PSEUDOCODE for opening file, reading, parse and format and create objects for each data structure.

VECTOR:

FUNCTION loadFileVector():

OPEN filename from dataStructure “courseList.txt”

WHILE not end of filename:

READ line from filename

IF line correct format:

courseList = parseList(dataStructure):

vector.append(course)

ELSE

DISPLAY end of list

CLOSE fileName

HASH Table:

FUNCTION loadFileHash():

OPEN filename from dataStructure “courseList.txt”

WHILE not end of filename:

READ line from filename:  
 IF line correct format: courseList = parseList(dataStructure):

hashTable.insert(course.courseNumber, prereq.preReqisits):

ELSE

CLOSE filename

Binary search Tree:

FUNCTION loadFileTree():

OPEN file from dataStructure “courseList.txt”

WHILE not end:

READ line from filename;

courseList = parseCourseList(line)(tree.insert)

ELSE

CLOSE fileName

PSUEDOCODE for menu

FUNCTION main():

BEGIN

SET data to load to dataStructure

DISPLAY options for menu “Choose an option from the menu below:”

DISPLAY “Print course List”

END

CALL loadCourse <dataStructure> ()

PRINT course list in alphanumeric order

CALL printCourse <dataStructure>()

PRINT course titles and prerequisites

CALL printCourse <dataStructure> courseNameandNumber()

courseNumber = user input

courseName = user input

EXIT

PSEUDOCODE for alphanumeric order in each data structure

Vector:

BEGIN

OPEN dataStructure filename():

INCREMENT list in alphanumeric order from low to high sort(vector)

FOR sorted courseList:

DISPLAY courseList

END

HASH table:

FUNCTION print courseListHash():

courseList = hashTable.values()

FOR course in couresList:

DISPLAY courseList():

EXIT

BINARY SEARCH TREE:

FUNCTION print courseListTree():

sortCourse = tree. inOrder()

inOrder(root->low):

inOrder(root->dataStructure):

inOrder(root->high):

FOR course in courseListTree():

DISPLAY courseList()

ELSE end

**EVALUATION**

Worst case run time

Vector pseudocode Cost per line Execution Big O Value

* Opening and reading a file 1 O(n) O(n)
* Parse lines and create 1 O(n) O(n)

objects

Hash table

* Opening and reading file 1 O(n) O(n)
* Parse lines and create O(1) O(n) O(n)

objects

BST

* Opening and reading file 1 O(n) O(n)
* Parse lines and create O(logn) O(n) O(logn)

Object

**Advantages and Disadvantages**

Vector-

Vectors can contain larger storage conditions and perform better when the access to the data is in a sequence.

Vectors are much slower in searching for elements compared to hash tables and BSTs. Insertions and deletions in vectors in the middle of the data structure are not very cost effective

Hash Table-

Hash tables can retrieve data much faster than other structures and can look up elements much faster than others.

Due to the complexity of structure, hash tables require much more storage than other structures

BSTs-

Trees are very efficient workflow when inserting or deleting data into the structure and is very suitable method for sorting

Trees also require more storage and memory.

CONCLUSION

BSTs are a much more efficient method when using data structures. The efficient workflow the BSTs provide allow them to be used for faster run times and be more cost efficient than the other data structure methods.