//Pseudocode to open file, read data from file, parse each line, and check for file format

//function Open file

Void OpenReadCloseFile(string filePath, vector<string>& originalCourses) {

Open filePath as ifstream object

If filePath is not open then

Output “could not open file”

Return -1

While file is not at end of file

Get line from the file

Append the row to originalCourses

Close the file

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open filepath as ifstream object | 1 | 1 | 1 |
| If filePath is not open | 1 | 1 | 1 |
| Output “could not open file” | 1 | 1 | 1 |
| Return -1 | 1 | 1 | 1 |
| While file is not at end of file | 1 | N + 1 | N +1 |
| Get line from the file | 1 | N | N |
| Append the row to originalCourses | 1 | N | N |
| Close the file | 1 | 1 | 1 |
| Total Cost | | | 3N + 6 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis

|  |  |
| --- | --- |
| Code | Total Cost |
| ifstream object | 1 |
| Get line | 1 |
| Append the row to originalCourses | N |
| Total Cost | N + 2 |
| Auxiliary space complexity | O(N) |

//function to parse

Void ParseFile(vector<string>& originalCourses, vector<vector<string>>& parsedCourses, char delimiter) {

For each row in originalCourses

Reset column count to 0

While not at end of line

Get line and split by delimiter

Append to parsedCourses[row][column]

Increment column count by 1

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in original Courses | 1 | N | N |
| Reset column count to 0 | 1 | N | N |
| While not at end of line | 1 | M +1 | M + 1 |
| Get line and split by delimiter | 1 | M | M |
| Append to parsedCourses[row][column] | 1 | M | M |
| Increment column count by 1 | 1 | M | M |
| Total Cost | | | 2N \* 3M + 1 |
| Runtime | | | O(N\*M) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Column count | | 1 |
| Get line | | 1 |
| Append to parsedCourses[row][column] | | N\*M |
| Total Cost | | N\*M + 2 |
| Auxiliary space complexity | O(N\*M) | | |

//function to ensure there are at least two parameters per line

Void ParameterCheck(vector<vector<string>>& parsedCourses) {

For each row in parsedCourses

Get size of parsedCourses[row]

If size is less than 2

Display row that has less than two parameters

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in parsedCourses | 1 | N | N |
| Get size of parsedCourses[row] | 1 | N | N |
| If size is less than 2 | 1 | N | N |
| Display row that has less than two parameters | 1 | N | N |
| Total Cost | | | 4N |
| Runtime | | | O(N) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Variable to hold size | | 1 |
| Total Cost | 1 | | |
| Auxiliary space complexity | O(1) | | |

//function to make sure each prerequisite has matching course in file

Void PrerequisiteCheck(vector<vector<string>>& parsedCourses) {

For each row in parsedCourses

If size of parsedCourses[row] is greater than 2

For each column in parsed courses greater than 2

Search each row at first column for that course

If a matching course is found

Continue

Else

Display prerequisite course that doesn’t have matching course

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in parsedCourses | 1 | N | N |
| If size of parsedCourses[row] is greater than 2 | 1 | N | N |
| For each column in parsed courses greater than 2 | 1 | M | M |
| Search each row at first column for that course | 1 | N | N |
| If a matching course is found | 1 | N | N |
| Continue | 1 | N | N |
| Else display prerequisite courses that doesn’t have matching course | 1 | N | N |
| Total Cost | | | N \* M + 4N |
| Runtime | | | O(N\*M) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Variable to hold current course | | 1 |
| Total Cost | 1 | | |
| Auxiliary space complexity | O(1) | | |

//create course objects

Course objects will have: course ID, name of course, a vector for prerequisites

Void CreateCoursesVector(vector<vector<string>>& parsedCourses, vector<Course>& courseObjects) {

For each row

Create a Course Object course

Course’s ID = parsedCourses[row][0]

Course’s name = parsedCourses[row][1]

If row size is greater than 2

For i = 2 and i < row’s size

Append parsedCourses[row][i] to vector of prerequisites

Append course to vector courseObjects

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row | 1 | N | N |
| Create a course object course | 1 | N | N |
| Course’s ID = parsedCourses[row][0] | 1 | N | N |
| Course’s name = parsedCourses[row][1] | 1 | N | N |
| If row size is greater than 2 | 1 | N | N |
| For i = 2 and i < row’s size | 1 | M | M |
| Append parsedCourses[row][i] to vector of prerequisites | 1 | M | M |
| Append course to vector courseObjects | 1 | N | N |
| Total Cost | | | 6N \* M |
| Runtime | | | O(N\*M) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Create a course object course | | N |
| Append course to vector courseObjects | | N |
| Total Cost | 2N | | |
| Auxiliary space | O(N) | | |

//search for and print out course information and prerequisites

Void PrintCourseInformationVector(vector<Course> courseObjects, string courseNumber) {

For all courses in courseObjects

If the course’s ID is the same as courseNumber

Print out the course information

For each prerequisite in the course’s prerequisite vector

Print out the prerequisite

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all courses in courseObjects | 1 | N | N |
| If the course’s ID is the same as courseNumber | 1 | N | N |
| Print out the course information | 1 | 1 | 1 |
| For each prerequisite in the course’s prerequisite vector | 1 | N | N |
| Print out the prerequisite | 1 | N | N |
| Total Cost | | | 4N + 1 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Auxiliary Space Complexity | | O(1) | |

//function to print all courses

Void PrintAllVector (vector<Course> courseObjects) {

For all rows in courseObjects

Print courseObject

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all rows in courseObjects | 1 | N | N |
| Print courseObject | 1 | N | N |
| Total Cost | | | 2N |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Auxiliary Space Complexity | | O(1) | |

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

int Partition(vector<Courses> courseObjects, int low, int high) {

Determine midpoint of the list and set it to mid

Set pivot value to list at mid’s courseID

While not done

While the pivot value is greater than courseID value at low

Increment low

While the pivot value is less than courseID value at high

Decrement high

If low is greater than or equal to high

Set done to true

Else

Swap values at low and high

Increment low

Decrement high

Return high

}

Runtime Analysis (not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Determine midpoint of the list and set it to mid | 1 | 1 | 1 |
| Set pivot value to list at mid’s courseID | 1 | 1 | 1 |
| While not done | 1 | ? | ? |
| While the pivot value is greater than courseID value at low | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| Increment low | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| While the pivot value is less than courseID value at high | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| Decrement high | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| If low is greater than or equal to high | 1 | 1 | 1 |
| Set done to true | 1 | 1 | 1 |
| Else  Swap values at low and high | 1 | 1 | 1 |
| Increment low | 1 | 1 | 1 |
| Decrement high | 1 | 1 | 1 |
| Return high | 1 | 1 | 1 |
| Total Cost | | | 2N+10 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis (not required)

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Midpoint | | 1 | |
| Pivot | | 1 | |
| Auxiliary Space Complexity | | O(1) | |

Void QuickSort(vector<Courses> courseObjects, int low, int high) {

If low is greater than or equal to high

Return

Set mid equal to index value returned by call to Partition(courseObjects, low, high)

Call QuickSort(courseObjects, low, mid)

Call QuickSort(courseObjects, mid + 1, high)

}

Runtime Analysis (not required)

In the worst case: T(n) = T(n-1) + T(1) + n

T(n-1) = T(n-2) + T(1) + n-1

T(n-2) = T(n-3) + T(1) + n – 1 – 1

Back substitution:

Substituting T(n-1) into T(n)

T(n) = [T(n-2) + T(1) + n-1] + T(1) + n

= T(n-2) + 1 + n – 1 + 1 + n

= T(n-2) + 2n + 1

Substituting T(n-2) into T(n)

T(n) = [T(n-3) + T(1) + n – 1 – 1] + 2n +1

=T(n-3) + 1 + n -2 + 2n + 1

= T(n-3) + 3n

…

T(n) = T(n-k) + kn

When n = k:

T(n) = T(0) + n2

T(n) = 1 + n2

T(n) = n2

Auxiliary Space Complexity (not required):

In the worst case, quicksort will make n recursive calls, therefore the height of a recursive call tree will be n. Therefore, the auxiliary space complexity is O(n)

Void PrintSortedVector(vector<Courses> courseObjects) {

Call QuickSort(courseObjects, low, high)

For all objects in courseObjects

Print course object

}

Runtime Analysis (not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Call QuickSort(courseObjects, low, high) | O(N2) | 1 | O(N2) |
| For all objects in courseObjects | 1 | N | N |
| Print course object | 1 | N | N |
| Total Cost | | | N2 + 2N |
| Runtime | | | O(N2) |