# 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) {

// Sorting Courses in alphanumeric order (using bubble sort)

Int size = size of course vector

For i from 0 to size - 1

For 0 to size – 1

If(course number of j > course number j + 1)

Swap course j and course j + 1

// Printing Courses

for all courses

print course

if course has a prereq

for all prereqs

print prereq

}

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

**int verify = 0;**

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**verify = 1**

**for each prerequisite of the course**

**print the prerequisite course information**

**if verify = 0**

**print out course not found**

}

Int main{ // Took code from some of our assignments for menu display

Vector<Course> courses;

String courseNumber;

int choice = 0;

while (choice != 4) {

cout << "Menu:" << endl;

cout << " 1. Load Data" << endl;

cout << " 2. Print Course List" << endl;

cout << " 3. Print Course Information" << endl;

cout << " 4. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

}

Switch statement (choice):

Case 1:

load data up;

Case 2:

printSampleSchedule(courses);

Case 3:

Cout << “Enter Course Number”

Cin >> courseNumber;

printCourseInformation(courses, courseNumber);

Case 4:

Choice = 4

Cout << “Thank you for using our Program”

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

totalPrerequisites = prereqs of course

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Hashtable<Course> courses) {

// Sorting through hashtable

For i from 0 to courses.size – 1

For j from 0 to courses.size – 1

If courses value at j > course value at j + 1

Swap course value j and courses value j + 1

// printing schedule

for all key and value pairs in courses

print key (Course name)

if the corresponding value has prereqs

for all prereqs

print prereqs

}

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

int verify = 0;

for all courses

if the course is the same as courseNumber

verify = 1

print out the course information

for each prerequisite of the course

print the prerequisite course information

if verify == 0

print out “Course Not Found”

}

}

Int main{ // Took code from some of our assignments for menu display

Hashtable<Course> courses;

String courseNumber;

int choice = 0;

while (choice != 4) {

cout << "Menu:" << endl;

cout << " 1. Load Data" << endl;

cout << " 2. Print Course List" << endl;

cout << " 3. Print Course Information" << endl;

cout << " 4. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

}

Switch statement (choice):

Case 1:

load data up;

Case 2:

printSampleSchedule(courses);

Case 3:

Cout << “Enter Course Number”

Cin >> courseNumber;

printCourseInformation(courses, courseNumber);

Case 4:

Choice = 4

Cout << “Thank you for using our Program”

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

totalPrerequisites = left and right child of node courses

for each prerequisite in totalPrerequisites

add left node and right node to fill with prereq

print prereqs

}

void printSampleSchedule(Tree<Course> courses) {

Traverse Tree using post traversal

for all Nodes that are courses

print course name

if course has left child

print left child as prereq

if course has right child

print right child as prereq

}

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

int verify = 0;

for all nodes in tree

if the course is the same as courseNumber

print out the course information

verify = 1

if course has left child

print left child as course prereq

if course has right child

print right child as course prereq

if course has left child

go to left node

if course has right child

go to right node

if verify = 0

print out “Course not found”

}

Int main{ // Took code from some of our assignments for menu display

BinarySearchTree\* courses;

String courseNumber;

int choice = 0;

while (choice != 4) {

cout << "Menu:" << endl;

cout << " 1. Load Data" << endl;

cout << " 2. Print Course List" << endl;

cout << " 3. Print Course Information" << endl;

cout << " 4. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

}

Switch statement (choice):

Case 1:

load data up;

Case 2:

printSampleSchedule(courses);

Case 3:

Cout << “Enter Course Number”

Cin >> courseNumber;

printCourseInformation(courses, courseNumber);

Case 4:

Choice = 4

Cout << “Thank you for using our Program”

## 

Vector:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Sorting courses alphanumerically** | **n** | **n** |  |
| **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^2) |

Hash table:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Sorting courses alphanumerically** | **n** | **n** |  |
| **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^2) |

Binary Search Tree:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Sorting courses alphanumerically** | **1** | **Log n** | **Log n** |
| **for all courses** | 1 | Log n | Log n |
| **if the course is the same as courseNumber** | 1 | Log n | Log n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | Log n | Log n |
| **print the prerequisite course information** | 1 | Log n | Log n |
| **Total Cost** | | | 4 log n+1 |
| **Runtime** | | | O(log n) |

Code Reflection:

Vectors are usually the go to for C++ code because they are a dynamic, contiguous array so the size can be changed also elements can be added removed. A vector takes constant time to access its elements. Disadvantages are that insertions and deletions are expensive, they can both cost O(n). The memory needed is also costly. Main advantage of a hash table is synchronization, hash tables are useful because of their key structure this makes looking up values extremely easily. Time complexity of a hash table at worst is O(n). Disadvantages are hash collisions are unavoidable and too many hash collisions will render the table inefficient. Average time complexity of a Binary Search Tree is O(log n) which is the best out of the three. You have to use a recursive solution for the BST. Advantages are that all operations will still keep the time complexity at O(log n). The main disadvantage is that the BST has to be balanced to keep that minty fresh time complexity. I will most likely use a vector for my code because it is simple, straight to the point and dynamic.