# CS 300 Pseudocode Document

LoadData(fileName) {

**Declare currRow as integer**

**Declare currCourse as string**

**Read file type from fileName**

**Declare errorCatch as boolean**

**If file type is not supported**

**Output error message**

**End**

**Verify fileName exists**

**If fileName doesn’t exist**

**Output error message**

**End**

**Try**

**Open fileName**

**Set currRow as 1**

**Set currCourse as currRow of fileName**

**While currCourse isn’t blank**

**Set errorCatch as ParseLine(currCourse)**

**If errorCatch is true**

**Output error message**

**Pause**

**Increment currRow**

**Set currCourse as currRow of fileName**

**Catch**

**Output Error Message**

}

boolean ParseLine(courseInfo) {

**Declare i as integer**

**Declare j as integer**

**Declare preReqs as array**

**Declare lineValues as array**

**Declare coursedId as string**

**Declare courseName as string**

**Set lineValues as courseInfo split by “,”**

**If lineValues.size < 2**

**Return true**

**Set courseId as lineValues[0]**

**Set courseName as lineValues[1]**

**Set i as 2**

**Set j as 0**

**While i < lineValues.size**

**Set preReqs [j] as lineValues[i]**

**Increment j**

**Increment i**

**Run addCourse(courseId, courseName, preReqs**

**Return false**

}

Void addCourse(courseId, courseName, preRegs) {

**Set newCourse as courseId, courseName, preRegs**

**Run insertCourse(newCourse)**

}

//Vector - Milestone 1

void searchCourse(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**

}

Void displayCourses() {

**If size.courses is 0**

**Print no courses exist**

**Else if size.courses is 1**

**Print course information**

**for each prerequisite of the course**

**print the prerequisite course information**

**Else Run quicksort(courses, 0, size.courses-1)**

**for each course in courses**

**print course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

Void quicksort(int lowIndex, int highIndex)

**If highIndex-lowIndex is 1 or less**

**End**

**Set midIndex as lowIndex + (highIndex – lowIndex) / 2)**

**If courses(lowIndex) > courses(midIndex)**

**Set tempCourse as courses(midIndex)**

**Set courses(midIndex) as courses(lowIndex)**

**Set courses(lowIndex) as tempCourse**

**Run quicksort(lowIndex, midIndex)**

**If courses(midIndex) > courses(highIndex)**

**Set tempCourse as courses(midIndex)**

**Set courses(midIndex) as courses(highIndex)**

**Set courses(highIndex) as tempCourse**

**Run quicksort(midIndex, highIndex)**

}

Void insertCourse(course course)

**Courses.push\_back(course)**

}

//Hash Table - Milestone 2

void searchCourse(String courseNumber) {

**convert courseNumber to hashkey**

**set printNode as searchList(bucket.node, courseNumber)**

**if printNode is null**

**print course not found**

**else print course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

Node searchList(node currNode, String courseNumber) {

**If currNode is null**

**Return null**

**Else If currNode is courseNumber**

**Return currNode**

**Else set currNode searchList(currNode.next, courseList)**

}

Void displayCourses() {

**Create vector sortedCourses**

**For each course in hashTable**

**If course is not null**

**Run walkChainSort(sortedCourses, course)**

}

Void walkChainSort(vector<Course> sortedCourses, course currCourse) {

**If currCourse.next is nullptr**

**run addToSortedVector(sortedCourses, course)**

**else run walkChainSort(sortedCourses, currCourse.next)**

**run addToSortedVector(sortedCourses, course)**

}

Void addToSortedVector(vector<course> sortedCourses, course newCourse)

**For each course in sortedCourses**

**If i is sortedCourses.size-1**

**If newCourse < sortedCourse(i)**

**sortedCourses.insert(i, newCourse)**

**else sortedCourses.push\_back(newCourse)**

**else if newCourse < sortedCourses(i)**

**sortedCourses.insert(i, newCourse)**

**break**

}

Void insertCourse(course course) {

**Set hashKey as getHash(course.courseNumber)**

**Set newNode with course**

**Set currNode as courses.at(hashKey)**

**If currNode is null**

**Set currNode as newNode**

**Else if currNode.key is UINT\_MAX**

**Assign newNode to currNode**

**Else while currNode.next is not null**

**currNode = currNode.ext**

**Set currNode.next as newNode**

**Set newNode.next as null**

}

//Binary Search Tree – Milestone 3

void searchCourse(String courseNumber) {

**set printNode searchTree(root, courseNumber)**

**if printNode is null**

**print course not found**

**else print course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

Node searchTree(node currNode, String courseNumber) {

**If currNode is null**

**Return null**

**Else if currNode is courseNumber**

**Return currNode**

**Else if courseNumber < currNode**

**Set currNode searchNode(currNode.left, courseNumber)**

**Else Set currNode searchNode(currNode.right, courseNumber)**

}

Void displayCourses() {

**Run inOrder(root)**

}

Void inOrder(node currNode)

**If currNode.left is not null**

**Run inOrder(currNode.left)**

**Else print course information**

**for each prerequisite of the course**

**print the prerequisite course information**

**if currNode.right is not null**

**run inOrder(currNode.right)**

}

Void insertCourse (course newCourse) {

**If root is null**

**set root as newCourse**

**else run addNode(root, newCourse)**

}

Void addNode(node currNode, course newCourse) {

**if newCourse < currNode.course**

**if currNode.left is null**

**Set currNode.left as newCourse**

**Else run addNode(currNode.left, newCourse)**

**Else**

**If currNode.right is null**

**Set currNode.right as newCourse**

**Else run addNode(currNode.right, newCourse)**

}

# Vector Runtime Analysis

| **Code**  **Vector::searchCourse** | **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 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **Total Cost** | | | (2n + 1) + (2n2) |
| **Runtime** | | | O(n) |

| **Code**  **Vector::** **displayCourses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If size.courses is 0** | 1 | 1 | 1 |
| **Print no courses exist** | 1 | 1 | 1 |
| **Else if size.courses is 1** | 1 | 1 | 1 |
| **Print course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **Else Run quicksort(courses, 0, size.courses-1)** | n | 1 | n |
| **for each course in courses** | 1 | n | n |
| **print course information** | 1 | n | n |
| **for each prerequisite of the course** | 1 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **Total Cost** | | | (3n + 4) + (2 n2) |
| **Runtime** | | | O(n) |

| **Code**  **Vector::** **quicksort** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If highIndex-lowIndex is 1 or less** | 1 | 1 | 1 |
| **End** | 1 | 1 | 1 |
| **Set midIndex as lowIndex + (highIndex – lowIndex) / 2)** | 1 | 1 | 1 |
| **If courses(lowIndex) > courses(midIndex)** | 1 | 1 | 1 |
| **Set tempCourse as courses(midIndex)** | 1 | 1 | 1 |
| **Set courses(midIndex) as courses(lowIndex)** | 1 | 1 | 1 |
| **Set courses(lowIndex) as tempCourse** | 1 | 1 | 1 |
| **Run quicksort(lowIndex, midIndex)** | 1 | n/2 | n/2 |
| **If courses(midIndex) > courses(highIndex)** | 1 | 1 | 1 |
| **Set tempCourse as courses(midIndex)** | 1 | 1 | 1 |
| **Set courses(midIndex) as courses(highIndex)** | 1 | 1 | 1 |
| **Set courses(highIndex) as tempCourse** | 1 | 1 | 1 |
| **Run quicksort(midIndex, highIndex)** | 1 | n/2 | n/2 |
| **Total Cost** | | | (3n + 4) + (2 n2) |
| **Runtime** | | | O(n) |

| **Code**  **Vector::** **insertCourse** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Courses.push\_back(course)** | 1 | 1 | 1 |
| **Total Cost** | | | 1 |
| **Runtime** | | | O(1) |

# Hash Table Runtime Analysis

| **Code**  **Hash Table::searchCourse** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **convert courseNumber to hashkey** | 1 | 1 | 1 |
| **set printNode searchList(bucket.node, courseNumber)** | 1 | 1 | 1 |
| **if printNode is null** | 1 | 1 | 1 |
| **print course not found** | 1 | 1 | 1 |
| **else print course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **Total Cost** | | | (2n2 +5) |
| **Runtime** | | | O(n2) |

| **Code (recursive)**  **Hash Table::searchList** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If currNode is null** | 1 | 1 | 1 |
| **Return null** | 1 | 1 | 1 |
| **Else If currNode is courseNumber** | 1 | 1 | 1 |
| **Return currNode** | 1 | 1 | 1 |
| **Else set currNode searchList(currNode.next, courseList)** | 1 | 1 | 1 |
| **Total Cost** | | | 5+T(n) |
| **Runtime** | | | O(n) |

| **Code**  **Hash Table::** **displayCourses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector sortedCourses** | 1 | 1 | 1 |
| **For each course in hashTable** | 1 | n/tablesize | n/tablesize |
| **If course is not null** | 1 | n/tablesize | n/tablesize |
| **Run walkChainSort(sortedCourses, course)** | T(n) | n/tablesize | T(n) /tablesize |
| **Total Cost** | | | (2n2 +5) |
| **Runtime** | | | O(n2) |
| **Code**  **Hash Table::** **walkChainSort** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **If currCourse.next is nullptr** | 1 | 1 | 1 |
| **run addToSortedVector(sortedCourses, course)** | T(n) | 1 | T(n) |
| **else run walkChainSort(sortedCourses, currCourse.next)** | T(n) | 1 | T(n) |
| **run addToSortedVector(sortedCourses, course)** | T(n) | n | T(n)\*n |
| **Total Cost** | | | (2n+1)+n2 |
| **Runtime** | | | O(n2) |
| **Code**  **Hash Table::** **addToSortedVector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **For each course in sortedCourses** | 1 | n | n |
| **If i is sortedCourses.size-1** | 1 | n | N |
| **If newCourse < sortedCourse(i)** | 1 | 1 | 1 |
| **sortedCourses.insert(i, newCourse)** | 1 | 1 | 1 |
| **else sortedCourses.push\_back(newCourse)** | 1 | 1 | 1 |
| **else if newCourse < sortedCourses(i)** | 1 | n | n |
| **sortedCourses.insert(i, newCourse)** | 1 | 1 | 1 |
| **break** | 1 | 1 | 1 |
| **Total Cost** | | | (3n+5) |
| **Runtime** | | | O(n) |

| **Code**  **Hash Table::** **insertCourse** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Set hashKey as getHash(course.courseNumber)** | 1 | 1 | 1 |
| **Set newNode with course** | 1 | 1 | 1 |
| **Set currNode as courses.at(hashKey)** | 1 | 1 | 1 |
| **If currNode is null** | 1 | 1 | 1 |
| **Set currNode as newNode** | 1 | 1 | 1 |
| **Else if currNode.key is UINT\_MAX** | 1 | 1 | 1 |
| **Assign newNode to currNode** | 1 | 1 | 1 |
| **Else while currNode.next is not null** | 1 | n/x | n/x |
| **currNode = currNode.ext** | 1 | n/x | n/x |
| **Set currNode.next as newNode** | 1 | 1 | 1 |
| **Set newNode.next as null** | 1 | 1 | 1 |
| **Total Cost** | | | (2n/x + 9) |
| **Runtime** | | | O(log n) |

# Binary Search Tree Run Time Analysis

| **Code**  **BST::searchCourse** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **set printNode searchTree(courses.root, courseNumber)** | 1 | 1 | 1 |
| **if printNode is null** | 1 | 1 | 1 |
| **print course not found** | 1 | 1 | 1 |
| **else print course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **Total Cost** | | | (2n2 +4) |
| **Runtime** | | | O(n2) |

| **Code (recursive)**  **BST::searchTree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If currNode is null** | 1 | 1 | 1 |
| **Return null** | 1 | 1 | 1 |
| **Else if currNode is courseNumber** | 1 | 1 | 1 |
| **Return currNode** | 1 | 1 | 1 |
| **Else if courseNumber < currNode** | 1 | 1 | 1 |
| **Set currNode searchNode(currNode.left, courseNumber)** | 1 | 1 | 1 |
| **Else Set currNode searchNode(currNode.right, courseNumber)** | 1 | 1 | 1 |
| **Total Cost** | | | 7+T(n/2) |
| **Runtime** | | | O(log n) |

| **Code (recursive)**  **BST::displayCourses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Run inOrder(root)** | n | 1 | 1 |
| **Total Cost** | | | n |
| **Runtime** | | | O(n) |

| **Code (recursive)**  **BST::inOrder** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If currNode.left is not null** | 1 | 1 | 1 |
| **Run inOrder(currNode.left)** | n/2 | 1 | n/2 |
| **Else print course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n2 | n2 |
| **print the prerequisite course information** | 1 | n2 | n2 |
| **if currNode.right is not null** | 1 | 1 | 1 |
| **run inOrder(currNode.right)** | 1 | n/2 | n/2 |
| **Total Cost** | | | (2n/2 + 3) + (2 n2) |
| **Runtime** | | | O(n) |

| **Code (recursive)**  **BST::inertCourse** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If root is null** | 1 | 1 | 1 |
| **set root as newCourse** | 1 | 1 | 1 |
| **else run addNode(root, newCourse)** | Log n | 1 | Log n |
| **Total Cost** | | | (log n + 2 |
| **Runtime** | | | O(log n) |

| **Code (recursive)**  **BST::addNode** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **if newCourse < currNode.course** | 1 | 1 | 1 |
| **if currNode.left is null** | 1 | n/2 | n/2 |
| **Set currNode.left as newCourse** | 1 | n/2 | n/2 |
| **Else run addNode(currNode.left, newCourse)** | Log n | n/2 | Log n/2 |
| **If currNode.right is null** | 1 | n/2 | n/2 |
| **Set currNode.right as newCourse** | 1 | n/2 | n/2 |
| **Else run addNode(currNode.right, newCourse)** | Log n | n/2 | Log n/2 |
| **Total Cost** | | | (2log n/2) + 4n/2) + 1 |
| **Runtime** | | | O(log n) |

# Advantages and Disadvantages

Vector

The main advantage of using the vector is the simplicity of adding a new course. This is essentially performed with one line of code, resulting in a O(1) runtime. Searching for a specific course has a best case of O(1), but this only happens if the course being searched for is the first course of the vector. Otherwise, the search results in O(n). Displaying all courses in alphanumerical order also results in an O(n) runtime. Using a vector makes the code easy to work with. A vector is a great choice for small amounts of data that won’t grow too much.

Hash Table

The Hash Table is similar to the vector, with both a distinct advantage and disadvantage. The search function on a Hash Table has more chances of reaching the O(1) runtime, but even when it doesn’t hit the best case, the runtime is O(log n). This is great for large amounts of data. However, hash tables aren’t great for sorting the data. The data needs to be parsed and sorted into a vector, before it can be displayed. This results in a runtime of O(n2). Hash Tables are great for large amounts of data, just don’t try to sort it.

Binary Search Tree

Binary Search Trees are great for large amounts of data. Sorting results in a runtime of O(n). Searching has a worst case runtime of O(n), but that is an extreme worst case and can be prevented by adding code to rebalance the tree. Otherwise, the search function has a runtime of O(log n). These runtimes hold true for inserting a course as well. While the average runtime of Binary Search Trees are faster than the others’ average, the others have a chance of having a much faster runtime. The complexity of the code could also be a hinderance if there isn’t enough data to take advantage of the O(log n) runtimes.

# Recommendation

A Hash Table and Binary Search Tree hybrid would bring the benefits of the Binary Search Tree while also getting some of the O(1) runtimes of the Hash Table. I really want to suggest this system, but the amount of data just doesn’t fit the use case for the complexity of the architecture. For ABCU’s specific case, because there is so little data, I suggest using a vector. Even if the amount of courses does increase, I don’t expect they will need the structure intended for thousands of sets of data. The simplicity of the code also allows easy amendments if the structure does need changing (i.e. going from a single array to a muti-dimensional array).