Project One

Eric Florence

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) {
   for all courses
      print course name
         if course has prerequisites
            for each prerequisite
               print prerequisite
}

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 courses, Course c) {

  totalPrerequisites = Hashtable[c]
  for each prerequisite p in totalPrerequisites
     add prerequisites in Hashtable[p] to totalPrerequisites
  print number of totalPrerequisites

}

void printSampleSchedule(Hashtable courses) {

  for all key, value pair in courses
     print key course name
        if value has prerequisites
           for each prerequisites
              print prerequisites

}

void printCourseInformation(Hashtable courses, String courseNumber) {

  for all courses
     if the course is the same as courseNumber
        print out the course information
        for each prerequisite of the Hashtable[course]
           print the prerequisite course information

}
```

TREE PSEUDOCODE

```
int numPrerequisiteCourses(Tree courses, Node c) {

   totalPrerequisites = left and right child of Node c
   for each prerequisite p in totalPrerequisites
      add left and right Nodes of node p to totalPrerequisites
   print number of totalPrerequisites

}

void printSampleSchedule(Tree courses) {

   for all Nodes as courses
      print course name
         if course has left node
               print left node as prerequisite
         if course has right node
               print right node as prerequisite

}
void printCourseInformation(Tree courses, String courseNumber) {

   for all Nodes
      if the course is the same as courseNumber
         print out the node's information
         if course has left node
               print left node as prerequisite course information
         if course has right node
               print right node as prerequisite course information
```

```
        end Function
      else
        if course has left node
            goto left node
        if course has right node
            goto right node
}
```

MENU PSEUDOCODE

```
Int Main(){
While(true){
int flag
string flag 2
flag = 0
PRINT    "1. Load Data Structure"
"2. Print Course List"
"3. Print Course"
"9. Exit"
```

INPUT flag

IF flag equals 9

BREAK

ELSE IF flag equals 1

Public LOADFILE (HASH TABLE FUNCTION/PSEUDOCODE)

ELSE IF flag equals 2

OUTPUT full course list

ELSE IF flag equals 3

INPUT Course Title

OUTPUT numPrerequisiteCourses() and printCourseInformation()

}

}

PRINT LIST PSEUDOCODE

string sort(string s){

LOOP the first and second String character by character and get a chunk of all strings or numbers

IF chunks are numbers or strings

IF numbers sort numerically

PRINT sorted list

ELSE use String compareTo()

PRINT sorted list

}

EVALUATION

VECTOR:

Linear and most expensive per line

HASH TABLE:

the average time required to search for an element in a hash table is O(1 or n).

TREE:

Access in the tree takes about log2(n) comparisons.

RECOMMENDATION

Since we know the size of the input of courses available for students, it would be best to use a Hash Table. But for example, if the courses were constantly changing in a way where we'd need to insert/update and delete courses within the data structure than going with a BST would be advantageous. Since we are also looking to order the courses, again Hash Table here would be the most advantageous. We want all functions on the course data to keep in constant time so my recommendation is using a hash table.