# 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).

//Menu Pseudocode

1. Load courses

Call to CSVParser method to open and load CSV data

Call to Table Structure (hash table, binary tree, linked list) in order to load data into its structure

1. Print sorted list

Call to Print Course List method in the data structure

1. Print Specific course

Prompt user for course Name/Number

Call to Search method within data structure

Print course name and information to screen

1. Exit

Exits program

//CSV Loading/Course Struct Pseudocode

Structure Course {

Course Number

Course Description

List Prerequisites

}

Void CSVParser{

Get and Open CSV File with data in it

While you have not reached the end of the file

Get next line of data

Parse by splitting at comma

Make a Course Structure object by:

Putting first item in course number

second in course description

and while you have not reached end of line

Add the rest of data to prerequisites list

Take this new course structure object and insert into table

(then will move onto next line of data and continue)

Close File

}

//Print All Courses

//Vector:

QuickSort (Vector<Course>){

Until our list is completely sorted

find a partition value

Take the low and high indexes (beginning and end) of your collection, and find the midpoint. This will become the pivot point.

While your value at low index is less than value at high index:

Compare value at low index to value at pivot index.

if the value in the low index is less than the value at the pivot, increment the low index by one.

While your value at pivot index is less than the value at high index:

Compare value at pivot index to value at high index.

If value at pivot index is less than value at high index, decrement high index by one

Compare the high index to low index.

If high index is less than low index, you have found your partition value (which is equal to the high index)

Otherwise, swap the value at the low index with the value at the high index.

Repeat this entire process until you have found your partition value.

Once this high index is found, we will divide our collection into a low part and a high part which will be divided at the partition value.

Then repeat and find partition values for these parts and continue until list is sorted.

}

Print All Courses(Vector<Course>){

Call Quick Sort Method to sort smallest -> largest

Then, start at index 0 and print each course until get to end of list

}

//Hash Table:

Print All Courses (Hashtable<Course>){

Start at index 0 and visit each bucket

If the bucket is null, then move to next

If item is in bucket, print this item.

Then,look at next pointer for this item

If next pointer is null, move to next bucket

Otherwise, set up current = the item.

While Current is not null, print out item

Reset current = the item in current’s next pointer

}

//Binary Tree:

Print All Courses(Tree<Course>){

Start at root

Recursively call through the tree down the left and down the right sides until you get to the bottom, and print along the way

}

//print specific course out (from original pseudocode:

//vector

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

}

//hash table

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

Take course number string and turn into a key

Run key through hash function to get the index

Go to this index

If the key of the item matches the key you are looking for:

Print this course information.

Return.

Else:

Set up current course variable = first course at index

While the current course isn’t null:

If the current course’s key = keyed course Number

Print this course information

Return

}

//Tree:

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

Make variable called current and set equal to root of tree.

If current course is empty -> course was not found.

If current’s course number matches courseNumber:

Print this course.

Else:

If current course’s number is larger than courseNumber,

Move to the current’s left and repeat comparison with this course.

Else:

Move to current’s right and repeat comparison with this course

(this method will recursively call to itself and repeat these steps until it finds the course or gets to the bottom of the tree and does not find it)

}

// Course Structure pseudocode:

Structure Course{

Course number

Course description

List of prerequisites

}

//File IO pseudocode:

Void CSVParser {

Get and open CSV file with data in it

While you have not reached the end of the file

Get next line of data

Parse data by splitting at the comma and construct a data object course out of the data.

Add this course object to vector

Move on to the next line of data and continue

Close the file.

}

// Vector pseudocode

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = 0

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

return number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

for each course in courses

if all prerequisites have been met by student or numPrerequisiteCourses is 0

print out this course to screen.

}

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, string courseNumber) {

get the key from the course number

run through hash function

return number prerequisites at that value (table index) returned from hash function

}

int hash(int number){

return number % courseTableSize

}

void printSampleSchedule(Hashtable<Course> courses) {

foreach bucket in the table

First, start at index 0 [no prerequisites]

Go to bucket at this index

Print each course at this bucket until “next” pointer is null for the courses

In this bucket.

}

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

Take course number string and turn into a key

Run key through hash function to get the index

Go to this index

If the key of the item matches the key you are looking for:

Print this course information.

Return.

Else:

Set up current course variable = first course at index

While the current course isn’t null:

If the current course’s key = keyed course Number

Print this course information

Return

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

Get the course number

Start at current = root.

If current’s course number matches the course number

Return number prerequisites for this course

Else If the course’s number is greater than at the current node

then move to the right and compare here.

Else

Move to left and compare here

Once found:

Return number prerequisites

}

void printSampleSchedule(Tree<Course> courses) {

Start at root.

If course has no prerequisites, print to screen.

Continue traversing down left side of tree, printing the courses that have no prerequisites out.

Then, down the right side, if the course prerequisite has been met, print out the course information.

}

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

Make variable called current and set equal to root of tree.

If current course is empty -> course was not found.

If current’s course number matches courseNumber:

Print this course.

Else:

If current course’s number is larger than courseNumber,

Move to the current’s left and repeat comparison with this course.

Else:

Move to current’s right and repeat comparison with this course

(this method will recursively call to itself and repeat these steps until it finds the course or gets to the bottom of the tree and does not find it)

}

## 

## Runtime Analysis

**Vector Analysis:**

| **Print Course Information 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) |

| **Num Prerequisites Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| totalPrerequisites = 0 | 1 | 1 | 1 |
| for each prerequisite p in totalPrerequisites | 1 | n | n |
| add prerequisites of p to totalPrerequisites | 1 | n | n |
| return number of totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 2n +2 |
| **Runtime** | | | O(n) |

| **Print Sample Schedule Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for each course in courses | 1 | n | n |
| if all prerequisites have been met by student or numPrerequisiteCourses is 0 | 2n+2 (runs the num prereq code) | n | 2n2+2n |
| print out this course to screen. | 1 | n | n |
| **Total Cost** | | | 2n2 +4n |
| **Runtime** | | | O(n2) |

| **Print All Courses Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Call Quick Sort Method to sort smallest -> largest | 1 | 1 | 1 |
| Quick Sort: | O(Log n) | 1 | Log n |
| For(i=0; i< vector size; i++) | 1 | n | n |
| print out this course to screen. | 1 | n | n |
| **Total Cost** | | | Logn + 2n + 1 |
| **Runtime** | | | O(n) |

Hash Table Analysis:

| **Num Prerequisites Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| get the key from the course number | 1 | 1 | 1 |
| run through hash function | 1 | 1 | 1 |
| return number prerequisites at that value (table index) returned from hash function | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

| **Print Sample Schedule** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| foreach bucket in the table | 1 | n | n |
| Print each course at this bucket until “next” pointer is null for the courses | 1 | n | n |
| **Total Cost** | | | 2n |
| **Runtime** | | | O(n) |

| **Print Couse Information** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| foreach bucket in the table | 1 | n | n |
| Take course number string and turn into a key | 1 | 1 | 1 |
| Run key through hash function to get the index | 1 | 1 | 1 |
| **If:** the key of the item at this index matches the key you are looking for:  Print this course information.  Return. | 1 | 1 | 1 |
| **Else:**  Set up current course variable = first course at index | 1 | 1 | 1 |
| While the current course isn’t null: | 1 | n | n |
| If the current course’s key = keyed course Number | 1 | 1 | 1 |
| Print this course information | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| **Total Cost** | | | n+5 |
| **Runtime** | | | O(n) |

| **Print All Courses Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at index 0 and visit each bucket | 1 | 1 | 1 |
| If the bucket is null, then move to next | 1 | n | n |
| If item is in bucket, look at next pointer for this item | 1 | n | n |
| While next pointer is not null print current item | 1 | n | n |
| Reset current to the next | 1 | n | n |
| **Total Cost** | | | 4n+1 |
| **Runtime** | | | O(n) |

Binary Tree Analysis:

| **Num Prerequisites Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at current = root. | 1 | 1 | 1 |
| If current’s course number matches the course number  Return number prerequisites for this course | 1 | 1 | 1 |
| Else If the course’s number is greater than at the current node  then move to the right and compare here. | 1 | Log n | Log n |
| Else  Move to left and compare here |  | Log n | Log n |
| Once found:  Return number prerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 2logn+2 |
| **Runtime** | | | O(log n) |

| **Print Sample Schedule Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at current = root. | 1 | 1 | 1 |
| If course has no prerequisites, print to screen. | 1 | n | 1 |
| Continue traversing down left side of tree, printing the courses that have no prerequisites out. | 1 | Log n | Log n |
| Then, down the right side, if the course prerequisite has been met, print out the course information. | 1 | Log n | Log n |
| **Total Cost** | | | 2logn+2 |
| **Runtime** | | | O(log n) |

| **Print Sample Schedule Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at current = root. | 1 | 1 | 1 |
| If current course is empty -> course was not found. | 1 | n | 1 |
| Else If current’s course number matches courseNumber:  Print this course. | 2 | 1 | 1 |
| Else:  If current course’s number is larger than courseNumber, | 1 | 1 | 1 |
| Move to the current’s left and Call to function again and repeat process | 2 | Log n | 2logn |
| Else:  Move to current’s right and repeat comparison with this course | 2 | Log n | 2logn  (this is same as above) |
| **Total Cost** | | | 2logn+4 |
| **Runtime** | | | O(log n) |

| **Print All Courses Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at Root | 1 | 1 | 1 |
| Recursively call through the tree down the left until you get to bottom of tree until the left pointer is null | 1 | Log n | Log n |
| Print item | 1 | Log n | Log n |
| Recursively call down the right sides until the right pointer is null | 1 | Log n | Log n |
| print item | 1 | Log n | Log n |
| **Total Cost** | | | 4 logn + 1 |
| **Runtime** | | | O(log n) |