Data Structures

&

Algorithms

Final Assignment

Mike LeBlanc

Table of Contents

[Part A: Complexity 3](#_Toc161470971)

[Part B: Data Structures 4](#_Toc161470972)

[Data Structure: 4](#_Toc161470973)

[Priority Queue: 4](#_Toc161470974)

[Algorithm: 4](#_Toc161470975)

[Hash Table: 4](#_Toc161470976)

[Binary Search: 4](#_Toc161470977)

[Processing Steps: 4](#_Toc161470978)

[Handling Large Numbers of Bids: 5](#_Toc161470979)

[Efficient Data Structures: 5](#_Toc161470980)

[Optimize Memory Usage: 5](#_Toc161470981)

[Handling Tie Breaker: 5](#_Toc161470982)

# Part A: Complexity

For relatively large values of n, the most efficient algorithm would be the one with the smallest time complexity. Let's review all the time complexities given:

A) O(n3) - This algorithm has cubic time complexity. It will become very slow as n increases.

B) Ω(n3) - This indicates a lower bound of n3 operations, meaning it's at least as bad as O(n3) in terms of performance.

C) O(n2log n) - This algorithm has a time complexity of n2log n, which grows faster than linear but slower than cubic. It's better than O(n3) but still not optimal.

D) Θ(n2log n) - This is the tight bound for the time complexity, indicating that the algorithm's performance is exactly n2log n, which is better than cubic but not as good as exponential time.

E) O(2n) - This algorithm has an exponential time complexity, which grows very rapidly and becomes impractical for relatively large n.

Given these complexities, the most efficient algorithm for relatively large n would be the one with Θ(n2log n) time complexity, which is option D. While it's not as good as linear time complexity, it's significantly better than cubic or exponential time complexities and is often feasible for many practical scenarios where n is relatively large.

# Part B: Data Structures

To efficiently store and process bids for the Unique Bid Blind On-line Auction scenario, we can use a combination of data structures and algorithms tailored to handle large numbers of bids efficiently.

## Data Structure:

### Priority Queue:

A min-heap based priority queue can be used to store the bids. This allows for efficient retrieval of the minimum bid value, which is crucial for determining the minimum bid required for the next round in case of a tie.

## Algorithm:

Hash Table:

A hash table can be used to keep track of the bids submitted by each bidder. This allows for quick lookup and updating of bids associated with each bidder.

Binary Search:

When determining if a bid is unique, binary search can be employed to quickly check if a bid value exists in the priority queue.

## Processing Steps:

Upon receiving a bid, first, check if it meets the minimum bid requirement. If it does, proceed with the following steps:

* Add the bid to the priority queue.
* Update the hash table with the bid submitted by the bidder.
* If there is a tie, restart the auction with the minimum bid set to the current highest bid plus $1.
* If the highest unique bid is found, declare the bidder as the winner.

## Handling Large Numbers of Bids:

Efficient Data Structures:

Using priority queues and hash tables ensures that the time complexity for insertion, retrieval, and updating of bids remains relatively low even with many bids.

Optimize Memory Usage:

Implement memory-efficient data structures to minimize the memory footprint of storing bids.

## Handling Tie Breaker:

If, in the case of a tie, the bidder who placed the highest bid first were to win, the processing steps would remain largely the same. However, in the case of a tie, instead of restarting the auction with the minimum bid set to the current highest bid plus $1, the bidder who placed the first highest bid among the tied bidders would be declared as the winner. This would require keeping track of the timestamp or the order in which bids were submitted by each bidder and selecting the earliest bid among the tied bidders as the winner. This could be achieved by augmenting the data structure used to store bids with additional information such as timestamps.

[ This page intentionally left blank ]