# 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

Class Course {

Private:

String id

String name

String[] preReq

Public:

}

Vector<Course> readFile(string file){

Create Vector<Course> courses

open file (file)

for int I while I < file.length()

create new Course Object c

divide into a new course id, name, and preReqs

courses.append(c)

}

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) {

}

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

Class Course {

Private:

String id

String name

String[] preReq

Public:

}

Class hashTable {

Private:

Node constructors

Hash(id)

Public:

Append(course)

Remove(id)

Search(id)

void readFile(string file){

Created int key

open file (file)

for int I while I < file.length()

create new Course Object c

divide into a new course id, name, and preReqs

key = Hash(id)

hashTable.Append(c,key)

}

int numPrerequisiteCourses(Hashtable<Course> courses) {

}

void printSampleSchedule(Hashtable<Course> courses) {

}

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

}

// Tree pseudocode

Class Course {

Private:

String id

String name

String[] preReq

Public:

}

Class tree {

Private:

Node constructors

Public:

inOrder()

preOrder()

postOrder()

Append(course)

Remove(id)

Search(id)

}

void readFile(string file){

open file (file)

for int I while I < file.length()

create new Course Object c

divide into a new course id, name, and preReqs

tree.Append

}

Void inOrder() {

if node is not null

inOrder (left)

Print node

inOrder (right)

}

Display() {

This->inOrder()

}

int numPrerequisiteCourses(Tree<Course> courses) {

}

void printSampleSchedule(Tree<Course> courses) {

}

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

}

**Menu:**

Display options

Switch

1 load data

Load in data

Display results (time/ how many created courses)

2 print course list

Inorder print tree

3 print course

Search tree

Print results

4: exit

**Sort:**

For each course ID

For each course

If the Course ID is less than the course swap

## Example Runtime Analysis

**Vector:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **for all lines** | 1 | n | n |
| **Create new course Object** | 1 | n | n |
| **Split line into course Object** | 1 | n | n |
| **Append courses** | 1 | n | n |
| **Total Cost** | | | 4n + 2 |
| **Runtime** | | | O(n) |

**Hashmap:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create int key** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **For all lines** | 1 | n | n |
| **Create new course Object** | 1 | n | n |
| **Split line into course Object** | 1 | n | n |
| **Set Hash(id)** | 1 | n | n |
| **Append course(c,key)** | 1 | n | n |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

**Tree:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **for all courses** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **Split line into course object** | 1 | n | n |
| **Append to tree** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Recommendation**

Based on the results of my Big O notation charts I believe that a tree will be the best option to fulfill the needs of this project. The resources required to store objects are less than a vector or HashMap. Another nice thing about creating a tree is that the data can be sorted as it is placed in the tree. That way when you call in order on the tree it will print in sorted order. With vectors and HashMap, the data will need to be sorted before it can be printed. The downfall of the tree however is that if all the courses are in sorted order then you will have a linked list which can cause the search function to be less effective.