Module Four Hash Tables Reflection & Pseudocode

The following is a reflection the code & it’s functions.

The pseudocode listed below is for understanding of a hash table implementation. The hash table is to efficiently manage bid data through a chained hash table structure. This method is used to handle operations such as the insertion, deletion, searching, & displaying of bids. At the core of this design is the hash function, which can calculate an index for each bid based on it’s bidID. By doing so, the program allows for quick access to the stored bids, reducing search time to nearly constant time under ideal conditions. Finally, we have the collusions, that are when two bids she the same hash value & are resolved through chaining. This allows multiple bids to be stored at the same index without losing accessibility.

Implementation’s strength lies in the adaptability, particularly through the “loadBids()” function, which enables dynamic reading of bid data from a CSV file. Great for when you have large datasets or various input sources, ensuring efficient operations across different scenarios. Making sure to include a simple to use interface, a menu system that allows loading data, searching, deleting bids, & an exit function. Including time-tracking for each operation gives insight into efficiency & performance throughout the task.

However, the chaining mechanism, while effective at handling collisions, may slow searches and deletions as the dataset grows. Longer chains at each index can reduce efficiency. Implementing rehashing or dynamic resizing could mitigate this issue and maintain performance with larger datasets.

Overall, the pseudocode provides a solid foundation for managing key operations with minimal complexity. While its design ensures efficient data access and adaptability, scalability and optimization should be considered for handling larger datasets effectively.

HashTable Class

HashTable holds:

Node struct with:

* + - A Bid struct (stores bid data).
    - An integer key.
    - A pointer to the next node (for chaining).
  + Methods:
    - hash()
    - insert()
    - printAll()
    - remove()
    - search()
  + Private Members:
    - Vector of nodes.
    - Size of the table.

Methods:

* HashTable(): Sets up the table with a default size.
* hash(key): Converts bidID into a key using the modulo operator.
* Insert(Bid): Adds a new Bid to the table. Handles collisions by chaining nodes.
* PrintAll(): Displays four values from the Bid struct.
* Remove(String): Finds and deletes a node with the given string (key).
* Search(String): Looks for a node with the given string and returns it if found.
* Size(): Returns the table size.

Other Functions:

* strToDouble: Converts CSV data into usable numbers.
* Bid Struct: Stores bid data.
* displayBid: Prints bid data.
* loadBids: Loads bids from a CSV file and adds them to the hash table.

Main Function:

* The program driver with a menu to:
  + Add a bid.
  + Load, view, or delete data.
  + Exit the program.
* Tracks how long operations take using the time.h library.

Pseudocode:

Main Function

1. Load CSV path from command-line or use default.
2. Show menu until the user selects '9':
   * Validate user input.
   * If choice is '1', load CSV data and display how long it took.
   * If choice is '2', display all bids.
   * If choice is '3', search for a bid and show how long it took.
   * If choice is '4', remove a bid.
   * If choice is '9', exit and say goodbye.

HashTable Methods:

* hash(int key): Returns key % tableSize.
* Insert(Bid): Adds a new Bid by:
  + Hashing the bidId and checking the location.
  + If empty, add the new node. If not, chain it to the existing nodes.
* PrintAll(): Loops through the table and prints bid info (ID, title, amount, fund).
* Search(String): Finds and returns a node with the matching bid ID.
* Remove(String): Removes a node with the matching bid ID by updating the chain or resetting the node.