Jordan Bankston  
  
CS 300  
  
Professor Ling  
  
June 15, 2025

# Pseudocode

## Partition Function

Vector

Function partition(vector, begin, end):  
 Initialize low to begin  
 Initialize high to end  
 Set pivot to vector at middle index (begin + (end - begin) / 2)  
  
 While true:  
 While vector[low].title < pivot.title:  
 Increment low  
 While vector[high].title > pivot.title:  
 Decrement high  
  
 If low >= high:  
 Return high  
  
 Swap vector[low] with vector[high]  
 Increment low  
 Decrement high

## Quick Sort Function

Function quickSort(vector, begin, end):  
 If begin >= end:  
 Return  
  
 Set partitionIndex = partition(vector, begin, end)  
 Recursively call quickSort on (vector, begin, partitionIndex)  
 Recursively call quickSort on (vector, partitionIndex + 1, end)

## Selection Sort Function

Function selectionSort(vector):  
 For i from 0 to vector.size - 1:  
 Set minIndex = i  
 For j from i + 1 to vector.size:  
 If vector[j].title < vector[minIndex].title:  
 Set minIndex = j  
 If minIndex != i:  
 Swap vector[i] and vector[minIndex]

## Timing Logic in Main

Start timer  
Call selectionSort or quickSort  
Stop timer  
Output time taken in seconds using ticks / CLOCKS\_PER\_SEC

## Hash Table

Main Function()

Read command line arguments  
 Store argument as CSV file path or use default  
Loop while choice is not 9  
 Display menu and get user input  
 If input is 1:  
 Start clock  
 Call loadBids with file path and hashTable  
 Stop clock and show elapsed time  
 If input is 2:  
 Call PrintAll() on hashTable  
 If input is 3:  
 Start clock  
 Call Search() with bidKey  
 Stop clock and show elapsed time  
 If input is 4:  
 Call Remove() with bidKey  
 If input is 9:  
 Exit loop  
End

HashTable Methods:

HashTable::hash(int key)  
 return key % tableSize  
End  
  
HashTable::Insert(Bid)  
 Calculate hash index from bidId  
 If bucket is empty (key == UINT\_MAX):  
 Assign bid to this node  
 Else:  
 Traverse to end of chain at index and append new node  
End  
  
HashTable::PrintAll()  
 For each node in hash table  
 While node exists and is valid:  
 Output bidId, title, amount, fund  
 Move to next node  
End  
  
HashTable::Search(string bidId)  
 Get hashed index from bidId  
 Traverse chain at index  
 If match found, return bid  
 Return empty Bid if not found  
End  
  
HashTable::Remove(string bidId)  
 Get hashed index from bidId  
 Traverse chain while tracking previous node  
 If match found:  
 If at head, reassign head pointer  
 Else, bypass and delete node  
End

## Binary Search Tree

Main Function()

Check for command-line arguments  
 If an argument is given, treat it as the CSV path  
 If no arguments, use the default CSV path for input  
Repeat loop until user selects option '9'  
 Display menu options to the user  
 Prompt user for input and store it in 'choice'  
 If choice == 1  
 Record start time  
 Call function loadBids() and load data into BinarySearchTree bst  
 Print how many records were loaded  
 Record end time and print time taken to load data  
 If choice == 2  
 Call InOrder() to print all bids in ascending order  
 If choice == 3  
 Record start time  
 Prompt user to enter a bid ID to search  
 Call Search() with the given bid ID  
 Display result of the search  
 Record end time and print time taken to search  
 If choice == 4  
 Prompt user for bid ID to remove  
 Call Remove() with that bid ID  
 If choice == 9  
 Exit the loop and end the program  
 Print a goodbye message  
End

BinarySearchTree Methods:

BinarySearchTree::Insert(Bid)  
 If root is null  
 Create a new node using the bid and assign it to root  
 Else  
 Traverse tree to find correct position based on bidId  
 Insert new node as left or right child accordingly  
End  
  
BinarySearchTree::InOrder(Node)  
 If the node is null, return  
 Call InOrder() on node's left child  
 Print bid details  
 Call InOrder() on node's right child  
End  
  
BinarySearchTree::Search(String)  
 Set current to root  
 While current is not null  
 If current.bidId matches input, return bid  
 Else move left or right based on comparison  
 If not found, return empty bid  
End  
  
BinarySearchTree::Remove(String)  
 Traverse to find the node to delete  
 Handle the three deletion cases:  
 No child – delete directly  
 One child – link parent to the child  
 Two children – find in-order successor, swap values, and delete successor  
End

## Evaluation

### Big O Runtime and Memory Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| Load Data | O(1) | O(1) - O(n)\* | O(log n) |
| Search | O(n) | O(1) - O(n)\* | O(log n) - O(n)\*\* |
| Sort / Print | O(n log n) | O(n) | O(n) |

\*Depends on collision resolution method.

\*\*Depends on tree balance.

### Advantages and Disadvantages

- Vector: Simple and fast for loading data. However, sorting is expensive and searching is linear.

- Hash Table: Fast search and insertion with well-distributed keys. Performance may degrade with too many collisions.

- Binary Search Tree: Offers sorted order and balanced performance. Can degrade to linear if unbalanced.

### Recommendation

Based on the expected frequent searches and relatively infrequent full data loads or prints, the Hash Table is recommended. It provides near-constant search time when designed properly and balances efficiency with memory use.