Project 1

**class CourseList**

**struct Course**

string courseNumber

string courseName

vector<string> prerequisites

**If vector** string key

**If vector** bool operator < (Course course)

return key < course.key

declare ifstream fileStream

declare courseList *(of class Vector, Hashtable, or BinarySearchTree)*

**bool openFile(string userFile)**

fileStream.open(userFile)

IF open fails

display error

RETURN 1

ENDIF

processFile()

fileStream.close()

END openFile

**bool processFile()**

declare set<string> courseNums

**If Hashtable** declare vector<string> numsToSort

WHILE there is still data to read

getLine from file and store in courseData

IF (get course number and store in token - getline(stream, token, ‘,’))

insert token into set courseNums

ENDIF

ENDWHILE

fileStream.clear()

fileStream.seekg(0, beg)

WHILE there is still data to read

instantiate empty Course object newCourse

getLine from file and store in courseData

IF courseData is not empty

instantiate stringstream stream with courseData

IF (get course number and store in token - getline(stream, token, ‘,’))

newCourse.courseNumber = token

**If vector** newCourse.key = token

**If Hashtable** numsToSort.push\_back(token)

IF (get course title and store in token - getline(stream, token, ‘,’))

newCourse.courseName = token

WHILE (get prerequisite and store in token - getline(stream, token, ‘,’))

IF this token is in the set courseNums

add to newCourse prerequisites vector with push\_back(token)

ELSE

invalid prerequisite error

ENDIF

ENDWHILE

ELSE

missing course name error

ENDIF

ELSE

missing course number error

ENDIF

ENDIF

**If vector** push\_back(newCourse)

**If Hashtable OR BinarySearchTree** courseList.Insert(newcourse)

ENDWHILE

END processFile

**void printCourse(String courseNumber)**

**If vector**

FOR each course in courseList

IF course is a match

print course number and name

FOR each prerequisite in vector of prerequisites

print prerequisite number and name

ENDFOR

ENDIF

ENDFOR

END printCourseInformation

**If Hashtable OR BinarySearchTree**

course = courseList.Search(courseNumber)

print course number and name

FOR each prerequisite in vector of prerequisites

print prerequisite course number

ENDFOR

END printCourse

**void printAll()**

**If vector**

sort(courseList.begin(), courseList.end())

FOR each course in courseList

print course number and name

FOR each prerequisite in vector of prerequisites

print prerequisite course number

ENDFOR

ENDFOR

END printAll

**If Hashtable**

sort(numsToSort.begin(), numsToSort.end())

FOR courseNum in numsToSort

course = courseList.search(courseNum)

print course number and name

FOR each prerequisite in vector of prerequisites

print prerequisite course number

ENDFOR

ENDFOR

**If BinarySearchTree**

courseList.inOrder()

**class Hashtable / BinarySearchTree**

**Hashtable**

**struct Node**

Course course

unsigned int key

Node next

Node()

key = UINT\_MAX

next = nullptr

Node(Course aCourse, unsigned int aKey) : Node()

course = aCourse

key = aKey

declare vector<Node> courseList

**unsigned int hash(string key)**

hashCode = choose initial value

FOR each character char in key

hashCode = (hashCode \* choose a multiplier) + char ASCII value

ENDFOR

return hashCode % size of vector

END hash

**Insert(Course course)**

key = hash(course.courseNumber)

currNode = courseList.at(key)

IF (currNode -> key == UINT\_MAX)

currNode -> key = key

currNode -> course = course

ELSE

WHILE currNode -> next != nullptr

currNode = currNode -> next

ENDWHILE

currNode -> next = new node (course, key)

END insert

**Search(string courseNumber)**

key = hash(course.courseNumber)

currNode = courseList.at(key)

IF currNode -> key == key

WHILE currNode != null

IF currNode -> course.courseNumber == courseNumber

return currNode -> course

ENDIF

currNode = currNode -> next

ENDWHILE

return empty course

ELSE

return empty course

ENDIF

END search

**BinarySearchTree**

**struct Node**

Course course

Node left

Node right

Node()

left = nullptr

right = nullptr

Node(Course aCourse) : Node()

course = aCourse

**Insert(Course course)**

IF root == nullptr

root = new node

ELSE

addNode(root, course)

ENDIF

END insert

**addNode(Node node, Course course)**

IF getCourseNumber(node -> course.courseNumber) > getCourseNumber(course.courseNumber)

IF node -> left is nullptr

node -> left = new node

ELSE

addNode(node -> left, course)

ENDIF

ELSE

IF node -> right is nullptr

node -> right = new node

ELSE

addNode(node -> right, course)

ENDIF

ENDIF

END addNode

**getCourseNumber(string courseNumber)**

create empty string courseNum

FOR each character char in courseNumber

IF isdigit(char)

add char to courseNum

ENDIF

ENDFOR

convert courseNum from string to int

RETURN courseNum

END getCourseNumber

**Search(string courseNumber)**

start at root = currNode

WHILE currNode is not nullptr

IF currNode -> course.courseNumber == courseNumber

RETURN currNode -> course

ELSE IF getCourseNumber(currNode -> course.courseNumber) > getCourseNumber(courseNumber))

continue traversal left

ELSE

continue traversal right

RETURN empty course

END search

**inOrder(Node node)**

IF node is not nullptr

inOrder(node->left)

print course number and name

FOR each prerequisite in vector of prerequisites

print prererquisite number and name

ENDFOR

inOrder(node->right)

ENDIF

**class main**

**main()**

instantiate new CourseList object courseList

WHILE choice != 9

display menu options

1. Load Data Structure
2. Print Course List
3. Print Course
4. Exit

switch (choice)

case 1:

prompt user for file to open, store in userFile

load file into data structure - courseList.openFile(userFile)

break

case 2:

courseList.printAll()

break

case 3:

prompt user for course number to search, store in courseNumber

courseList.printCourse(courseNumber)

break

ENDWHILE

RETURN 0

| **openFile()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| declare ifstream fileStream | 1 | 1 | 1 |
| declare vector courseList | 1 | 1 | 1 |
| fileStream.open(userFile) | 1 | 1 | 1 |
| IF open fails | 1 | 1 | 1 |
| display error | 1 | 1 | 1 |
| RETURN 1 | 1 | 1 | 1 |
| processFile() |  |  |  |
| fileStream.close | 1 | 1 | 1 |
| **processFile()** |  |  |  |
| declare set<string> courseNums | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| insert token into set courseNums | log n | n | n log n |
| fileStream.clear() | 1 | 1 | 1 |
| fileStream.seekg(0, beg) | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| instantiate empty Course object newCourse | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF courseData is not empty | 1 | n | n |
| instantiate stringstream stream with courseData | 1 | n |  |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseNumber = token | 1 | n | n |
| newCourse.key = token | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseName = token | 1 | n | n |
| WHILE (getline(stream, token, ‘,’)) | 1 | 2n | 2n |
| IF this token is in the set courseNums | log n | n | n log n |
| add to newCourse prerequisites vector with push\_back(token) | 1 | n | n |
| push\_back(newCourse) | 1 | n | n |
| **Total Cost** | | 2(n log n) + 15n + 10 | |
| **Runtime** | | O(n log n) | |

**Vector**

**Hashtable**

| **openFile()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| declare ifstream fileStream | 1 | 1 | 1 |
| declare vector courseList | 1 | 1 | 1 |
| fileStream.open(userFile) | 1 | 1 | 1 |
| IF open fails | 1 | 1 | 1 |
| display error | 1 | 1 | 1 |
| RETURN 1 | 1 | 1 | 1 |
| processFile() |  |  |  |
| fileStream.close | 1 | 1 | 1 |
| **processFile()** |  |  |  |
| declare set<string> courseNums | 1 | 1 | 1 |
| declare vector<string> numsToSort | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| insert token into set courseNums | log n | n | n log n |
| fileStream.clear() | 1 | 1 | 1 |
| fileStream.seekg(0, beg) | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| instantiate empty Course object newCourse | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF courseData is not empty | 1 | n | n |
| instantiate stringstream stream with courseData | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseNumber = token | 1 | n | n |
| numsToSort.push\_back(token) | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseName = token | 1 | n | n |
| WHILE (getline(stream, token, ‘,’)) | 1 | 2n | 2n |
| IF this token is in the set courseNums | log n | n | n log n |
| add to newCourse prerequisites vector with push\_back(token) | 1 | n | n |
| courseList.Insert(newcourse) | 1 | n | n |
| **Total Cost** | | 2(n log n) + 17n + 11 | |
| **Runtime** | | O(n log n) | |

**Binary Search Tree**

| **openFile()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| declare ifstream fileStream | 1 | 1 | 1 |
| declare vector courseList | 1 | 1 | 1 |
| fileStream.open(userFile) | 1 | 1 | 1 |
| IF open fails | 1 | 1 | 1 |
| display error | 1 | 1 | 1 |
| RETURN 1 | 1 | 1 | 1 |
| processFile() |  |  |  |
| fileStream.close | 1 | 1 | 1 |
| **processFile()** |  |  |  |
| declare set<string> courseNums | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| insert token into set courseNums | log n | n | n log n |
| fileStream.clear() | 1 | 1 | 1 |
| fileStream.seekg(0, beg) | 1 | 1 | 1 |
| WHILE there is still data to read | 1 | n | n |
| instantiate empty Course object newCourse | 1 | n | n |
| getLine from file and store in courseData | 1 | n | n |
| IF courseData is not empty | 1 | n | n |
| instantiate stringstream stream with courseData | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseNumber = token | 1 | n | n |
| IF (getline(stream, token, ‘,’)) | 1 | n | n |
| newCourse.courseName = token | 1 | n | n |
| WHILE (getline(stream, token, ‘,’)) | 1 | 2n | 2n |
| IF this token is in the set courseNums | log n | n | n log n |
| add to newCourse prerequisites vector with push\_back(token) | 1 | n | n |
| courseList.Insert(newcourse) | log n | n | n log n |
| **Total Cost** | | 3(n log n) + 15n + 10 | |
| **Runtime** | | O(n log n) | |

A vector has the advantage of being able to insert into the end of the list in constant time. They are less efficient at adding into the middle, but for this case we only need to add to the end. The size is also dynamic which is good if the list needs to grow, but resizing may require reallocation. A vector also has the advantage of requiring less code to write. Using a vector has the disadvantage of being inefficient at searching and may require sorting depending on the search algorithm used. Considering that we are using a small vector, a sort wouldn’t be necessary and a linear search should work just fine.

A hashtable has the advantage of efficient inserting and searching, both of which are required functionalities for the program. However, these advantages are more apparent for large sets of data and we are working with a small set of data. A disadvantage is that collisions need to be handled and chaining can slow search time. A hashtable is also not great for sorted iteration which is needed for printing the sample schedule.

An advantage of the binary search tree is that if it is balanced, then the operations are more efficient since minimum tree height is log2(n). A binary search tree also has the advantage of always being ordered since elements are compared and placed in the correct position when being inserted. A disadvantage of using a binary search tree is that it requires moree coding to geet it implemented. Also, if the tree is unbalanced then the time complexity of operations is diminished and has a worst case of O(n).

Based on the Big O analysis results and my own analysis of the data structures, I know that I don’t want to use Hashtable since the data set is so small and also since sorting can be an issue. A binary search tree would be my choice if the data set weren’t so small, but since doing a linear search on a vector for the printCourse function won’t be too expensive there shouldn’t be any major differences in performance with our value of n, even if more courses were to be added since there can only be so many courses. The same goes for the sort() function for printing all courses. For these reasons, I will choose to use a vector for my implementation.