# CS 300 Pseudocode Document

## Example Function Signatures

Below is an example of a function signature that you can use as a guide to help address the program requirements using each data structure for the milestones. The pseudocode for finding and printing course information is also given below and depicted in bold to help you get started. The provided pseudocode is for a vector data structure, so you may use this pseudocode in your first milestone as is. The hash table and tree structures are also shown below. But these structures are left for you to do in future milestones.

//Vector - Milestone 1

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

open file

if no file found

out no file found

else

read file.

Close file

Build Course class

Build vector courseList

Build Object that adds courses to Course

Build identifiers

Course.id

Course.name

Course.preReq

Loop through file size

Add data to list

Add course id

Add course name

Add course preReq

}

//Hash Table - Milestone 2

void searchCourse(HashTable<Course> courses, String courseNumber) {

open file

if no file found

out no file found

else

read file.

Close file

Build “Course” Class

Build a courseList

Build an object to add courses to Course

Build Course identifiers (name, id, preReq)

Loop the list

Add course id(i,1)

Add course name(I,0)

Add course preReq(I,2)

Build hashTable class

Build node UNIT\_MAX

Build node key

Build method to insert into hashTable

Loop through the file

Split each line into different values

Value 1 = ID

Value 2 = name

Value 3 = preReq

If node is opem

Add method for values in hashTable

Else

Make UNIT\_MAX = old node

Name and preReq = Null

Find next node

Fill node

Attach node to the end of the list

Loop course

Print name

Print id

Print preReq

}

//Binary Search Tree – Milestone 3

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

open file

if no file found

out no file found

else

read file.

Close file

Call Course class

Loop the list

Add course id(i,1)

Add course name(I,0)

Add course preReq(I,2)

Build “BinaryTree” class

Build root that points to null

Build insert method

If root is equal to null

Course is root

Else

If coursedId is less than the root

Add to the left

If the left node is null

Add couseId

Else

If courseId is less than leaf

Add node to left

If courseId is greater than leaf

Add node to the right

Else

If the courseID is greater than root

Add node to the Right

If the right is null

Add CourseID

Else

If courseID is less than leaf

Add node to left

If courseId is greater than leaf

Add node to the right

Request user input

Build Print method

If root does not equal null

Traverse to the left

If found

Output

Traverse to the right

If found

Output

## Example Runtime Analysis

When you are ready to analyze the runtime for the Project One data structures for which you created the pseudocode, use the example chart below to support your work. This particular example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document. The example only covers the search function for the vector structure. You do not have to complete your runtime analysis until Project One. However, working on your analysis now may help you understand the changes as you complete the milestones. Don’t forget to include your charts in Project One. You will submit Project One in Module Six.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Build vector** | 1 | 1 | 1 |
| For each line in file | 1 | n | N |
| **Build vector course item for each.** | n | n | N |
| **ID** | 1 | n | n |
| **NAME** | 1 | n | N |
| **preReq** | 1 | n | N |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Build hash table** | 1 | 1 | 1 |
| Build key | 1 | 1 | 1 |
| **Each line in file** | 1 | n | n |
| If nothing in Node | 1 | n | n |
| **Add items** | n | n | n |
| **Else** | 1 | n | n |
| **Make old key UNIT\_MAX** | 1 | n | n |
| **Make old course null** | 1 | n | n |
| **Make preReq null** | 1 | n | n |
| **Find node** | n | n | n |
| **Fill node** | 1 | n | n |
| **Attach node to the end** | 1 | n | n |
| **Total Cost** | | | 10n + 2 |
| **Runtime** | | | O(n) |

| **BinaryTree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Build BinaryTree method** | 1 | 1 | 1 |
| If root = null | 1 | 1 | 1 |
| **Add root** | 1 | 1 | 1 |
| If node < root | 1 | N | N |
| **Go left** | N | n | N |
| **If no left node** | 1 | n | N |
| **Add node to left** | 1 | n | n |
| **If node > root** | 1 | n | n |
| **Go right** | N | n | n |
| **If no right node** | 1 | n | n |
| **Add node to right** | 1 | n | n |
| **Each line in file** | 1 | n | n |
| **ID** | 1 | N | N |
| **Name** | 1 | n | N |
| **preReq** | 1 | n | N |
| **Total Cost** | | | 13n + 3 |
| **Runtime** | | | O(n) |

Analysis

Vector

Vectors are good for adding and removing items with ease as they are quick and simple to edit. If you are trying to search through a vector on the other hand it can take some time and extra resources as nothing is in order and you must search through the entire list.

Hash Table

Hash tables are good for sorting values. Depending on the hash values being used it could cut the time to search through data. Although the time it takes to find the information you are looking for is quick it takes a lot of time to write the code for hash tables compared to other methods. Hash tables also only work with integers so if you wanted to use strings or decimals you wouldn’t be able to.

Binary Tree

Binary Trees are good for sorting data. It sets a value to a node and then compares it to a different one. If the value is smaller, it goes left and if it is larger, it goes right until your value is found. It is not as quick as the hash table. It is however set up for the ability to add more data to it easily.