# CS 300 Pseudocode Document

**// 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) {**

**courses = all current courses scheduled/registered**

**if registered courses is not null && matched courseNumber**

**cout courses**

**else**

**cout no scheduled classes**

**}**

**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) {**

**Hash::totalPrerequisites**

**For totalPrerequisites**

**Print totalPrerequisites**

**}**

**void printSampleSchedule(Hashtable<Course> courses) {**

**Hash:: courses**

**For all courses**

**If courseNumber is equal to course in courses**

**Cout courses**

**Else**

**Cout no courses**

**}**

**void printCourseInformation(Hashtable<Course> courses, String courseNumber)**

**if courseNumber == courses**

**cout course information**

**else**

**cout no course information**

**}**

**// Tree pseudocode**

**int numPrerequisiteCourses(Tree<Course> courses) {**

**Node totalPrerequisites = new node**

**for each prerequisite p in totalPrerequisites**

**add prerequisites of p to node totalPrerequisites**

**print node totalPrerequisites**

**}**

**void printSampleSchedule(Tree<Course> courses) {**

**Node courses = new node**

**For each node courses is not null**

**Cout node courses**

**else**

**cout no scheduled classes**

**}**

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

**node courses = new node**

**node course number = new node**

**node course information = new node**

**if node courseNumber == node courses**

**cout node course information**

**else**

**cout no course information**

**}**

**void loadCourses(cvspath, hashTable) {**

**Print loading file << cvspath**

**// initialize the parser with given path**

**cvs parser file equals cvs parser(cvspath);**

**// read and display header row - optional**

**vector<string> header = file.getHeader();**

**for auto const c header {**

**print c**

**}**

**Print empty string**

**Try {**

**For int equals I file row count increase i**

**Course course**

**course id equals file index 0**

**course title equals file index 1**

**course prereqs equals file index 2**

**Hash table insert(course)**

**}**

**Catch (cvs error) [**

**Get error**

**}**

**}**

**// Below defines a table to hold course in formation this will also be done in the hashTable class in private and public**

**Int main()int argc, char\* argv[] {**

**Hash table \* courseTable**

**}**

**void printSampleSchedule(Hashtable<Course> courses) {**

**Hash:: courses**

**For all courses**

**If courseNumber is equal to course in courses**

**print courses**

**Else**

**Print string no courses**

**}**

**void printCourseInformation(Hashtable<Course> courses, String courseNumber)**

**if courseNumber equals courses**

**print course information**

**else**

**print string no course information**

**}**

**// main method for menu**

**int main() {**

**cout welcome message**

**while (true) {**

**int flag**

**string flag2**

**flag=0**

**//menu options**

**cout 1.Load Data Structure**

**cout 2.Print Course List**

**cout 3.Print Course**

**cout 8. Exit**

**cout What would you like to do ?**

**cin flag;**

**cout new empty line**

**if (flag==9){**

**break;**

**} else if (flag==1) {**

**loadCourses(cvspath, hashTable)**

**}**

**else if(flag==2) {**

**// method to sort hash table**

**sortUsingHash(a, n)**

**}**

**else if (flag==3) {**

**Cout What course do you want to know about?**

**Cin flag2;**

**cout empty new line**

**transform(flag2.begin(), flag2.end(), flag2.begin(), ::tolower);**

**if (flag2 == "csci400") {**

**Cout CSCI400,Large Software Development\n Prerequisites:CSCI301, CSCI350**

**} else**

**Cout flag**

**Cout that is not a valid option**

**return 0;**

**}**

**// print course with each data structure**

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

**while courses**

**std::sort()**

**}**

**void printComputerScienceCourses (Hashtable<Course> courses, String courseNumber)**

**while courses**

**std::multimap<String, std::string> multimap = invertMap(map)**

**}**

**void printComputerScienceCourses (Tree<Course> courses, String courseNumber) {**

**struct Node \*root equals NULL;**

**root equals insert(root, arr[0]);**

**for (int I equals 1 I less than n I inscreases) {**

**root equals insert(root, arr[i])**

**}**

**int i equals 0**

**storeSortedCourseTree(root, arr, i)**

**}’**

**---------------------------------------RUN TIM ANALYSIS IS ON THE NEXT PAGE ------------------------------------------**

## Vector Runtime Analysis

Run time would be O(n) for a vector as they are fast, but the iterations of the pseudocode above is 20n+1. This would be 20-line cost and 1 time of execution to print course information

* Advantages for vectors are that is it is a dynamic array. The size can be increased and there is reserved space for the vector. Finally, a vector can store different type of objects.
* Disadvantages for vectors is that they can consume a large amount of data.

## HashMap Runtime Analysis

Run time would be O(n) for a hash table but the iterations of the pseudocode above is 18n+1. This would be 18-line cost and 1 time of execution to print course information

* Advantages for hash tables are that is it is faster than searching through an array, great for nested for loops.
* Disadvantages for hash table is that they can cause collisions, which is unavoidable. These collisions can cause slow run time speeds

## Binary Search Tree Runtime Analysis

Run time would be O(n) for a hash table but the iterations of the pseudocode above is 26n+1. This would be 26-line cost and 1 time of execution to print course information

* Binary Search trees have one main advantage, their simplicity. This us allows us to see and reflect relationships between data.
* Disadvantages is that it normally employs a recursive approach which could be more difficult to some developers and that at first trees are error prone and difficult to implement.

**---------------------------------------RECOMMENDATION IS ON THE NEXT PAGE -----------------------------------------**

**Recommendation**

All the data structures we are presented to use are great and all are fairly the same the same speed wise when it comes to this project, according to the runtime analysis I did above. For the next project I plan on using a vector to implement the code for the criteria. The reason for this is that the given amount of data we were given to load and parse through, will not be large enough to cause a slow down for a large. Vectors are easily implemented and I am most comfortable with hem, and this is another reason why I will be using it for my project two.