_at\_price(&self, price: f64) -> f64`: Return the total volume at a given price level.
3. Write unit tests to verify the correctness of your implementation.

Bonus Points

- Implement error handling for invalid input data.  
- Optimize for performance, especially for frequent updates and queries.  
- Implement a simple WebSocket client to connect to Binance streams and update the orderbook in real-time.  
- Handle the case where the depth snapshot and updates need to be synchronized (using `lastUpdateId`).

Evaluation Criteria- Correctness of the implementation  
- Code organization and readability  
- Efficient use of data structures  
- Proper error handling  
- Test coverage  
- Understanding of WebSocket streams and order book concepts

**Additional Notes**

- For the Partial Book Depth stream, assume you're using the 20 levels version (<symbol>@depth20@100ms).  
- The Book Ticker stream provides real-time updates to the best bid and ask prices and quantities.  
- The Partial Book Depth stream provides updates to the order book levels, which may include new levels, updates to existing levels, or removal of levels.  
- Pay attention to the `lastUpdateId` in the Partial Book Depth updates for maintaining the correct order of updates.