Briana Long
CS 300

```
# include <iostream>
#include <class files name>

//open courses file
Function readCourseData(file):
        Open file
        For each line:
                courseData = split(line, “,”)
                Course = newCourse

Function print courseList
        Sort list
        For courses in courseList
                Print

Function printCourseNum
        For course
                If course.number == courseNum
                        Print Course title
                        Print course prerequisites

Function exit
        If yes
                Exit course
        If no
                Loop to choice menu


//Define the course data
Class Course
        Course name = name
        Int course number
        Int totalPrerequisite course
                If none = null

// Vector pseudocode
```

```
int numPrerequisiteCourses(Vector<Course> courses, Course c) {
        totalPrerequisites = prerequisites of course c
        for each prerequisite p in totalPrerequisites
                add prerequisites of p to totalPrerequisites
        print number of totalPrerequisites
}

void printSampleSchedule(Vector<Course> courses) {

}

void printCourseInformation(Vector<Course> courses, String courseNumber) {
        for all courses
                if the course is the same as courseNumber
                        print out the course information
                        for each prerequisite of the course
                                print the prerequisite course information
}

// Hashtable pseudocode
int numPrerequisiteCourses(Hashtable<Course> courses) {
        totalPrerequisites = perquisites of all courses
                For each prerequisite
                        P = totalPrerequisite
                Add prerequisite in Hashtable
        Print number of totalPrerequisites
return
}

void printSampleSchedule(Hashtable<Course> courses) {
        Print course name
                If course has prerequisites
                        Print number of courses
return
}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {
        Print course title
        Print course prerequisites
```

```
return
}


// Tree pseudocode
int numPrerequisiteCourses(Tree<Course> courses) {
        totalPrerequisites
        For each prerequisite in totalPrerequisites
                Add nodes to totalPrerequisites
        Print number of totalPrerequisites
return
}


void printSampleSchedule(Tree<Course> courses) {
        Print course name
        Print course information
return
}


void printCourseInformation(Tree<Course> courses, String courseNumber) {
        Course = find courseNumber
        If found
                Print course information
                For each prerequisite
                        Print course information
return
}
```

Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

Code	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	n
`  if the course is the same as courseNumber`	1	n	n
`  print out the course information`	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)

The flow of the program would start with opening the file and the first line being read. The data on the first line is used to create a new course object. Then the course is inserted into the data structure. This is then repeated for the whole file. Then the user menu is used to choose a data structure to load the file from sorting the course name and numbers with prerequisites. Then the list is printed. When a specific course is prompted it will then be searched for using the course number. The title and prerequisites are printed. Then a choice to terminate occurs.

Hash tables have an operation time of O(1). While a binary tree has an operation time of O(log n). A vector sorting has an operation time of O(n log n). Thus hash tables are the quickest then binary trees then vectors. The binary search tree is the most suitable choice as a sorting method. As this method is easy to implement. The courses have a small number of elements. And it can support multiple keys with the same value. As a few courses may have the same prerequisites.