# 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).

// 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) {

sort(courses, by courseNumber)

for course in courses:

print(course.courseNumber + ": " + course.title)

}

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) {

course = courses.get(courseNumber)

totalPrerequisites = prerequisites of course

for each prerequisite p in totalPrerequisites:

add prerequisites of p to totalPrerequisites

return size of totalPrerequisites

}

void printSampleSchedule(Hashtable<Course> courses) {

courseNumbers = getKeys(courses)

sort(courseNumbers)

for courseNumber in courseNumbers:

course = courses.get(courseNumber)

print(course.courseNumber + ": " + course.title)

}

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

course = courses.get(courseNumber)

if course != null:

print("Course Number: " + course.courseNumber)

print("Title: " + course.title)

print("Prerequisites: ")

if course.prerequisites.isEmpty():

print("None")

else:

for prerequisite in course.prerequisites:

print(prerequisite)

else:

print("Course not found: " + courseNumber)

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

course = findCourse(courses.root, courseNumber)

totalPrerequisites = prerequisites of course

for each prerequisite p in totalPrerequisites:

add prerequisites of p to totalPrerequisites

return size of totalPrerequisites

}

void printSampleSchedule(Tree<Course> courses) {

printCourseList(courses.root)

}

void printCourseList(TreeNode node){

if node != null:

printCourseList(node.left)

print(node.course.courseNumber + ": " + node.course.title)

printCourseList(node.right)

}

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

TreeNode node = findCourse(courses.root, courseNumber)

if node != null:

print("Course Number: " + node.course.courseNumber)

print("Title: " + node.course.title)

print("Prerequisites: ")

if node.course.prerequisites.isEmpty():

print("None")

else:

for prerequisite in node.course.prerequisites:

print(prerequisite)

else:

print("Course not found: " + courseNumber)

}

## Example Runtime Analysis

| **Code VECTOR** | **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) |

| **Code HASHTABLE** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Access course by courseNumber** | 1 | 1 | 1 |
| **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** | | | 2n + 2 |
| **Runtime** | | | O(n) |

| **Code TREE** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Search for course by courseNumber** | 1 | Log n if symmetry tree | Log 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** | | | 2n + log (n) + 1 |
| **Runtime** | | | O(log n) |

|  |  |  |
| --- | --- | --- |
| Data Structure | Advantages | Disadvantages |
| Vector | Simple to implement | Requires sorting for order linear search |
| Hash | Fast element access. | Potential collisions |
| Tree | Maintains order. Fast search, and manipulation | More complex when it comes to building it. Balance important and a factor |

I would begrudgingly suggest using a Binary Search Tree (BST) for this application. A BST naturally keeps data in order, fulfilling the advisor's requirement to display courses alphabetically. Search, insertion, and deletion operations within a balanced BST have a time complexity of O(log n), providing efficiency when working with course information. Additionally, in-order traversal allows for straightforward iteration over courses in the correct order. That all being said, making a BST is no simple task and a huge barrier to entry when trying this out. It is much more complex than alternatives, but its advantages in ordered data and efficient operations make it the ideal choice, especially as the number of courses may increase over time.