# CS 300 Pseudocode Document

## Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

**// CSV Parser Pseudocode**

CSV Parser {

Define a vector to collect data structures of courses

Initialize parser using specific path

Try (error catching)

For each file row

Create data structure from the CSV

Add the data structure to the list

Catch errors

Return list

Sort list using a sort function

Display sorted list

}

**// 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) {

Prompt user for course

If course not in “courses vector”

Display error

return

Else

Count number of prerequisites for all courses with numPrerequisisteCourses function

Print desired courses course information with printCourseInformation function

}

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 = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Hashtable hashtable(Hashtable<Course> courses) {

create struct for course

create numPrerequisites using the numPrerequisites function

create course pointer to next course

initialize with a course

create a vector to hold courses

define table size

define hash key variable

return courses

}

Void printSampleSchedule(Hashtable<Course> courses) {

For table size

Start at first bucket

While current course is not null

Print course

}

node printCourseInformation(Hashtable<Course> courses, String courseNumber) {

For table size

Start at first bucket

If key is not default value

Print course information of that bid

Point course to next course

While the current node is not pointing to null

Print information until there are no more courses

Create empty node

Return empty node

}

**// Tree pseudocode**

int numPrerequisiteCourses(Tree<Course> courses) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Tree tree() {

Set root to null

}

Tree add(root node, course c) {

Current node is root node

If root is null, course c becomes root

Else

If current key is bigger than course c

If current left pointer is null

Course c becomes current left pointer

Else

Recurse down left subtree

Else If current key is less than course c

If current right pointer is null

Course c becomes current right pointer

Else

Recurse down right subtree

}

void printSampleSchedule(Tree<Course> courses) {

if course node is not null

recursive call to printSampleSchedule(course’s left pointer as a parameter)

Display course information

recursive call to printSampleSchedule(course’s right pointer as a parameter)

}

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

Set current course node to root node

While current node is not nullptr

If current node’s course number is equal to course number

Return current course node

If course number is smaller than current node

Current node is now current node’s left pointer

Else

Current node is now current node’s right pointer

Create empty course node

Return empty course node

}

**// Main Pseudocode**

Int main (){

Insert csvPath

Print menu choices

While choice is not 4

switch

case 1

Load in CSV data with csv parser function

Case 2

Print sorted course List with PrintSampleSchedule

Case 3

Print “Enter Course key”

Print Course Information with printCourseInformation with key as parameter

If courseInformation returns empty bid

Print “Course not found”

Else

For totalPrerequisites of course

Print Course

Case 4

exit

Print “Goodbye”

}

**Runtime Analysis**

| **Code(Vector)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 5 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information in order (quicksort)** | 5 | Log n | 5N |
| **for each prerequisite of the course** | 1 | n | N |
| **print the prerequisite course information** | 4 | n | N |
| **Total Cost** | | | 16n + 13 |
| **Runtime** | | | O(n) |

| **Code(hash table)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 5 | n | N |
| **if the course is the same as courseNumber** | 1 | logn | Logn |
| **print out the course information in order** | 4 | logn | logn |
| **for each prerequisite of the course** | 1 | 1 | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | | | 15n + 1 |
| **Runtime** | | | O(n) |

| **Code(Binary Tree)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 12 | Log2n | n |
| **print out the course information** | 10 | 1 | 1 |
| **for each prerequisite of the course** | 4 | Log2n | n |
| **print the prerequisite course information** | 4 | Log2n | n |
| **Total Cost** | | | 31n + 1 |
| **Runtime** | | | O(n) |

Advantages and Disadvantages

The advantage of a vector list is that there are less computations to make, and no wasted memory. The only memory used is the only memory that is used for keeping the list. All of the other techniques to holding data structures require extra memory for pointers, or unused buckets. The advantage of a hash table is the quick search time. Hash tables have a faster search time than the other two techniques, but the drawback of the hash table is if the table has too many collisions, the hash table can take up unnecessary space in the memory. The advantage of a hash table is compromised by all the extra space a hash bucket may have and not even use. Lastly, the advantage of a binary tree is the balance of memory allocatio and speed. Binary search trees have a fast search time and don’t use too much extra memory. For that reason, I recommend the binary search tree for the task.