Below you will find the big o analysis and the pseudocode for three structures, vectors, hash tables, and binary search trees, to implement code for a school’s classes that they offer. For the vector structure we see that to open a file we have a big o of n^3 which isn’t the best run time analysis but is the same for the other structures. The vector big o for creating objects is n which is a very good runtime analysis that is only comparable to hash tables. For the vector pseudocode, there are quite a few more lines of code than required for other structures. For hash tables, it has the same disadvantages as the runtime analysis for opening the file on n^3. The hash table advantages are that to create the course objects it only takes a big o of n and far fewer executable lines of code making it the best option. For binary search tree, the disadvantages are the run time analysis of opening the file which is n^3 and creating the objects since it is a recursive search, which is nlogn, which is a terrible run time analysis. The structure I would choose to implement my code in is the hash table structure. It has the best runtime analysis for the two functions we calculated and includes the least amount of redundant code in its pseudocode.

**// Vector open file**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **inFile = P&G filename from user** | 1 | 1 | 1 |
| **open(inFile)** | 1 | 1 | 1 |
| **IF (inputFile open failed)** | 1 | 1 | 1 |
| **Display error** | 1 | 1 | 1 |
| **Return error code** | 1 | 1 | 1 |
| **IF (loadFileData(inputFile, fileData))** | 1 | 1 | 1 |
| **displayFileData(fileData)** | 1 | 1 | 1 |
| **inputFile.close** | 1 | 1 | 1 |
| **WHILE file is not EOF** | 1 | N | N |
| **Declare empty tempRow (couseRow object)** | 1 | 1 | 1 |
| **getline from inputFile and Store in filedata (string)** | 1 | 1 | 1 |
| **IF fileData not empty** | 1 | N | N |
| **Instantiate stringstream object – ss(fileData)** | 1 | 1 | 1 |
| **IF (get data and store in token, parsen on ‘,’)** | 1 | N | N |
| **tempRow.courseNum = token** | 1 | N | N |
| **IF (get data and store in token, parsed on ‘,’)** | 1 | N | N |
| **tempRow.name = token** | 1 | N | N |
| **while (get data and store token, parsed on ‘,’ is not null)** | 1 | N | N |
| **tempRow.preReq[i] = token** | 1 | N | N |
| **I incremented by 1** | 1 | N | N |
| **IF tempRow.courseNum and tempRow.name are not empty** | 1 | N | N |
| **Push\_back(tempRow) to add to fileData vector** | 1 | 1 | 1 |
| **for (i=o; i < fileData.size: i plus 1)** | 1 | N | N |
| **validation vector equals fileData[i].courseNum** | 1 | N | N |
| **for (j=0; j < fileData.size; j plus 1)** | 1 | N | N |
| **for (k=0; k < preReq.size; k plus 1)** | 1 | N | N^2 |
| **for all data in validation vector** | 1 | N | N^3 |
| **if fileData[j].preReq[k] does not equal a validation vector entry** | 1 | N | N^3 |
| **delete fileData[j].preReq[k]** | 1 | N | N^3 |
| **Total Cost** | | | 3n^3 + n^2 + 13n + 12 |
| **Runtime** | | | O(n^3) |

**// Vector create objects**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **WHILE file is not EOF** | 1 | N | N |
| **Declare empty tempRow (couseRow object)** | 1 | 1 | 1 |
| **getline from inputFile and Store in filedata (string)** | 1 | 1 | 1 |
| **IF fileData not empty** | 1 | N | N |
| **Instantiate stringstream object – ss(fileData)** | 1 | 1 | 1 |
| **IF (get data and store in token, parsen on ‘,’)** | 1 | N | N |
| **tempRow.courseNum = token** | 1 | N | N |
| **IF (get data and store in token, parsed on ‘,’)** | 1 | N | N |
| **tempRow.name = token** | 1 | N | N |
| **while (get data and store token, parsed on ‘,’ is not null)** | 1 | N | N |
| **tempRow.preReq[i] = token** | 1 | N | N |
| **I incremented by 1** | 1 | N | N |
| **Total Cost** | | | 9n + 3 |
| **Runtime** | | | O(n) |

**// HashTable open file**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **inFile = P&G filename from user** | 1 | 1 | 1 |
| **open(inFile)** | 1 | 1 | 1 |
| **IF (inputFile open failed)** | 1 | 1 | 1 |
| **Display error** | 1 | 1 | 1 |
| **Return error code** | 1 | 1 | 1 |
| **IF (loadFileData(inputFile, fileData))** | 1 | 1 | 1 |
| **displayFileData(fileData)** | 1 | 1 | 1 |
| **inputFile.close** | 1 | 1 | 1 |
| **WHILE file is not EOF** | 1 | N | N |
| **Declare empty tempRow (couseRow object)** | 1 | 1 | 1 |
| **getline from inputFile and Store in filedata (string)** | 1 | 1 | 1 |
| **IF fileData not empty** | 1 | N | N |
| **Instantiate stringstream object – ss(fileData)** | 1 | 1 | 1 |
| **IF (get data and store in token, parsen on ‘,’)** | 1 | N | N |
| **tempRow.courseNum = token** | 1 | N | N |
| **IF (get data and store in token, parsed on ‘,’)** | 1 | N | N |
| **tempRow.name = token** | 1 | N | N |
| **while (get data and store token, parsed on ‘,’ is not null)** | 1 | N | N |
| **tempRow.preReq[i] = token** | 1 | N | N |
| **I incremented by 1** | 1 | N | N |
| **IF tempRow.courseNum and tempRow.name are not empty** | 1 | N | N |
| **Push\_back(tempRow) to add to fileData vector** | 1 | 1 | 1 |
| **for (i=o; i < fileData.size: i plus 1)** | 1 | N | N |
| **validation vector equals fileData[i].courseNum** | 1 | N | N |
| **for (j=0; j < fileData.size; j plus 1)** | 1 | N | N |
| **for (k=0; k < preReq.size; k plus 1)** | 1 | N | N^2 |
| **for all data in validation vector** | 1 | N | N^3 |
| **if fileData[j].preReq[k] does not equal a validation vector entry** | 1 | N | N^3 |
| **delete fileData[j].preReq[k]** | 1 | N | N^3 |
| **Total Cost** | | | 3n^3 + n^2 + 13n + 12 |
| **Runtime** | | | O(n^3) |

**// Hash Table create objects**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Declare an unsigned key equal to a given hashes fileData’s courseNum** | 1 | 1 | 1 |
| **Declare a node oldNode equal to a nodes key** | 1 | 1 | 1 |
| **If (old node is equal to a null pointer)** | 1 | 1 | 1 |
| **Declare a node newNode equal to a node with the given key position** | 1 | 1 | 1 |
| **Insert the beginning node + the newNode** | 1 | 1 | 1 |
| **Else If (oldNode’s key is equal to UINT\_MAX)** | 1 | 1 | 1 |
| **Assign oldNode’s key to the given key** | 1 | 1 | 1 |
| **Assign oldNode’s fileData to the given fileData** | 1 | 1 | 1 |
| **Assign oldNode’s next entry as null pointer** | 1 | 1 | 1 |
| **Else** | 1 | 1 | 1 |
| **While (oldNode’s next entry is not a null pointer)** | 1 | N | N |
| **Assign oldNode equal to oldNode’s next entry** | 1 | N | N |
| **Set oldNodes’s next entry to a new node** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 11 |
| **Runtime** | | | O(n) |

**// Binary Search Tree open file**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **inFile = P&G filename from user** | 1 | 1 | 1 |
| **open(inFile)** | 1 | 1 | 1 |
| **IF (inputFile open failed)** | 1 | 1 | 1 |
| **Display error** | 1 | 1 | 1 |
| **Return error code** | 1 | 1 | 1 |
| **IF (loadFileData(inputFile, fileData))** | 1 | 1 | 1 |
| **displayFileData(fileData)** | 1 | 1 | 1 |
| **inputFile.close** | 1 | 1 | 1 |
| **WHILE file is not EOF** | 1 | N | N |
| **Declare empty tempRow (couseRow object)** | 1 | 1 | 1 |
| **getline from inputFile and Store in filedata (string)** | 1 | 1 | 1 |
| **IF fileData not empty** | 1 | N | N |
| **Instantiate stringstream object – ss(fileData)** | 1 | 1 | 1 |
| **IF (get data and store in token, parsen on ‘,’)** | 1 | N | N |
| **tempRow.courseNum = token** | 1 | N | N |
| **IF (get data and store in token, parsed on ‘,’)** | 1 | N | N |
| **tempRow.name = token** | 1 | N | N |
| **while (get data and store token, parsed on ‘,’ is not null)** | 1 | N | N |
| **tempRow.preReq[i] = token** | 1 | N | N |
| **I incremented by 1** | 1 | N | N |
| **IF tempRow.courseNum and tempRow.name are not empty** | 1 | N | N |
| **Push\_back(tempRow) to add to fileData vector** | 1 | 1 | 1 |
| **for (i=o; i < fileData.size: i plus 1)** | 1 | N | N |
| **validation vector equals fileData[i].courseNum** | 1 | N | N |
| **for (j=0; j < fileData.size; j plus 1)** | 1 | N | N |
| **for (k=0; k < preReq.size; k plus 1)** | 1 | N | N^2 |
| **for all data in validation vector** | 1 | N | N^3 |
| **if fileData[j].preReq[k] does not equal a validation vector entry** | 1 | N | N^3 |
| **delete fileData[j].preReq[k]** | 1 | N | N^3 |
| **Total Cost** | | | 3n^3 + n^2 + 13n + 12 |
| **Runtime** | | | O(n^3) |

**// Binary Search Tree creates objects**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **if (node is larger than zero)** | 1 | 1 | 1 |
| **if (the left node equal to a null pointer)** | 1 | n | nlogn |
| **set the left node equal to a new node** | 1 | 1 | 1 |
| **else** | 1 | 1 | 1 |
| **addNode(left node, fileData)** | 1 | 1 | 1 |
| **else** | 1 | 1 | 1 |
| **if (the right node is equal to a null pointer)** | 1 | n | nlogn |
| **set the right node equal to a new node** | 1 | 1 | 1 |
| **else** | 1 | 1 | 1 |
| **addNode(right node, fileData)** | 1 | 1 | 1 |
| **Declare node equal to a new Node** | 1 | 1 | 1 |
| **If (root is equal to a a null pointer)** | 1 | 1 | 1 |
| **Set root equal to node** | 1 | 1 | 1 |
| **else** | 1 | 1 | 1 |
| **point this to addNode passing the root and fileData** | 1 | N | N |
| **Total Cost** | | | n + 2nlogn + 12 |
| **Runtime** | | | O(nlogn) |

**// Vector pseudocode**

Pseudocode for opening a file, reading the data, parsing each line, and checking the format

Bool openFile(ifstream& inputFile)

inFile = P&G filename from user

open(inFile)

IF (inputFile open failed)

Display error

Return error code

Endif

IF (loadFileData(inputFile, fileData))

displayFileData(fileData)

endif

inputFile.close

end openFile

Bool readParseValidate (ifstream& inputFile, courseRow vector fileData, vector valdation)

WHILE file is not EOF

Declare empty tempRow (couseRow object)

getline from inputFile and Store in fileData (string)

IF fileData not empty

Instantiate stringstream object – ss(fileData)

IF (get data and store in token, parsen on ‘,’)

tempRow.courseNum = token

IF (get data and store in token, parsed on ‘,’)

tempRow.name = token

while (get data and store token, parsed on ‘,’ is not null)

tempRow.preReq[i] = token

I incremented by 1

End while

endif

endif

endif

IF tempRow.courseNum and tempRow.name are not empty

Push\_back(tempRow) to add to fileData vector

Endif

End while

for (i=o; i < fileData.size: i plus 1)

validation vector equals fileData[i].courseNum

end for

for (j=0; j < fileData.size; j plus 1)

for (k=0; k < preReq.size; k plus 1)

for all data in validation vector

if fileData[j].preReq[k] does not equal a validation vector entry

delete fileData[j].preReq[k]

endif

end for

end for

end for

end readParseValidate

Pseudocode to create course objects and store them

Bool createObjects (ifstream& inputFile, courseRow vector fileData, vector preReq)

WHILE file is not EOF

Declare empty tempRow (courseRow object)

getline from inputFile and Store in filedata (string)

IF fileData not empty

Instantiate stringstream object – ss(fileData)

IF (get data and store in token, parsen on ‘,’)

tempRow.courseNum = token

IF (get data and store in token, parsed on ‘,’)

tempRow.name = token

while (get data and store token, parsed on ‘,’ is not null)

tempRow.preReq[i] = token

I incremented by 1

End while

endif

endif

endif

end while

end createObjects

Pseudocode for searching and printing a sample course

Void printCourseInfo (courseRow vector fileData, string courseNum, vector preReq)

For (i=o; i < fileData.size; i incremented by 1)

If fileData.courseNum at i is equal exactly to the given courseNum

Display fileData.courseNum at i and fileData.name at i

For (j=0; j < preReq.size; j incremented by 1)

If fileData[i].preReq[j] is not empty

Display fileData[i].preReq[j]

Endif

End for

Endif

End for

End printCourseInfo

Pseudocode for printing a menu

void displayMenu(int argc, char\* argv[1], courseRow vector fileData, vector preReq)

Declare a string for the csvPath

Switch (argc)

Case 2:

Set csvPath to equal argv[1]

Break

Default:

Set csvPath to equal the file with the course information

end switch

define a vector to hold all the data named fileData

Declare an interger named choice to equal 0

While (choice!=9)

Display Menu:

Display 1. Load data

Display 2. Print course list

Display 3. Print course info

Display 9. Exit

Display enter choice:

Read the users choice variable

Switch (choice)

Case 1:

Set fileData equal to readParseValidate(csvPath)

Break

Case 2:

For (i=o; i < fileData.size; i incremented by 1)

Display fileData.courseNum at i and fileData.name at i

For (j=0; j < preReq.size; j incremented by 1)

If fileData[i].preReq[j] is not empty

Display fileData[i].preReq[j]

Endif

End for

End for

Break

Case 3:

Display enter desired course number

Read in the courseNum variable from the user

printCourseInfo(fileData, courseNum, preReq)

break

end switch

end while

display Good bye

end displayMenu

Pseudocode for alphanumeric course sort

Int partition (coursRow vector fileData, int begin, int end)

Declare integer low as the beginning of vector fileData

Declare interger high as the end of vector fileData

Declare interge pivot as begin + (end – begin) / 2

Declare boolean done as false

While not done

While (fileData[low] < fileData[pivot])

Increment low by 1

EndWhile

While (fileData[pivot] < fileData[high]

Decrement high by 1

EndWhile

If (low is less than or equal to high)

Declare done as true

Else

Swap fileData[low] with fileData[high]

Increment low by 1

Decrement high by 1

endIF

EndWhile

Return high

end partition

Void quicksort (courseRow vector fileData, int begin, int end)

Declare integer mid as 0

If (begin is less than or equal to end)

Return

endIf

declare mid equal to partition(vector fileData, int begin, int end)

call function quickSort(vector fileData, int begin, int mid)

call function quickSort(vector fileData, int mid plus 1, int end)

end quicksort

**// Hash Table pseudocode**

Pseudocode for opening a file, reading the data, parsing each line, and checking the format

Bool openFile(ifstream& inputFile)

inFile = P&G filename from user

open(inFile)

IF (inputFile open failed)

Display error

Return error code

Endif

IF (loadFileData(inputFile, fileData))

displayFileData(fileData)

endif

inputFile.close

end openFile

Bool readParseValidate (ifstream& inputFile, courseRow vector fileData, vector valdation, vector preReq)

WHILE file is not EOF

Declare empty tempRow(courseRow object)

getline from inputFile and Store in filedata (string)

IF fileData not empty

Instantiate stringstream object – ss(fileData)

IF (get data and store in token, parsed on ‘,’)

tempRow.courseNum = token

IF (get data and store in token, parsed on ‘,’)

tempRow.name = token

for (int i = 0; get data and store token, parsed on ‘,’ not empty; i++)

tempRow.preReq[i] = token

endFor

endif

endif

endif

IF tempRow.courseNum and tempRow.name are not empty

Push\_back(tempRow) to add to fileData vector

Endif

End while

for (i=o; i < fileData.size: i plus 1)

validation vector equals fileData[i].courseNum

end for

for (j=0; j < fileData.size; j plus 1)

for all data in validation vector

for (int k =0; for preReq[k] not empty; k++)

if fileData[j].preReq[k] does not equal a validation vector entry

delete fileData[j].preReq[k]

endif

end for

end for

end readParseValidate

Pseudocode to create course objects and store them

void createObjects (courseRow vector fileData)

Declare an unsigned key equal to a given hashes fileData’s courseNum

Declare a node oldNode equal to a nodes key

If (old node is equal to a null pointer)

Declare a node newNode equal to a node with the given key position

Insert the beginning node + the newNode

endIf

Else If (oldNode’s key is equal to UINT\_MAX)

Assign oldNode’s key to the given key

Assign oldNode’s fileData to the given fileData

Assign oldNode’s next entry as null pointer

EndelseIf

Else

While (oldNode’s next entry is not a null poiter)

Assign oldNode equal to oldNode’s next entry

endWhile

Set oldNodes’s next entry to a new node

endElse

end createObjects

Pseudocode for searching and printing a sample course

Void printCourseInfo ()

For (iter equal to the beginning of the fileData; iter does not equal the end of the fileData; increment the iter by 1)

If (iter’s key is not equal to the UINT\_MAX)

Display the key, courseNum, name, and don’t end line

For (i=0; iter’s preReq[i] is not empty; i incremented by 1)

Display the preReq[i]

endFor

endline

Declare a new node equal to the next iter

While (node does not equal a null pointer)

Display the key, courseNum, name, and don’t end line

For (i=0; iter’s preReq[i] is not empty; i incremented by 1)

Display the preReq[i]

endFor

Set node equal to the next node

EndWhile

endIf

endFor

end printCourseInfo

Pseudocode for printing a menu

void displayMenu(int argc, char\* argv[1], courseRow vector fileData, vector preReq)

Declare a string for the csvPath

Switch (argc)

Case 2:

Set csvPath to equal argv[1]

Break

Default:

Set csvPath to equal the file with the course information

end switch

define a vector to hold all the data named fileData

Declare an interger named choice to equal 0

While (choice!=9)

Display Menu:

Display 1. Load data

Display 2. Print course list

Display 3. Print course info

Display 9. Exit

Display enter choice:

Read the users choice variable

Switch (choice)

Case 1:

Set fileData equal to readParseValidate(csvPath)

Break

Case 2:

printCourseInfo()

break

Case 3:

Display enter desired course number

Read in the courseNum variable from the user

Declare an unsigned key equal to the given hash with the courseNum

Declare a new node at the given key

If (node does not equal a null pointer and the node’s key does not equal uint\_max and the node’s courseNum is found)

display the node’s fileData

endIf

if (node equals a null pointer or node’s key equals uint\_max)

return fileData

endIf

while (node does not equal a null pointer)

If (node does not equal a null pointer and the node’s key does not equal uint\_max and the node’s courseNum is found)

display node’s fileData

endIf

set node equal to the next node

end while

break

end switch

end while

display Good bye

end displayMenu

Pseudocode for alphanumeric course sort

Int partition (coursRow vector fileData, int begin, int end)

Declare integer low as the beginning of vector fileData

Declare interger high as the end of vector fileData

Declare interge pivot as begin + (end – begin) / 2

Declare boolean done as false

While not done

While (fileData[low] < fileData[pivot])

Increment low by 1

EndWhile

While (fileData[pivot] < fileData[high]

Decrement high by 1

EndWhile

If (low is less than or equal to high)

Declare done as true

Else

Swap fileData[low] with fileData[high]

Increment low by 1

Decrement high by 1

endIF

EndWhile

Return high

end partition

Void quicksort (courseRow vector fileData, int begin, int end)

Declare integer mid as 0

If (begin is less than or equal to end)

Return

endIf

declare mid equal to partition(vector fileData, int begin, int end)

call function quickSort(vector fileData, int begin, int mid)

call function quickSort(vector fileData, int mid plus 1, int end)

end quickSort

**// Binary Search Tree pseudocode**

Pseudocode for opening a file, reading the data, parsing each line, and checking the format

Bool openFile(ifstream& inputFile)

inFile = P&G filename from user

open(inFile)

IF (inputFile open failed)

Display error

Return error code

Endif

IF (loadFileData(inputFile, fileData))

displayFileData(fileData)

endif

inputFile.close

end openFile

Bool readParseValidate (ifstream& inputFile, courseRow vector fileData, vector valdation, vector preReq)

WHILE file is not EOF

Declare empty tempRow (courseRow object)

getline from inputFile and Store in filedata (string)

IF fileData not empty

Instantiate stringstream object – ss(fileData)

IF (get data and store in token, parsed on ‘,’)

tempRow.courseNum = token

IF (get data and store in token, parsed on ‘,’)

tempRow.name = token

for (int i = 0; get data and store token, parsed on ‘,’ not empty; i++)

tempRow.preReq[i] = token

endFor

endif

endif

endif

IF tempRow.courseNum and tempRow.name are not empty

Push\_back(tempRow) to add to fileData vector

Endif

End while

for (i=o; i < fileData.size: i plus 1)

validation vector equals fileData[i].courseNum

end for

for (j=0; j < fileData.size; j plus 1)

for all data in validation vector

for (int k =0; for preReq[k] not empty; k++)

if fileData[j].preReq[k] does not equal a validation vector entry

delete fileData[j].preReq[k]

endif

end for

end for

end readParseValidate

Pseudocode to create course objects and store them

void addNode(Nod\* node, courseRow vector fileData)

if (node is larger than zero)

if (the left node equal to a null pointer)

set the left node equal to a new node

endIf

else

addNode(left node, fileData)

endElse

endIf

else

if (the right node is equal to a null pointer)

set the right node equal to a new node

endIf

else

addNode(right node, fileData)

endElse

endElse

end addNode

void createObjects (courseRow vector fileData)

Declare node equal to a new Node

If (root is equal to a a null pointer)

Set root equal to node

endIF

else

point this to addNode passing the root and fileData

endElse

end createObjects

Pseudocode for searching and printing a sample course

Void printCourseInfo (Node\* node)

If (node is equal to a null pointer)

Return

endIf

printCourseInfo(left node)

Display the courseNum and name, do not end line

i = 0

while (node’s preReq(i) does not equal a null pointer)

display node’s preReq(i), do not end line

increment i by one

endWhile

end line

printCourseInfo(right node)

end printCourseInfo

Pseudocode for printing a menu

void displayMenu(int argc, char\* argv[1], courseRow vector fileData, vector preReq)

Declare a string for the csvPath

Switch (argc)

Case 2:

Set csvPath to equal argv[1]

Break

Default:

Set csvPath to equal the file with the course information

end switch

define a vector to hold all the data named fileData

Declare an interger named choice to equal 0

While (choice!=9)

Display Menu:

Display 1. Load data

Display 2. Print course list

Display 3. Print course info

Display 9. Exit

Display enter choice:

Read the users choice variable

Switch (choice)

Case 1:

Set fileData equal to readParseValidate(csvPath)

Break

Case 2:

printCourseInfo(node)

break

Case 3:

Display enter desired course number

Read in the courseNum variable from the user

Declare a node named current equal to the root

While (current does not equal a null pointer)

If (the current’s courseNum is equal to the given courseNum

Display current’s fileData

endIf

if (the courseNum is smaller than the current node)

set current equal to current’s left node

endIf

else

set current equal to current’s right node

end else

end while

break

end switch

end while

display Good bye

end displayMenu

Pseudocode for alphanumeric course sort

Int partition (coursRow vector fileData, int begin, int end)

Declare integer low as the beginning of vector fileData

Declare interger high as the end of vector fileData

Declare interge pivot as begin + (end – begin) / 2

Declare boolean done as false

While not done

While (fileData[low] < fileData[pivot])

Increment low by 1

EndWhile

While (fileData[pivot] < fileData[high]

Decrement high by 1

EndWhile

If (low is less than or equal to high)

Declare done as true

Else

Swap fileData[low] with fileData[high]

Increment low by 1

Decrement high by 1

endIF

EndWhile

Return high

end partition

Void quicksort (courseRow vector fileData, int begin, int end)

Declare integer mid as 0

If (begin is less than or equal to end)

Return

endIf

declare mid equal to partition(vector fileData, int begin, int end)

call function quickSort(vector fileData, int begin, int mid)

call function quickSort(vector fileData, int mid plus 1, int end)

end quickSort