**Oliver Milani**

**CS 300 Project 1**

//Open File

void openFile(){

**get return value for open file function**

**if that return value is less than 0**

**print “File not found.”**

**else**

**read File**

**check if function has necessary parameters.**

**}**

//Create course objects

void createCourseObject(Vector<Course> courses, String courseNumber)

**for all courses**

**if the course is the same as courseNumber**

**read fileType**

**if course has fileType**

**read object and store into hash table**

**else**

**//Print course info**

void printCourseInformation(Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information from Hash Table**

**for each prerequisite of the course**

**print the prerequisite course information**

**}**

//Menu

int courseMenu(int selection){

**display button 1 “Load Data Structure”**

**display button 2 “Print Course List”**

**display button 3 “Print Course”**

**display button 4 “Exit”**

**if selection is 1**

**call createCourseObject()**

**if selection is 2**

**call printCourseList()**

**if selection is 3**

**call printCourseInformation()**

**if selection is 4**

**print “Exiting”**

**exit program.**

**}**

//Print Course List in Alphanumeric Order

Bid printCourseList(){

**implement inserting a course into the tree**

**if course is alphanumerically higher then add to left**

**if no left course**

**this course becomes left**

**else recurse down the left course**

**else**

**IF no right course**

**this course becomes right**

**else**

**recurse down the left course**

**print tree**

**}**

| **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 |
| **get return value for open file function** | 1 | 1 | 1 |
| **if that return value is less than 0** | 1 | n | n |
| **print “File not found.”** | 1 | 1 | 1 |
| **read File** | 1 | n | n |
| **read fileType** | 1 | 1 | 1 |
| **if course has fileType** | 1 | n | n |
| **read object and store into hash table** | 1 | n | n |
| **implement inserting a course into the tree** | 1 | 1 | 1 |
| **if course is alphanumerically higher then add to left** | 1 | n | n |
| **if no left course** | 1 | n | n |
| **this course becomes left** | 1 | n | n |
| **else recurse down the left course** | 1 | n | n |
| **IF no right course** | 1 | n | n |
| **this course becomes right** | 1 | n | n |
| **recurse down the left course** | 1 | n | n |
| **print tree** | 1 | n | n |
| **Total Cost** | | | 16n + 5 |
| **Runtime** | | | O(n) |

Vector sorting, hash tables, and binary search trees all have good advantages and disadvantages. Vector sorting is the simplest of the three. Vector sorting will perform basic sorting needs and will get general sorting done easily. It’s disadvantages come up quickly as it lacks the ability to quickly sort or perform well at all compared to other sorting algorithms. There are some faster ways of vector sorting than others, but in general it is not as efficient as the n-squared steps can be too high.

Hash tables have a lot of advantages, starting with that they are easy to synchronize. Hash tables also end up being more efficient than most other methods. They are widely used because of those abilities to be easy to use and manipulate and stay efficient at the same time. The biggest downfall for hash tables are hash collisions. A hash collision is when two input string are producing the same result. They are hard to predict and when they happen, it can cause huge inefficiency.

Binary Search Trees are very easy to implement. They are equally as easy to traverse. Binary Search Trees can also be a bit more independent as they do not rely on other libraries. They excel at finding higher or lower elements and ordering stats. Biggest disadvantage with a tree is that I can take a long time to search causing it to be inefficient at times.

Given those facts, I think a binary search tree would be the best to use in the code. At first glance I think I would have chosen a hash table. But due to the similar names of the courses and the small chance that there could be a hash collision, choosing a tree would just allow for more security.