Pseudocode and Runtime Analysis

**Vector Pseudocode:**

Void loadCsvFile(vector<Course> courses){

DECLARE vector<string> lineData

OPEN file

FOR each line in file

FOR each character in line

IF character equals ‘,’

APPEND string from previous ‘,’ to current ‘,’ to vector lineData

IF vector lineData size is less than 2

DISPLAY file corrupted

EXIT program

ELSE

DECLARE string courseId = first element of lineData

DECLARE string courseTitle = second element of lineData

DECLARE vector<strings> prerequisites = remaining elements of lineData

CREATE Course object from courseId, courseTitle, and prerequisites

APPEND Course to vector courses

}

Void validateCourses(vector<Course> courses){

FOR each course

IF course has prerequisites

FOR each prerequisite

Check that prerequisite also exists as a course

IF a prerequisite is not found

DISPLAY file corrupted

EXIT program

}

Void searchCourse(vector<Course> courses, string courseId){

FOR each course

IF course matches courseId

DISPLAY course information

FOR each prerequisite in course

DISPLAY each prerequisite

IF course not found

DISPLAY course not found

}

int partition(vector<Course> courses, int begin, int end){

SET low equals begin

SET high equals end

CALCULATE pivot equals (low plus high) divided by 2

WHILE not done

WHILE courses[low].courseID is less than courses[pivot].courseID

I NCREMENT low

WHILE courses[high].courseID is greater than courses[pivot].courseID

DECREMENT high

IF courses[low].courseID is less than courses[high].courseID

SWAP courses[low] and courses[high]

ELSE done

RETURN high

}

Void quicksort(vector<Course> courses, int begin, int end){

SET mid equals 0

SET low equals begin

SET high equals end

IF low greater than or equal to high

RETURN

ELSE

mid equals partition(courses, low, high) function

quickSort(bids, low, mid)

quickSort(bids, mid+1, high

}

Void printCourseList(vector<Course> courses){

IF courses not sorted

quickSort(courses, begin, end)

FOR each course in courses

Display course information

}

**Hash Table Pseudocode:**

Void loadCsvFile(HashTable courses) {

DECLARE vector<string> lineData

OPEN file

FOR each line in file

FOR each character in line

IF character equals ‘,’

APPEND string from previous ‘,’ to current ‘,’ to vector lineData

IF vector lineData size is less than 2

DISPLAY file corrupted

EXIT program

ELSE

DECLARE string courseId = first element of lineData

DECLARE string courseTitle = second element of lineData

DECLARE vector<strings> prerequisites = remaining elements of lineData

CREATE Course object from courseId, courseTitle, and prerequisites

GET bucket from courseID

IF bucket is empty

INSERT Course into bucket

ELSE

TRAVERSE to last item in bucket

INSERT Course as new last item in bucket

}

Void validateCourses(HashTable courses) {

FOR each bucket in courses

FOR each Course object in bucket

IF course has prerequisites

FOR each prerequisite

Check that prerequisite matches an existing bucket in courses

IF bucket empty

DISPLAY file corrupted

EXIT program

}

Void searchCourse(HashTable courses, string courseId) {

GET bucket corresponding to courseId

FOR each Course object in bucket

IF course matches courseId

DISPLAY course information

FOR each prerequisite in course

DISPLAY each prerequisite

RETURN

IF course not found

DISPLAY course not found

}

Void printCourseList(HashTable courses){

FOR each bucket

IF bucket not empty

DISPLAY course information

}

**Binary Search Tree Pseudocode:**

Void loadCsvFile(BinarySearchTree courses) {

DECLARE vector<string> lineData

OPEN file

FOR each line in file

FOR each character in line

IF character equals ‘,’

APPEND string from previous ‘,’ to current ‘,’ to vector lineData

IF vector lineData size is less than 2

DISPLAY file corrupted

EXIT program

ELSE

DECLARE string courseId = first element of lineData

DECLARE string courseTitle = second element of lineData

DECLARE vector<strings> prerequisites = remaining elements of lineData

CREATE Course object from courseId, courseTitle, and prerequisites

INSERT Course into Courses using courseId as key

}

Void validateCourses(Course\* course) {

IF Course not equal nullptr

validateCourses(Course->left)

IF Course has prerequisites

FOR each prerequisite

searchCourse(prerequisite)

IF not found

DISPLAY file corrupted

EXIT program

validateCourses(Course->right)

}

Void printCourseList(BinarySearchTree courses){

inOrder(root)

}

Void inOrder(Course) {

IF Course not equal nullptr

inOrder(Course->left)

DISPLAY Course information

inOrder(Course->right)

}

Void searchCourse(BinarySearchTree courses, string courseId) {

SET current node to root

WHILE current node not equal to null pointer

IF current node course Id matches courseId

Display course information

ELSE IF current node course Id is less than course Id

current node equals current node->left

ELSE

current node equals current node->right

}

**Menu:**

Void printMenu(){

DO WHILE True

DISPLAY 1. Load file

DISPLAY 2 .Print Course List

DISPLAY 3. Search for course

DISPLAY 4. Exit program

GET user selection

IF user selection is 1

loadCsvFile() function

ELSE IF user selection is 2

printCourseList() function

ELSE IF user selection is 3

searchCourse() function

ELSE IF user selection is 4

EXIT program

Worst Case Run Time

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # times execution | Total Cost |
| **Reading the File** | | | |
| For each Line in file | 1 | n | n |
| For each character in line | 1 | n | n |
| If character = ‘,’ | 1 | n | n |
| Append string to lineData | 1 | n | n |
| Else | 1 | n | n |
| **Create Object** | | | |
| Declare course id | 1 | 1 | 1 |
| Declare course name | 1 | 1 | 1 |
| Declare prerequisites | 1 | 1 | 1 |
| Create Course Object | 1 | 1 | 1 |
| **Append Course Object to courses** |  |  |  |
| Vector | 1 | 1 | 1 |
| Hash Table | 8 | n2 | n2 + 8 |
| Binary Search Tree | 8 | n | n + 8 |
| **Total Cost** | | |  |
| Vector | | | 5n + 5 |
| Hash Table | | | n3 + 12 |
| Binary Search Tree | | | n2 + 12 |
| **Runtime** | | | |
| Vector | | | O(n) |
| Hash Table | | | O(n3 ) |
| Binary Search Tree | | | O(n2 ) |

The advantage of using a vector to store the information is that it is easy to implement, and easy to sort. The disadvantage of using a vector is that if the list becomes large enough, then search times will start to become slower because it has to iterate through the entire list. The advantage of using a hash table is searching for a course will be very quick, sometimes even O(1). However, because a hash table is difficult to sort without drastically increasing run times, the requirement of printing all courses alphanumerically makes this not a viable option. A binary search tree’s advantage is that searching for a course typically has a run time of O(log2n) and inserting new objects into the tree automatically sorts them. The disadvantage of binary search trees is that they are much more difficult to implement than a vector, and if the data being read is already sorted or nearly sorted, then the run time difference won’t be very much between a vector and a binary search tree when searching.

For my code, if the data is already sorted or nearly sorted, I will use a vector. If it is in random order, I will use a binary search tree. If the data is already sorted, the increased run time of using a binary search tree for loading and creating the objects make them not a viable option. Hash tables are not an option because they aren’t sortable, and one of the requirements is to be able to print the course list in alphanumerical order.