Sarah Tomlinson Project One

Pseudocode

Read File:

```
CREATE method void loadCourses (string csvPath, data structure)
OPEN file
    IF file cannot be opened
        PRINT error message
WHILE not at the end of the file (EOF):
    FOR each line
        IF a line has fewer than two values
            RETURN an error
        ELSE
            IF there are 3 or more parameters on the line
                CONTINUE reading line until newline is encountered
CLOSE file
```

function Main ():

```
SET CSV file path to argument
IF no argument
    SET CSV file path to default

INITIALIZE user choice to 0
WHILE menu choice is not 9
    PRINT menu choices
    GET user input and SET menu choice to user input
    GET user input and SET data structure choice to user input
    IF user choice is not 1-3 or 9, THROW error //input validation
    IF user choice is 1 // "Load Courses" Option
        IF BinarySearchTree
            CALL loadCourses and SET BinarySearchTree to CSV data
        ELSE IF vector
            CALL loadCourses and SET vector courseList to CSV data
        ELSE IF HashTable
            CALL loadCourses and SET HashTable courseTable to CSV data
        PRINT number of records that are in the CSV file
    IF user choice is 2 // "Print Courses in Alphanumeric Order"
        IF BinarySearch Tree
            CALL printTree()
        ELSE IF vector
            CALL sortList()
            CALL printList()
        ELSE IF HashTable
            CALL sortTable()
            CALL printTable()
```

```
        IF user choice is 3 // "Find Course"
            GET user input to search and SET to userSearch
            IF BinarySearch Tree
                CALL printCourseTree(userSearch)
            ELSE IF vector
                CALL printCourseList(userSearch)
            ELSE IF HashTable
                CALL printCourseTable(userSearch)
        IF user choice is 9 // "Exit"
            EXIT application
        PRINT "Goodbye"


// Vector Method
struct Course:
    courseNum
    courseTitle
    preReq

Vector<Course> loadCourses (string csvPath):
    FOR each row of file
        CREATE course object (courseNum, courseTitle, prereqs)
        SET course.courseNum to courseNum
        SET course.courseTitle to courseTitle
        SET course.prereqs to prereqs

void printCourseInfo(vector<Course> courseInfo, string courseNum):
    FOR all courses
        IF courseNum matches input
            PRINT course info
            FOR each prereq of the course
                PRINT the prereq course info
void parseLine (line):
    SPLIT line using comma as delimiter

vector createVector (Vector<Course> courseInfo):
    FOR entire file
        FOR all lines in file
            ADD courseNum to vector
            ADD courseTitle to vector
            WHILE there is no new line
                ADD prerequisite to vector
```

```
void searchCourse (Vector<Course> courseInfo, String courseNum):
    FOR all courses
        IF the course is the same as courseNum
            PRINT out the course info
            FOR each prereq
                PRINT prereq info
void printSorted(courses):

int partition(vector<Course>& courses, int begin, int end):
    INITIALIZE lowIndex to first element
    INITIALIZE highIndex to last element
    INITIALIZE midpoint to lowIndex + (highIndex – lowIndex) / 2
    INITIALIZE pivot to midpoint
    WHILE pivot is less than highIndex
        DECREMENT highIndex
    // SWAP low and high index
    SET temp value to lowIndex
    SET lowIndex to highIndex
    SET highIndex to temp

void quicksort(vector<Course>& courses, int begin, int end):
    SET mid to 0, lowIndex to begin, highIndex to end
    IF begin is greater than or equal to end
        RETURN
    SET lowEndIndex to partition (courses, lowIndex, highIndex)
    CALL recursively to quickSort
    quicksort (courses, lowIndex, lowEndIndex)
    quicksort (courses, lowEndIndex + 1, highIndex)
```

// BinarySearchTree Method

```
Class BinaryTree{}
    Struct Node
        Course
        Right pointer
        Left pointer
    Root

BinarySearchTree (Tree<Course> loadCourses (string csvPath):
    FOR each row of file
        CREATE course object (courseNum, courseTitle, prereqs)
        IF node is greater than courseNum
            IF left node is null
                SET left node to new node
```

```
            ELSE
                ADD this node
        ELSE
            IF right node is null
                SET right node to new node
            ELSE
                ADD this node
    RETURN Tree

void printCourseInfo (Tree<Course> courseInfo, string courseNum):
    IF course node is not null
        CALL printCourseInfo for left child recursively
        PRINT course info from course node
        CALL printCourseInfo for right child recursively
        PRINT course info from course node

void parseLine (line):
    SPLIT line using comma as delimiter

void searchCourse (Tree<Course> courses, String courseNum):
    INITIALIZE current node equal to root
    FOR all courses
        IF current courseNum and courseNum is equal to 0
            RETURN current courseNum
        ELSE IF courseNum is smaller than current node
            SET current equal to current->left (TRAVERSE left)
        ELSE (courseNum is larger than current node
            SET current node to current->right (TRAVERSE right)
        RETURN course
```

// Hash Method

```
Class HashTable{}
    Struct bucket
        Course
        Key
        Next pointer
    HashTable()

HashTable<Course> loadCourses (string csvPath):
    FOR each row of file
        CREATE course object (courseNum, courseTitle, prereqs)
        ADD to structure at hash position
        SET first string to course structure at courseNum
        SET second string to structure at CourseTitle
```

```
        CALL numPrereqs to count prereqs
            SET prereqs to structure at prereqs
    RETURN HashTable

int Hash key (key):
    //need to decide how we want to hash the string CourseNum
    DEFINE hash of key
    RETURN hash of key

int numPrereqs (HashTable<Course> courseInfo, Course c):
    INITIALIZE key that hashes courseNum
    GET node using key set to new node
    WHILE node is not null
        IF node pointer course equals courseNum
            SET numPrereqs to node prereqs size
        FOR all prereqs in total prereqs
            INCREMENT numPrereqs
        ELSE
            SET node to next node

void printCourseInfo (HashTable<Course> courseInfo, string courseNum):
    INITIALIZE key by hashing course
    GET number with key
    SET number to new node
    FOR all courses
        IF courseNum matches input
            PRINT course info
            FOR each prereq of the course
                PRINT the prereq course info
        ELSE
            SET node to point to next node

void parseLine (line):
    SPLIT line using comma as delimiter

void searchCourse (HashTable<Course> courses, String courseNum):
    FOR all courses
        IF the course is the same as courseNum
            PRINT course info
            FOR each prereq
                PRINT prereq info
```

Vector	Line Cost	# Times Executes	Total Cost
Create vector	1	1	1
For each line in file	1	n	n
Create vector course object	1	1	1
While prereq exists	1	n	n
Append prereq	1	n	n
Exit file	1	n	n
Get courseNum	1	n	1
Return prereqs	n	n	n
		Total Cost	5n+1
		Runtime	O(n)

HashTable	Line Cost	# Times Executes	Total Cost
Create Hash Table method	1	1	1
Create key for course	1	n	n
If no key found	1	n	n
Add key to node	1	n	n
Else	1	n	n
Set old node to UNIT_MAX	1	n	n
Set old node to course	1	n	n
Set old node to null	1	n	n
Else	1	n	n
Find next open node	1	n	n
Set new node to current node to end	2	n	n
		Total Cost	10n + 1
		Runtime	O(n)

Tree	Line Cost	# Times Executes	Total Cost
Create Tree method	1	1	1
If root is null	1	1	1
Add root	1	1	1
If node is less than root, traverse left	1	n	n
If there is no left node	1	n	n
This node is left	1	n	n
If node is greater than root, traverse left	1	n	n
If there is no right node	1	n	n
This node is right	1	n	n
For each line of file	1	n	n
Create vector for courseId, name and prereqs	3	n	3n
		Total Cost	10n + 3
		Runtime	O(n)

Advantages and Disadvantages

Vectors make it easy to add and remove items from a list, but you must search the vector line by line for specific courses until the course is found. This can increase runtime for the worst-case scenario where the course being searched for is the last one in the list. This is a simple method to implement for this type of program and could work efficiently but is not as sustainable as the scale of the number of courses grows.

Hash tables use a created key that can help search more efficiently for courses in the list. Instead of searching each line as with vectors, hash tables use the key to search buckets in a logical order until the item is either found or not found. The actual creation of the hash table is more complex than creating a vector, and the table cannot be sorted, because it is created using buckets as items are added.

Binary Search Trees are efficient to search because they compare the value with the root, and traverse left or right depending on if the value is less than or greater than the root. This way, only half of the tree is searched, no matter where the item being searched for lives. This is the most efficient data structure of these three to search for because of this, and the one that I recommend using for this project. It is easy to traverse the list to add and delete items, but can become unbalanced based on the root and new items that are added.