CS 300 Project One Submission

Main Function()

Open file using fstream

If file does not open

Print ERROR

Else

Open file

Return

For each line within file

Parse into courseTitle, courseNumber, prerequisite

Check for errors

If line does not have 2 or more parameters

Print ERROR

Check next line

Close File

Read CMD argument

Store argument as CSV file path

If no argument load default CSV file path

Loop

Output menu

Get input

Store menuchoice

Get input

Store datachoice

Validate

If choice not 1-4 or 8

Error

If choice = 1

If BinarySearchTree

loadBids store CSV in BinarySearchTree

Else if vector

loadBids and store CSV in vector courseList

Else HashTable

loadBids and store CSV in HashTable

Output

Amount records in CSV

If choice = 2

If BinarySearchTree

validateTree()

Else if vector

validateList()

Else HashTable

validateTable()

If choice = 3

If BinarySearchTree

printCourseTree()

Else if vector

printCourseList()

Else HashTable

printCourseTable()

If choice = 4

If BinarySearchTree

printTree()

Else if vector

sortList()

printList()

Else HashTable

sortTable()

printTable()

If choice = 8

Exit

Struct Course

courseNum

courseName

preCount

prelist

Course()

courseNum = courseId

courseName= courseName

preCount = 0

prelist = “”

Class BinaryTree

Struct Node

Course

Right pointer

Left pointer

Root

printTree()

BinaryTree()

sortList()

get vector sort

if lowest index ≥ highest index

return 0

partition()

set lowIndex = value returned from partition

recursively quicksort passing vector, lowest index, lowIndex

recursively quicksort passing vector, lowIndex +1, highest index

Class HashTable

Struct bucket

Course

Key

Next pointer

Hash()

printTable

list HashTable

printTree()

new Node pointer call root

root = NULL

if

root = NULL

return

recursively find left most Node

print

courseNum, courseName

loop 0 to preCount

for each course in prelist

print courseNum

recursively find right most Node

printList()

loop courseList

print

courseNum, courseName

loop 0 to preCount

for each course in prelist

print

courseNum

printTable()

create new Node pointer

loop through list

print

courseNum

courseName

loop 0 to preCount

for each Course in prelist

printCourse()

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

Given the above analysis, I would recommend utilizing a HashTable because HashTables can outperform O(n). A Binary Search Tree is slower than a HashTable but faster than a vector, so while it is not the worst option it is not the best option either. A vector would take the longest amount of time to run and return values. Since my pseudocode focuses on searching for course numbers the fastest return would be via HashTable.