Technical Solution/Design Document

Project Title: Unique Bid Blind Online Auction

Prepared By: Jasmine Fowler

---

1. Problem Statement

The task is to develop an online auction system where bidders submit integer bids (in Canadian dollars) for items. A bidder may place multiple bids. The highest *unique* bid wins the auction. If the highest bid is a tie, the auction either restarts or awards the win to the first bidder (based on a rule toggle).

---

2. Objectives

- Accept valid user bids in real time
- Enforce a minimum bid value
- Record multiple bids per bidder
- Detect and resolve ties or determine a unique winner
- Efficiently handle thousands of bids
- Log auction results to a file
- Provide an optional rule to resolve ties based on who bid first

---

3. Solution Overview

The solution consists of three Java classes:

- **Bid.java**: Represents a single bid by a bidder.
- **Auction.java**: Handles all auction logic including storing bids, determining winners, and handling ties.
- **Main.java**: Runs the program and accepts real-time bid input from the user.

---

4. Data Structures & Algorithms

Data Structure Used:

- HashMap<Integer, Queue<Integer>>

    - Key: Bid amount
    - Value: Queue of bidder IDs (preserves order of entry)

Why this structure?

- Fast lookup and insertions even with thousands of bids
- Queue keeps track of who placed a bid first (important for tie rules)
- Makes it simple to check for unique bids or ties

Algorithm Logic:

- On bid: validate and insert bidder ID into the appropriate queue
- On auction close:
    - Sort bid amounts in descending order
    - If highest bid has only one bidder: declare winner
    - If tie and rule = "first bidder wins": declare first in queue as winner
    - If tie and rule = "restart": restart auction with new minimum bid = (highest tie + 1)

---

5. Assignment Requirement Answers

[Part A - 5%] Time complexities of five algorithms are given: a) $O(n^3)$, b) $\Omega(n^3)$, c) $O(n^2 \log n)$, d) $\Theta(n^2 \log n)$, e) $O(2^n)$ Which one would you choose as the best for a relatively large n? Explain.

The best option for large inputs is **d) $\Theta(n^2 \log n)$**. This notation means the algorithm is guaranteed to run in that exact time range, not faster or slower; making it more predictable. It performs better than $O(n^3)$ and far better than exponential $O(2^n)$, especially as input size grows. Compared to $O(n^2 \log n)$, which only provides an upper limit, $\Theta(n^2 \log n)$ gives both the upper and lower bounds. It's efficient and consistent for scaling with large n.

[20%] How do you store and process bids efficiently?

To handle bids efficiently, especially when we're expecting large amounts of input, we use a HashMap where each key is the bid amount and the value is a Queue of bidder IDs who placed that bid. This setup allows constant-time access to any bid amount, and the queue helps us keep track of the order the bids came in. It works well even with thousands of bids and keeps things organized when checking for ties or picking a winner. Bids are processed one at a time through the placeBid() method, which keeps it clean and scalable.

[15%] What if the tie rule changes to "first bidder wins"?

If the rule changes so that the first person who placed the highest bid wins, we don't need to change the structure at all. Since we're already using a queue for each bid amount, the first person in the queue is the one who bid first. We just check who's at the front of the queue when a tie is detected. The logic is already built in to support this, we just flip a boolean to choose which rule to follow.

---

6. Tie-Breaker Rule Toggle

- The program supports a boolean tieGoesToFirstBidder
- If true: first person who placed the bid wins
- If false: auction restarts with updated minimum bid

---

7. Input and Output

Input:

- Bidders enter: <bidderId> <bidAmount>
- Special command: end to close bidding round

Output:

- Winner announced on screen
- Full auction log saved to auction_results.txt

---

8. Performance & Scalability

- Efficient due to hash-based storage and simple sorting
- Can handle thousands of bids easily
- Minimal memory overhead

---

9. Optional Features Implemented

- Logging results to a file
- Auto-restarting auctions
- Real-time console input for interactive experience

---

10. Technologies Used

- Java 18
- IntelliJ IDEA (macOS)
- Maven (for project setup and dependencies)

---

11. Conclusion

This solution provides a fully functional, scalable, and flexible system for managing a unique bid blind online auction, meeting all assignment requirements and supporting future enhancements.

---

12. GitHub Repository

https://github.com/Winter2024-NSCC-ECampus/final-project-online-auction-solution-minlcve/tree/main