# CS 300 Pseudocode Document

At the end of each line, there will be 3 numbers. These correspond, respectively, to line cost, number of times executed, and total cost. Class declaration not analyzed because they will all be O(1). Menu also not analyzed in accordance with rubric.

**Pseudocode**

=====================================================================================

**GLOBAL STRUCT** Course:

DEFINE course ID string 1 | 1 | 1

DEFINE course name string 1 | 1 | 1

DEFINE course prerequisite vector of strings 1 | 1 | 1

CONSTRUCTOR default: 1 | 1 | 1

INITIALIZE above parameters at defaults

CONSTRUCTOR with parameters:

INITIALIZE above listed data to input parameters 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 5

Runtime: O(1)

=====================================================================================

**FUNCTION** VectorFind:

PARAMETERS: Course ID

DECLARE searchCourse 1 | 1 | 1

LOOP through the vector: 1 | n | n

IF current element matches course ID: 1 | n | n

SET searchCourse to current Course 1 | 1 | 1

RETURN searchCourse (Even if no matching course ID found) 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n + 3

Runtime: O(n)

**FUNCTION** VectorPrintAll:

PARAMETERS: printVector

LOOP through the printVector: 1 | n | n

DISPLAY Course ID and name 1 | n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n

Runtime: O(n)

**FUNCTION** PrintOneCourse:

PARAMETERS: Course

DISPLAY Course ID and name 1 | 1 | 1

LOOP through prerequisites: 1 | n | n

DISPLAY each prerequisite 1| n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n + 1

Runtime: O(n)

**FUNCTION** VectorPartition:

PARAMETERS: sortVector (pass by reference), lowIndex, highIndex

CALCULATE middle point of sortVector 1 | 1 | 1

DEFINE pivot as the Course at the middle point 1 | 1 | 1

LOOP until partition found: 1 | n | n

LOOP while element at lowIndex < pivot: 1 | n^2 | n^2

Increment lowIndex 1 | n^2 | n^2

LOOP while element at highIndex > pivot: 1 | n^2 | n^2

Decrement highIndex 1 | n^2 | n^2

IF lowIndex >= highIndex: 1 | n | n

SET partition found to true 1 | 1 | 1

ELSE: 1 | n | n

SWAP Courses at lowIndex and highIndex 1 | 1 | 1

INCREMENT lowIndex 1 | n | n

DECREMENT highIndex 1 | n | n

RETURN highIndex 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 4n^2 + 5n + 5

Runtime: O(n^2)

**FUNCTION** VectorQuickSort:

PARAMETERS: sortVector (pass by reference), lowIndex, highIndex

// Base case

RETURN IF lowIndex >= highIndex 1 | 1 | 1

// Recursion

CALL VectorPartition sortVector, lowIndex, and highIndex n^2 | 1 | n^2

STORE low partition lastIndex 1 | 1 | 1

CALL VectorQuickSort with sortVector, lowIndex, and lastIndex n^2 | 1 | n^2

CALL VectorQuickSort with sortVector, lastIndex + 1, and highIndex n^2 | 1 | n^2

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 3n^2 + 1

Runtime: O(n^2)

=====================================================================================

**CLASS** LinkedList:

PUBLIC Members:

METHOD Constructor

METHOD Destructor

METHOD Append

METHOD Prepend

METHOD PrintList

METHOD Remove

METHOD Find

METHOD ListEmpty

METHOD Size

PRIVATE Members:

**STRUCT** Node:

DEFINE Course

DEFINE next pointer

CONSTRUCTOR default:

INITIALIZE next to null

CONSTRUCTOR with parameters:

INITIALIZE next to null

INITIALIZE Course to parameter Course

POINTER head

POINTER tail

LinkedList **METHOD** Constructor:

PARAMETER: None

SET head and tail pointers to null 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

LinkedList **METHOD** Destructor:

PARAMETERS: None

LOOP through the whole list: 1 | n | n

DELETE each Node 1 | n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n

Runtime: O(n)

LinkedList **METHOD** Append:

PARAMETERS: Course

INITIALIZE newNode with Course 1 | 1 | 1

IF list empty: 1 | 1 | 1

SET head and tail to newNode 1 | 1 | 1

ELSE: 1 | 1 | 1

POINT tail next to newNode 1 | 1 | 1

SET tail to newNode 1 | 1 | 1

INCREASE list size 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 7

Runtime: O(1)

LinkedList **METHOD** Prepend:

PARAMETERS: Course

INITIALIZE newNode with Course 1 | 1 | 1

IF list empty: 1 | 1 | 1

SET head and tail to newNode 1 | 1 | 1

ELSE: 1 | 1 | 1

POINT newNode next to head 1 | 1 | 1

SET head to newNode 1 | 1 | 1

INCREASE list size 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 7

Runtime: O(1)

LinkedList **METHOD** PrintList:

PARAMETERS: Key (default unsigned int max)

IF key < unsigned int max: 1 | 1 | 1

DISPLAY “Key “ 1 | 1 | 1

LOOP while current node is not null: 1 | n | n

IF key < unsigned int max: 1 | n | n

DISPLAY “: “ 1 | 1 | 1

DISPLAY Course ID and name 1 | n | n

LOOP through prerequisites: 1 | n^2 | n^2

DISPLAY each prerequisite 1 | n^2 | n^2

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n^2 + 3n + 3

Runtime: O(n^2)

LinkedList **METHOD** Remove:

PARAMETERS: Course ID

IF the head is the course to be removed: 1 | 1 | 1

SET head to head next 1 | 1 | 1

DECREASE list size 1 | 1 | 1

RETURN 1 | 1 | 1

LOOP while current node is not null: 1 | n | n

IF next node matches the course ID: 1 | n | n

SET current next to node after next 1 | 1 | 1

DELETE next node 1 | 1 | 1

DECREASE list size 1 | 1 | 1

BREAK from loop 1 | 1 | 1

SET current node to next node 1 | n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 3n + 8

Runtime: O(n)

LinkedList **METHOD** Find:

PARAMETERS: Course ID

DECLARE searchCourse 1 | 1 | 1

LOOP while current node is not null: 1 | n | n

IF current Node Course ID matches: 1 | n | n

SET searchCourse to current Course 1 | 1 | 1

BREAK from loop 1 | 1 | 1

SET current to next node 1 | n | n

RETURN searchCourse (Even if no matching course ID found) 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 3n + 4

Runtime: O(n)

LinkedList **METHOD** ListEmpty:

PARAMETERS: None

IF head is null: 1 | 1 | 1

RETURN true 1 | 1 | 1

ELSE: 1 | 1 | 1

RETURN false 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 4

Runtime: O(1)

LinkedList **METHOD** Size:

PARAMETERS: None

RETURN list size 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

=====================================================================================

**CLASS** HashTable:

PUBLIC Members:

METHOD Constructor

METHOD Constructor (Override parameter)

METHOD Destructor

METHOD Insert

METHOD PrintAll

METHOD Remove

METHOD Search

METHOD Sort

PRIVATE Members:

Vector<LinkedList> CourseList

METHOD Hash

HashTable **METHOD** Constructor:

PARAMETERS: None

RESIZE LinkedList vector to default size 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

HashTable **METHOD** Constructor (Override parameter):

PARAMETERS: TableSize

RESIZE LinkedList vector to TableSize 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

HashTable **METHOD** Destructor:

PARAMETERS: None

LOOP through CourseList: 1 | n | n

CALL pop on each course from the back (Calls LinkedList destructor) 1 | n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n

Runtime: O(n)

HashTable **METHOD** Insert:

PARAMETERS: Course

CALL Hash with Course ID parameter and STORE hash location 1 | 1 | 1

AT hash location in CourseList CALL LinkedList method Append with Course 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2

Runtime: O(1)

HashTable **METHOD** PrintAll:

PARAMETERS: None

LOOP through the CourseList: 1 | n | n

IF LinkedList not empty at current position in CourseList: 1 | n | n

CALL LinkedList method PrintList with current CourseList index n^2 | 1 | n^2

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n^2 + 2n

Runtime: O(n^2)

HashTable **METHOD** Remove:

PARAMETERS: Course ID

CALL Hash with Course ID parameter and STORE hash location 1 | 1 | 1

AT hash location in CourseList CALL LinkedList method Remove with Course ID n | 1 | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n + 1

Runtime: O(n)

HashTable **METHOD** Search:

PARAMETERS: Course ID

CALL Hash with Course ID parameter and STORE hash location 1 | 1 | 1

AT hash location in CourseList CALL LinkedList method Find with Course ID n | 1 | n

RETURN Course from previous call 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n + 2

Runtime: O(n)

HashTable **METHOD** Sort:

PARAMETERS: None

DEFINE sortVector 1 | 1 | 1

LOOP through CourseList: 1 | n | n

LOOP through each Node in a bucket: 1 | n^2 | n^2

CALL push\_back on sortVector with Course 1 | n^2 | n^2

CALL VectorQuickSort with sortVector n^2 | 1 | n^2

` RETURN sortVector 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 3n^2 + n + 2

Runtime: O(n^2)

HashTable **METHOD** Hash:

PARAMETERS: Key integer

CALCULATE Key mod TableSize and STORE in hashKey 1 | 1 | 1

RETURN hashKey 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2

Runtime: O(1)

=====================================================================================

**CLASS** BinarySearchTree:

PUBLIC Members:

METHOD Constructor

METHOD Destructor

METHOD DisplayCourses

METHOD Insert

METHOD Remove

METHOD Search

PRIVATE Members:

POINTER root

METHOD recursiveDelete

METHOD printCourse

METHOD recursivePrint

BinarySearchTree **METHOD** Constructor:

PARAMETERS: None

SET root to null pointer 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

BinarySearchTree **METHOD** Destructor:

PARAMETERS: None

CALL recursiveDelete with root pointer n | 1 | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n

Runtime: O(n)

BinarySearchTree **METHOD** recursiveDelete:

PARAMETERS: Node pointer

// Base case

RETURN IF node points to null 1 | 1 | 1

// Recursion

CALL recursiveDelete on left node of node pointer n | 1 | n

CALL recursiveDelete on right node of node pointer n | 1 | n

DELETE node at pointer 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n + 2

Runtime: O(n)

BinarySearchTree **METHOD** Insert:

PARAMETERS: course data

IF root is null: 1 | 1 | 1

SET root to node with course data 1 | 1 | 1

ELSE: 1 | 1 | 1

LOOP while node not pointing to null: 1 | n | n

IF course ID is less than pointer course ID: 1 | n | n

IF left pointer null: 1 | 1 | 1

INSERT node with course data on left side 1 | 1 | 1

BREAK 1 | 1 | 1

ELSE: 1 | n | n

MOVE node pointer to left side 1 | n | n

ELSE: 1 | n | n

IF right pointer null: 1 | 1 | 1

INSERT node with course data on right side 1 | 1 | 1

BREAK 1 | 1 | 1

ELSE: 1 | n | n

MOVE node pointer to the right side 1 | n | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 7n + 9

Runtime: O(n)

BinarySearchTree **METHOD** Remove:

PARAMETERS: course ID

LOOP while node not pointing to null: 1 | n | n

IF node found: 1 | 1 | 1

IF removing leaf node: 1 | 1 | 1

IF removing root: 1 | 1 | 1

SET root to null 1 | 1 | 1

ELSE IF removing left leaf: 1 | 1 | 1

SET left leaf pointer to null 1 | 1 | 1

ELSE: 1 | 1 | 1

SET right leaf pointer to null 1 | 1 | 1

ELSE IF removing left node with one child: 1 | 1 | 1

IF removing root: 1 | 1 | 1

MOVE current node to root and remove root 1 | 1 | 1

ELSE IF removing left internal node: 1 | 1 | 1

MOVE parent left pointer to current left pointer 1 | 1 | 1

ELSE: 1 | 1 | 1

MOVE parent right pointer to current left pointer 1 | 1 | 1

ELSE IF removing right node with one child: 1 | 1 | 1

IF removing root: 1 | 1 | 1

MOVE current node to root and remove root 1 | 1 | 1

ELSE IF removing left internal node: 1 | 1 | 1

MOVE parent left pointer to current right pointer 1 | 1 | 1

ELSE: 1 | 1 | 1

MOVE parent right pointer to current right pointer 1 | 1 | 1

ELSE: // Removing node with two children 1 | 1 | 1

SET successor node to right side 1 | 1 | 1

LOOP while successor is not null: 1 | n | n

SET successor to leftmost node on this branch 1 | n | n

MOVE successor data to parent node 1 | 1 | 1

DELETE successor node 1 | 1 | 1

RETURN // Node has been found 1 | 1 | 1

ELSE IF course ID is less than pointer course ID: 1 | n | n

MOVE pointer left 1 | n | n

ELSE: 1 | n | n

MOVE pointer right 1 | n | n

RETURN // Node not found 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 7n + 28

Runtime: O(n)

BinarySearchTree **METHOD** Search:

PARAMETERS: course ID

LOOP while node pointer not null 1 | n | n

IF course found at node: 1 | 1 | 1

RETURN course 1 | 1 | 1

ELSE IF course ID less than node course ID: 1 | n | n

MOVE pointer left 1 | n | n

ELSE: 1 | n | n

MOVE pointer right 1 | n | n

RETURN course // Even if no course was found 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 5n + 3

Runtime: O(n)

BinarySearchTree **METHOD** DisplayCourses:

PARAMETERS: None

CALL recursivePrint with root node n | 1 | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n

Runtime O(n)

BinarySearchTree **METHOD** recursivePrint:

PARAMETERS: Node pointer

// Base case

RETURN IF node points to null 1 | 1 | 1

// Recursion

CALL recursivePrint with left node n | 1 | n

CALL printCourse on course at node pointer 1 | 1 | 1

CALL recursivePrint with right node n | 1 | n

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n + 2

Runtime: O(n)

BinarySearchTree **METHOD** printCourse:

PARAMETERS: course

DISPLAY course data formatted in an easily readable 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 1

Runtime: O(1)

=====================================================================================

**FUNCTION** ReadFileIntoVector:

PARAMETER is a file string

DEFINE input file stream 1 | 1 | 1

DEFINE courseVector 1 | 1 | 1

DEFINE string input vector 1 | 1 | 1

OPEN file at file string 1 | 1 | 1

IF file open: 1 | 1 | 1

LOOP until end of file: 1 | n | n

DEFINE string stream 1 | n | n

LOOP while next call finds data 1 | n^2 | n^2

CALL getline with string stream, a string holder, 1 | n^2 | n^2

and the comma for the delimitor

PUSH the string holder to the back of the string 1 | n^2 | n^2

input vector

IF less than two elements in vector: 1 | n^2 | n^2

CALL clear on string input vector n | n^2 | n^3

CONTINUE to next loop iteration 1 | n^2 | n^2

ELSE: 1 | n^2| n^2

CREATE Course object (defined above ReadFile) 1 | n^2 | n^2

with course ID, name, and any

prerequisites

CALL push\_back vector method with Course 1 | n^2 | n^2

LOOP through courseVector: 1 | n | n

CALL VectorFind with all prerequisites from Course object at n | n | n^2

each node

IF node contains prerequisites that are not in the vector: 1 | n | n

CALL pop on vector element index 1 | n | n

RETURN courseVector 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n^3 + 9n^2 + 5n + 6

Runtime: O(n^3)

=====================================================================================

**FUNCTION** ReadFileIntoHashTable:

PARAMETER is a file string

DEFINE input file stream 1 | 1 | 1

DEFINE hash table 1 | 1 | 1

DEFINE string input vector 1 | 1 | 1

OPEN file at file string 1 | 1 | 1

IF file open: 1 | 1 | 1

LOOP until end of file: 1 | n | n

DEFINE string stream 1 | 1 | 1

LOOP while next call finds data 1 | n^2 | n^2

CALL getline with string stream, a string holder, and 1 | n^2 | n^2

the comma for the delimitor

PUSH the string holder to the back of the string input 1 | n^2 | n^2

vector

IF less than two elements in vector: 1| n^2 | n^2

CALL clear on string input vector n | n^2 | n^3

CONTINUE to next loop iteration 1 | n^2 | n^2

ELSE: 1 | n^2 | n^2

CREATE Course object (defined above ReadFile) 1 | n^2 | n^2

with course ID, name, and any prerequisites

CALL HashTable Insert with Course object 1 | n^2 | n^2

LOOP through HashTable nodes: 1 | n | n

CALL HashTable Search with all prerequisites from Course object n | n | n^2

at each node

IF node contains prerequisites that are not in the table: 1 | n | n

CALL HashTable Remove on node n | n | n^2

RETURN hash table 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: n^3 + 10n^2 + 7

Runtime: O(n^3)

=====================================================================================

**FUNCTION** ReadFileIntoBST:

PARAMETER is a file string

DEFINE input file stream 1 | 1 | 1

DEFINE binary search tree 1 | 1 | 1

DEFINE string input vector 1 | 1 | 1

OPEN file at file string 1 | 1 | 1

IF file open: 1 | 1 | 1

LOOP until end of file: 1 | n | n

DEFINE string stream 1 | n | n

LOOP while next call finds data 1 | n^2 | n^2

CALL getline with string stream, a string holder, 1 | n^2 | n^2

and the comma for the delimitor

PUSH the string holder to the back of the string input 1 | n^2 | n^2

vector

IF less than two elements in vector: 1 | n^2 | n^2

CALL clear on string input vector n | n^2 | n^3

CONTINUE to next loop iteration 1 | n^2 | n^2

ELSE: 1 | n^2 | n^2

CREATE Course object (defined above ReadFile) 1 | n^2 | n^2

with course ID, name, and any

prerequisites

CALL BinarySearchTree Insert with Course object n | n^2 | n^3

LOOP through binary search tree nodes: 1 | n | n

CALL BinarySearchTree Search with all prerequisites from Course n | n | n^2

object at each node

IF node contains prerequisites that are not in the tree: 1 | n | n

CALL BinarySearchTree Remove on node n | n | n^2

RETURN binary search tree 1 | 1 | 1

\_\_\_\_\_\_\_\_\_\_\_\_

Total Cost: 2n^3 + 8n^2 + 4n + 6

Runtime: O(n^3)

=====================================================================================

**FUNCTION** DisplayMenu:

PARAMETERS: dataLoaded

DISPLAY option to load data from file into data structure

IF dataLoaded:

DISPLAY option to print course list

DISPLAY option to print individual course

DISPLAY option to exit program

**FUNCTION** MenuChoice:

PARAMETERS: None

DEFINE dataLoaded as false

CALL DisplayMenu

OBTAIN user input:

IF valid input:

SWITCH on user input:

CASE load data structure:

// Vector Data Type

CALL ReadFileIntoVector with file string

// Hash Table Data Type

CALL ReadFileIntoHashTable with file string

// Binary Search Tree Data Type

CALL ReadFileIntoBST with file string

SET dataLoaded to true

CASE print course list:

IF dataLoaded:

// Vector Data Type

CALL VectorQuickSort with VectorData

CALL VectorPrint with VectorData

// Hash Table Data Type

CALL HashTable Sort

CALL VectorPrint with sorted hash table vector

// Binary Search Tree Data Type

// (No sorting needed due to DisplayCourses printing each

// successive node)

CALL BinarySearchTree DisplayCourses

ELSE:

INFORM user no data loaded

CASE print individual Course:

IF dataLoaded:

// Vector Data Type

LOOP through VectorData:

IF element Course ID matches ID to be printed:

CALL PrintOneCourse with Course

// Hash Table Data Type

CALL HashTable Search with Course ID

CALL PrintOneCourse with returned Course

// Binary Search Tree Data Type

CALL BinarySearchTree Search with Course ID

CALL PrintOneCourse with returned Course

ELSE:

INFORM user no data loaded

CASE exit program:

RETURN false to exit program

RETURN true so that program continues

=====================================================================================

**Analysis**

**Vector:**

ADVANTAGES:

* Simple and easy to code
* Easy to understand

DISADVANTAGES:

* Insertion requires moving all downstream elements
* Removal requires moving all downstream elements
* Search speed is always linear
* Requires sorting to print alphanumerically

**Hash Table:**

ADVANTAGES:

* Insertion is very fast at O(1)
* If data is spread out well, removal and search are fast

DISADVANTAGES:

* Sort is very slow at O(n^2)
* Due to alphanumeric printing requiring a sort, it is also very slow
* If data is not spread out well, removal and search are O(n)

**Binary Search Tree:**

ADVANTAGES:

* If data is not sorted prior to loading, insertion, removal, and search generally happen at O(log\_2(n))
* No sort required to print alphanumerically
* Alphanumeric print runs at O(n)

DISADVANTAGES:

* Large memory requirement
* If data is sorted prior to loading, insertion, removal, and search run at O(n)

=====================================================================================

**Recommendation**

I recommend using the binary search tree data type. Due to many of its operations running at logarithmic speed on average and linearly in the worst-case scenario, it is the most consistently fast for this application. A strict memory limit was not mentioned, therefore one of its greatest disadvantages is not an issue. The binary search tree also does not require any sorting in order to print all of the data alphanumerically, and the print method runs on linear time in every case. The other data types required sorting before printing all data. All data types required the same algorithmic speed in order to load from the file, so this is not part of the considerable criteria. For the aforementioned reasons, I plan to use the binary search tree as the data type of choice to store and use the data of this application.