# CS300 Project One

Kaitlyn Elric

Kaitlyn.elric@snhu.edu

Southern New Hampshire University

void displayMenu()

PRINT menu items

“1. Load File Data

2. Print All Courses

3. Print Course Data

9. Exit”

IN user input

Switch cases (userInput)

Case 1:

getFile(file)

PRINT “File data loaded”

break

Case 2:

PRINT courseListSorted()

break

Case 3:

PRINT “Please input course number”

IN user input courseId

printCourse(courseId)

break

Case 9:

Exit

printCourse(courseId)

PRINT “Course number: “ + course.courseId + new line

PRINT “Course name: “ + course.courseName + newline

IF course.prereqs is not null

PRINT “Course prerequisites: “ + course.prereqs + newline

ELSE

PRINT “No prerequisites” + newline

courseListSorted

**//For Vectors**

quick sort (vector courseList, int begin, int end)

SET high low variables

CALCULATE midpoint

SET bid pivot to midpoint

IF begin is greater than or equal to end **//base case**

Break

Mid = partition(list, begin, end)

SET low equal to start and high equal to end

WHILE list at low is less than list at pivot

Increment low

WHILE list at high is greater than list at pivot

Decrement high

IF less than or equal to one elements remaining

Done, return high

Recursively sort low partition, QuickSort(list, begin, mid) (loops until base case)

Recursively sort high partition, QuickSort(list, mid+1, end) (loops until base case)

**//For Hash Table**

courses = vector<Courses> initialized with n elements

n = HashTable::size

HashTable::InOrder()

For each course in HashTable add to Courses

Return Courses

**//For Binary Search Tree**

Binary search trees are already sorted from bottom left, up to the head node, then down to the bottom right.

Return BST

getFile(file)

**//This will be the same for each data structure, time and space complexity is constant**

OPEN file stream

CREATE vector courseList

WHILE file OPEN

GET line

WHILE line length = 0

IF next line length = 0

CLOSE file  
 PRINT “All courses added”

ELSE get next line

WHILE line length greater than 0

SET delimiter to comma “,”

IF line has less than 2 strings

PRINT “Error, incorrect course format”

ELSE

SET struct Course courseID = string 1  
 SET struct Course courseName = string 2  
 WHILE more than 2 strings,

ADD string to struct Course list string prereqs

SET struct Course numPrereqs += 1

ADD Course to vector courseList

insert(Course course)

void insert (Course course)

**//For Hash Table**

CREATE key course.courseID

CREATE new node

IF table at key is empty

Initialize linkedlist with course

ELSE

Append to linked list

Set tail to new node

**//For BST**

IF root is nullptr

root = newnode(course)

ELSE

add node(root, course)

add node (node\* node, Course coursebid)

IF node is greater than root, traverse right

IF right is null

SET right node to node

ELSE IF right is greater than node

SET next node to right node

SET right child to node

ELSE IF right is less than node

SET next node to right node

SET left child to node

IF node is less than root, traverse left

IF left is null

SET left node to node

ELSE IF left is greater than node

SET next node to left node

SET right child to node

ELSE IF left is less than node

SET next node to left node

SET left child to node

**Analysis Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course object** | **Best Time** | **Worst Time** | **Space S(N) =** |
| Vector | O(1) | O(N) | Done in getFile |
| Hash | O(1) | O(1) | N + 2 |
| BST | O(1) | O(1) | N + 3 |
| **Course list sorted** |  |  |  |
| Vector | O(1) | O(N) | N + 5 |
| Hash | O(1) | O(N) | N + 1 |
| BST | O(1) | O(1) | N |
| Read file | O(1) | O(N) | N + 4 |

I think that a hash table is the best way to sort courses because the courseId simplifies the process and makes searching easier and more efficient. While sorting a binary search tree is unnecessary as it is presorted due to the structure of the data type, the hash table takes up less space as a course object. The ability to break the table up in half to find a key, then in half again, saves on runtime, is more efficient than the runtime of a vector, and uses less space than a vector as well. Of course, the best time for each course list sorting would be if the list was already sorted, giving the binary search tree an advantage in both space and runtime over vector and hash structures. The worst runtime for sorting would be for the vector and hash data structures both, as they would have to break themselves up and sort data as needed in order to put the course list in ascending alphanumerical order. The course structure for the hash table takes the least amount of time in the worst case scenario, however, as it is a simple if this, then that structure that requires little computer effort. While technically the binary search tree has the same best and worst case times as the hash key, it’s if else set up is more complex and would take longer to adjust the nodes if the root is not null. The vector is being created in the getFile method, so it would have the same best and worst case scenarios as getFile. If there is only one line, it only iterates once. If there are N lines, it iterates N times into the vector. Using if else statements saves on runtime complexity and helps programs run faster. Overall, the hash table uses less code and space, and it takes up less time than the other two structures. Therefore, I recommend using the hash table data structure in the ABC course program.