**CS300 MOD 4 Milestone Pseudocode and Runtime Complexities**

Pseudocode

**BEGIN**

STRUCT Course

INIT string courseNumber

INIT string courseName

INIT vector <string type> preReq

END STRUCT

CLASS HashTable

PRIVATE:

STRUCT Node

INITIALIZE Course type course

INITIALIZE Unsigned Integer type key

INITIALIZE Node type next

DEFAULT CONSTRUCTOR Node

ASSIGN key with uint\_max

ASSIGN next with null

END DEFAULT CONSTRUCTOR

CONSTRUCTOR Node

PASS IN: Course type acourse, unsigned integer type akey

ASSIGN key with akey

ASSIGN course with acourse

END CONSTRUCTOR

END STRUCT

INITIALIZE vector<Node type> nodes

INITIALIZE unsigned integer type tableSize with number of lines in fileName

INITIALIZE FUNCTION hash with pass in value: integer type key

INITIALIZE vector<Course> transferVector

END PRIVATE

PUBLIC:

INITIALIZE DEFAULT CONSTRUCTOR HashTable

INITIALIZE CONSTRUCTOR HashTable with pass in value: unsigned integer type size

INITIALIZE FUNCTION printAll

INITIALIZE FUNCTION insert with pass in value: Course type course

INITIALIZE FUNCTION search with pass in value: string courseNumber

INITIALIZE FUNCTION loadVector

END PUBLIC

END CLASS

DEFAULT CONSTRUCTOR HashTable

RESIZE nodes to the size of tableSize

END DEFAULT CONSTRUCTOR

CONSTRUCTOR HashTable

PASS IN: unsigned integer type size

ASSIGN this->tableSize with size

RESIZE nodes to the size of size

END CONSTRUCTOR

FUNCTION hash

PASS IN: integer type key

RETURN key modulo tableSize

END FUNCTION

FUNCTION printAll

INITIALIZE auto type startNode to equal the beginning node in nodes

INITIALIZE auto type endNode to equal the end node in nodes

WHILE startNode does NOT equal endNode THEN

If startNode’s key is NOT uint\_max THEN

OUTPUT “Key “

OUTPUT startNode’s key

OUTPUT “: “

OUTPUT startNode’s courseNumber

OUTPUT” | “

OUTPUT startNode’s courseName

INITIALIZE integer type index as 0

INITIALIZE integer type length as size of startNode->course.preReq

FOR index is less than length THEN

OUTPUT “ | “

OUTPUT startNode->course.preReq[index]

IF index does NOT equal (length minus 1) THEN

OUTPUT “ | “

END IF

END FOR

INITIALIZE Node type nextNode as startNode->next

WHILE nextNode is NOT null THEN

OUTPUT nextNode’s key

OUTPUT “: “

OUTPUT nextNode’s courseNumber

OUTPUT” | “

OUTPUT nextNode’s courseName

INITIALIZE integer type index as 0

INITIALIZE integer type length as size of nextNode->course.preReq

FOR index in length THEN

OUTPUT “ | “

OUTPUT nextNode->course.preReq[index]

IF index does NOT equal (length minus 1) THEN

OUTPUT “ | “

END IF

INCREMENT index

END FOR

END WHILE

END IF

ASSIGN startNode with startNode->next

END WHILE

END FUNCTION

FUNCTION loadTable

PASS IN: string filename, HashTable\* hashTable

INIT string type line

INIT string type parser

INIT vector<string type> tempVector

OUTPUT "Opening the following file: "

OUTPUT fileName

OUTPUT NEWLINE

OPEN fileName

IF fileName does NOT open THEN

OUTPUT "File cannot be opened"

CLOSE filename

ELSE THEN

WHILE NOT end of file THEN

EMPTY tempVector

GET LINE with fileName, line

IF length of line equals 0 THEN

OUTPUT “No data in line”

END IF

SWITCH (count of "," in line)

CASE 0 THEN

OUTPUT "Invalid line"

BREAK

CASE 1 THEN

WHILE getline(line, parser, “,”) THEN

PUSH BACK parser into tempVector

END WHILE

ASSIGN course.courseNumber with tempVector[0]

ASSIGN course.courseName with tempVector[1]

CALL hashTable->insert(course)

BREAK

DEFAULT THEN

WHILE getline(line, parser, “,”) THEN

PUSH BACK parser into tempVector

END WHILE

ASSIGN course.courseNumber with tempVector[0]

ASSIGN course.courseName with tempVector[1]

INIT integer preReqCount with 2

WHILE preReqCount is less than size of tempVector THEN

INIT integer courseNumCount with count of tempVector[preReqCount] in fileName

IF courseNumCount > 1 THEN

ASSIGN course.preReq with tempVector[preReqCount]

END IF

INCREMENT preReqCount

END WHILE

CALL hashTable->insert(course)

BREAK

END CASE

END WHILE

CLOSE fileName

END IF

END FUNCTION

FUNCTION insert

PASS IN: Course type course

INITIALIZE id as course.courseNumber data cast into an integer

INTIALIZE key as CALL hash function with id as pass in value

INITIALIZE Node type currNode as nodes[key]

IF currNode->key == uint\_max THEN

ASSIGN currNode->key with key

ASSIGN currNode->course with course

ELSE IF currNode->next == null THEN

ASSIGN currNode->next with a new Node(course, key)

ELSE THEN

WHILE currNode->next is NOT null THEN

ASSIGN currNode with currNode->next

END WHILE

ASSIGN currNode->next with a new Node(course, key)

END IF

END FUNCTION

FUNCTION search

PASS IN: string type courseNumber

INIT Course course

INIT integer key as CALL to hash with courseNumber

INIT Node currNode as &nodes[key]

IF currNode->key is NOT uint\_max AND currNode->course.courseNumber equals courseNumber THEN

RETURN currNode->course

END IF

IF currNode->key equals uint\_max THEN

RETURN ccourse

END IF

WHILE currNode is NOT null THEN

IF currNode->key is NOT uint\_max AND currNode->course.courseNumber equals courseNumber THEN

RETURN currNode->course

END IF

ASSIGN currNode with currNode->next

END WHILE

RETURN course

END FUNCTION

FUNCTION printCourse

PASS IN: string courseNumber

INIT Course course as CALL to search with courseNumber

IF course is empty THEN

DISPLAY “Course was not found”

DISPLAY NEWLINE

ELSE THEN

DISPLAY course.courseNumber

DISPLAY “ | “

DISPLAY course.courseName

INITIALIZE integer type index as 0

INITIALIZE integer type length as size of course.preReq

FOR index in length THEN

OUTPUT “ | “

OUTPUT course.preReq[index]

IF index does NOT equal (length minus 1) THEN

OUTPUT “ | “

END IF

INCREMENT index

END FOR

END IF

END FUNCTION

FUNCTION loadVector

PASS IN: vector<Course>& transferVector

INITIALIZE auto type startNode to equal the beginning node in nodes

INITIALIZE auto type endNode to equal the end node in nodes

WHILE startNode does NOT equal endNode THEN

If startNode’s key is NOT uint\_max THEN

PUSH BACK startNode->course into transferVector

INIT Node currNode as startNode->next

WHILE currNode is NOT null THEN

PUSH BACK currNode->course into transferVector

ASSIGN currNode with currNode->next

END WHILE

END IF

END WHILE

END FUNCTION

FUNCTION insertionSort

PASS IN: vector<Course>& transferVector

INIT integer index1 as 1

INIT integer index2 as 0

INIT string temp

INIT vectorSize as size of transferVector

IF vectorSize is less than 2 THEN

DISPLAY “Insufficient courses to sort”

DISPLAY NEWLINE

RETURN

END IF

FOR index1 is less than vectorSize THEN

ASSIGN index2 with index1

WHILE index2 is greater than 0 AND transferVector[index2] is less than transferVector[index2 minus 1] THEN

ASSIGN temp with transferVector[index2]

ASSIGN transferVector[index2] as transferVector[index2 minus 1]

ASSIGN transferVector[index2 minus 1] with temp

DECREMENT index2

END WHILE

INCREMENT index1

END WHILE

END FUNCTION

FUNCTION printVector

PASS IN: vector<Course>& transferVector

INIT integer type coursesSize as the size of transferVector

INIT integer type index1 as 0

INIT integer type index2 as 0

WHILE index1 is less than coursesSize THEN

` OUTPUT transferVector[index1].courseNumber

OUTPUT “, “

OUTPUT transferVector[index1].courseName

INIT integer length as size of transferVector[index1].preReq

WHILE index2 is less than length THEN

OUTPUT “, “

OUTPUT transferVector[index1].preReq[index2]

IF index2 does NOT equal (length minus 1) THEN

OUTPUT ","

END IF

INCREMENT index2

END WHILE

INCREMENT index1

END WHILE

END FUNCTION

MAIN

INIT integer type choice as 0

INIT string type csvName as the name of the CSV file being loaded

INIT HashTable\* courseTable

INIT Course course

INIT courseTable as a new HashTable object

INIT vector<Course> sortVector

WHILE choice is NOT 9 THEN

DISPLAY “Menu:”

DISPLAY NEWLINE

DISPLAY “1. Load Courses”

DISPLAY NEWLINE

DISPLAY “2. Print all Courses”

DISPLAY NEWLINE

DISPLAY “3. Find Course”

DISPLAY NEWLINE

ASSIGN choice with input from the user.

SWITCH (choice)

CASE 1 THEN

CALL loadTable with csvName, courseTable

BREAK

CASE 2 THEN

CALL loadVector with sortVector

CALL insertionSort with sortVector

CALL printVector with sortVector

BREAK

CASE 3 THEN

INIT string type courseNum

DISPLAY “Please input the Course Number you are searching for.

DISPLAY NEWLINE

INPUT user inputs a Course Number

ASSIGN courseNum with INPUT

ASSIGN course with CALL to courseTable->printCourse(courseNum)

BREAK

CASE 9 THEN

EXIT

END SWITCH

END WHILE

END MAIN

**END**

Analysis

**hash**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| RETURN key modulo tableSize | 1 | 1 | 1 |
| **Total Cost** | | | 1 |
| **Runtime** | | | O(1) |

**Course**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT string courseNumber | 1 | 1 | 1 |
| INIT string courseName | 1 | 1 | 1 |
| INIT vector <string type> preReq | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

**Node**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE Course type course | 1 | 1 | 1 |
| INITIALIZE Unsigned Integer type key | 1 | 1 | 1 |
| INITIALIZE Node type next | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

**Default Constructor Node**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| ASSIGN key with uint\_max | 1 | 1 | 1 |
| ASSIGN next with null | 1 | 1 | 1 |
| **Total Cost** | | | 2 |
| **Runtime** | | | O(1) |

**Constructor Node**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| ASSIGN key with akey | 1 | 1 | 1 |
| ASSIGN course with acourse | 1 | 1 | 1 |
| **Total Cost** | | | 2 |
| **Runtime** | | | O(1) |

**Class**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE vector<Node type> nodes | 1 | 1 | 1 |
| INITIALIZE unsigned integer type tableSize with number of lines in fileName | 1 | 1 | 1 |
| INITIALIZE FUNCTION hash with pass in value: integer type key | 1 | 1 | 1 |
| INITIALIZE vector<Course> transferVector | 1 | 1 | 1 |
| **Total Cost** | | | 4 |
| **Runtime** | | | O(1) |

**Class (Public)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE DEFAULT CONSTRUCTOR HashTable | 1 | 1 | 1 |
| INITIALIZE CONSTRUCTOR HashTable with pass in value: unsigned integer type size | 1 | 1 | 1 |
| INITIALIZE FUNCTION printAll | 1 | 1 | 1 |
| INITIALIZE FUNCTION insert | 1 | 1 | 1 |
| INITIALIZE FUNCTION search with pass in value: string courseNumber | 1 | 1 | 1 |
| INITIALIZE FUNCTION transferVector | 1 | 1 | 1 |
| **Total Cost** | | | 6 |
| **Runtime** | | | O(1) |

**Default Constructor HashTable**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| RESIZE nodes to the size of tableSize | 1 | 1 | 1 |
| **Total Cost** | | | 1 |
| **Runtime** | | | O(1) |

**Constructor HashTable**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| ASSIGN this->tableSize with size | 1 | 1 | 1 |
| RESIZE nodes to the size of size | 1 | 1 | 1 |
| **Total Cost** | | | 2 |
| **Runtime** | | | O(1) |

**printAll**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE auto type startNode to equal the beginning node in nodes | 1 | 1 | 1 |
| INITIALIZE auto type endNode to equal the end node in nodes | 1 | 1 | 1 |
| WHILE startNode does NOT equal endNode THEN | 1 | N | N, 1 |
| If startNode’s key is NOT uint\_max THEN | 1 | 1 | 1 |
| OUTPUT “Key “ | 1 | 1 | 1 |
| OUTPUT startNode’s key | 1 | 1 | 1 |
| OUTPUT “: “ | 1 | 1 | 1 |
| OUTPUT startNode’s courseNumber | 1 | 1 | 1 |
| OUTPUT” | “ | 1 | 1 | 1 |
| OUTPUT startNode’s courseName | 1 | 1 | 1 |
| INITIALIZE integer type index as 0 | 1 | 1 | 1 |
| INITIALIZE integer type length as size of startNode->course.preReq | 1 | 1 | 1 |
| FOR index is less than length THEN | 1 | N | N, 1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| OUTPUT startNode->course.preReq[index] | 1 | 1 | 1 |
| IF index does NOT equal (length minus 1) THEN | 1 | 1 | 1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| END FOR | 1 | 1 | 1 |
| INITIALIZE Node type nextNode as startNode->next | 1 | 1 | 1 |
| WHILE nextNode is NOT null THEN | 1 | N | N,1 |
| OUTPUT nextNode’s key | 1 | 1 | 1 |
| OUTPUT “: “ | 1 | 1 | 1 |
| OUTPUT nextNode’s courseNumber | 1 | 1 | 1 |
| OUTPUT” | “ | 1 | 1 | 1 |
| OUTPUT nextNode’s courseName | 1 | 1 | 1 |
| INITIALIZE integer type index as 0 | 1 | 1 | 1 |
| INITIALIZE integer type length as size of nextNode->course.preReq | 1 | 1 | 1 |
| FOR index in length THEN | 1 | N | N,1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| OUTPUT nextNode->course.preReq[index] | 1 | 1 | 1 |
| IF index does NOT equal (length minus 1) THEN | 1 | 1 | 1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| INCREMENT index | 1 | 1 | 1 |
| END FOR | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| INCREMENT startNode | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 3  Worst Case: 2 + N(10 + N(5) + 2 + N(8 + N(6) + 1) + 2) + 1 |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N^3) |

**insert**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE id as course.courseNumber data cast into an integer | 1 | 1 | 1 |
| INTIALIZE key as CALL hash function with id as pass in value | 1 | 1 | 1 |
| INITIALIZE Node type currNode as nodes[key] | 1 | 1 | 1 |
| If currNode->key == uint\_max THEN | 1 | 1 | 1 |
| ASSIGN currNode->key with key | 1 | 1 | 1 |
| ASSIGN currNode->course with course | 1 | 1 | 1 |
| ELSE IF currNode->next == null THEN | 1 | 1 | 1 |
| ASSIGN currNode->next with a new Node(course, key) | 1 | 1 | 1 |
| ELSE THEN | 1 | 1 | 1 |
| WHILE currNode->next is NOT null THEN | 1 | N | N, 1 |
| ASSIGN currNode with currNode->next | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| ASSIGN currNode->next with a new Node(course, key) | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 6  Worst Case: 7 + 2N |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N) |

**loadTable**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT String line | 1 | 1 | 1 |
| INIT String parser | 1 | 1 | 1 |
| INIT vector<String type> tempVector | 1 | 1 | 1 |
| OUTPUT "Opening the following file: " | 1 | 1 | 1 |
| OUTPUT fileName | 1 | 1 | 1 |
| OUTPUT NEWLINE | 1 | 1 | 1 |
| OPEN fileName | 1 | 1 | 1 |
| IF fileName does NOT open THEN | 1 | 1 | 1 |
| OUTPUT "File cannot be opened" | 1 | 1 | 1 |
| CLOSE fileName | 1 | 1 | 1 |
| ELSE THEN | 1 | 1 | 1 |
| WHILE NOT end of file THEN | 1 | N | N, 1 |
| EMPTY tempVector | 1 | 1 | 1 |
| GET LINE with fileName, line | 1 | 1 | 1 |
| IF length of line equals 0 THEN | 1 | 1 | 1 |
| OUTPUT “No data in line” | 1 | 1 | 1 |
| SWITCH (count of "," in line) | 1 | 1 | 1 |
| CASE 0 THEN | 1 | 1 | 1 |
| OUTPUT "Invalid line" | 1 | 1 | 1 |
| BREAK | 1 | 1 | 1 |
| CASE 1 THEN | 1 | 1 | 1 |
| WHILE getline(line, parser, “,”) THEN | 1 | N | N, 1 |
| PUSH BACK parser into tempVector | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| ASSIGN course.courseNumber with tempVector[0] | 1 | 1 | 1 |
| ASSIGN course.courseName with tempVector[1] | 1 | 1 | 1 |
| CALL hashTable->insert(course) | 1 | 1 | Best Case: 5  Worst Case: 8 + 2N |
| BREAK | 1 | 1 | 1 |
| DEFAULT THEN | 1 | 1 | 1 |
| WHILE getline(line, parser, “,”) THEN | 1 | N | N, 1 |
| PUSH BACK parser into tempVector | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| ASSIGN course.courseNumber with tempVector[0] | 1 | 1 | 1 |
| ASSIGN course.courseNumber with tempVector[1] | 1 | 1 | 1 |
| INIT integer preReqCount with 2 | 1 | 1 | 1 |
| WHILE preReqCount is less than size of tempVector THEN | 1 | N | N, 1 |
| INIT integer courseNumCount with count of tempVector[preReqCount] in fileName | 1 | 1 | 1 |
| IF courseNumCount > 1 THEN | 1 | 1 | 1 |
| ASSIGN course.preReq with tempVector[preReqCount] | 1 | 1 | 1 |
| INCREMENT preReqCount | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| CALL hashTable->insert(course) | 1 | 1 | Best Case: 5  Worst Case: 8 + 2N |
| BREAK | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| CLOSE fileName | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 10  Worst Case: 9 + N(8 + N(2) + 4 + N(5) + 10 + N(2)) + 2 = 11 + 31N^2 |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N^2) |

**search**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT Course course | 1 | 1 | 1 |
| INIT integer key as CALL to hash with courseNumber | 1 | 1 | 1 |
| INIT Node currNode as &nodes[key] | 1 | 1 | 1 |
| IF currNode->key is NOT uint\_max AND currNode->course.courseNumber equals courseNumber THEN | 1 | 1 | 2 |
| RETURN currNode->course | 1 | 1 | 1 |
| IF currNode->key equals uint\_max THEN | 1 | 1 | 1 |
| RETURN course | 1 | 1 | 1 |
| WHILE currNode is NOT null THEN | 1 | N | N, 1 |
| IF currNode->key is NOT uint\_max AND currNode->course.courseNumber equals courseNumber THEN | 1 | 1 | 2 |
| RETURN currNode->course | 1 | 1 | 1 |
| ASSIGN currNode with currNode->next | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| RETURN course | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 6  Worst Case: 6 + 4N + 2 = 8 + 4N |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N) |

**printCourse**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT Course course as CALL to search with courseNumber | 1 | 1 | Best Case: 6  Worst Case: 6 + 4N + 2 = 8 + 4N |
| IF course is empty THEN | 1 | 1 | 1 |
| DISPLAY “Course was not found” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| ELSE THEN | 1 | 1 | 1 |
| DISPLAY course.courseNumber | 1 | 1 | 1 |
| DISPLAY “ | “ | 1 | 1 | 1 |
| DISPLAY course.courseName | 1 | 1 | 1 |
| INITIALIZE integer type index as 0 | 1 | 1 | 1 |
| INITIALIZE integer type length as size of course.preReq | 1 | 1 | 1 |
| FOR index in length THEN | 1 | N | N, 1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| OUTPUT course.preReq[index] | 1 | 1 | 1 |
| IF index does NOT equal (length minus 1) THEN | 1 | 1 | 1 |
| OUTPUT “ | “ | 1 | 1 | 1 |
| INCREMENT index | 1 | 1 | 1 |
| END FOR | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 9  Worst Case: 8 + 4N + 7 + 6N + 1 = 16 + 10N |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N) |

**loadVector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE auto type startNode to equal the beginning node in nodes | 1 | 1 | 1 |
| INITIALIZE auto type endNode to equal the end node in nodes | 1 | 1 | 1 |
| WHILE startNode does NOT equal endNode THEN | 1 | N | N, 1 |
| If startNode’s key is NOT uint\_max THEN | 1 | 1 | 1 |
| PUSH BACK startNode->course into transferVector | 1 | 1 | 1 |
| INIT Node currNode as startNode->next | 1 | 1 | 1 |
| WHILE currNode is NOT null THEN | 1 | 1 | N, 1 |
| PUSH BACK currNode->course into transferVector | 1 | 1 | 1 |
| ASSIGN currNode with currNode->next | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 3  Worst Case: 2 + N(4 + N(3) + 1) + 1 = 3 + 8N^2 |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N^2) |

**insertionSort**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT integer index1 as 1 | 1 | 1 | 1 |
| INIT integer index2 as 0 | 1 | 1 | 1 |
| INIT string temp | 1 | 1 | 1 |
| INIT vectorSize as size of transferVector | 1 | 1 | 1 |
| IF vectorSize is less than 2 THEN | 1 | 1 | 1 |
| DISPLAY “Insufficient courses to sort” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| RETURN | 1 | 1 | 1 |
| FOR index1 is less than vectorSize THEN | 1 | N | N, 1 |
| ASSIGN index2 with index1 | 1 | 1 | 1 |
| WHILE index2 is greater than 0 AND transferVector[index2] is less than transferVector[index2 minus 1] THEN | 1 | 1 | N, 2 |
| ASSIGN temp with transferVector[index2] | 1 | 1 | 1 |
| ASSIGN transferVector[index2] as transferVector[index2 minus 1] | 1 | 1 | 1 |
| ASSIGN transferVector[index2 minus 1] with temp | 1 | 1 | 1 |
| DECREMENT index2 | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| INCREMENT index1 | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 8  Worst Case: 3 + N(4 + N(3) + 1) + 2 = 5 + 8N^2 |
| **Runtime** | | | Best case: O(1)  Worst case:  O(N^2) |

**printVector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT integer type coursesSize as the size of transferVector | 1 | 1 | 1 |
| INIT integer type index1 as 0 | 1 | 1 | 1 |
| INIT integer type index2 as 0 | 1 | 1 | 1 |
| WHILE index1 is less than coursesSize THEN | 1 | N | N, 1 |
| OUTPUT transferVector[index1].courseNumber | 1 | 1 | 1 |
| OUTPUT “, “ | 1 | 1 | 1 |
| OUTPUT transferVector[index1].courseName | 1 | 1 | 1 |
| INIT integer length as size of transferVector[index1].preReq | 1 | 1 | 1 |
| WHILE index2 is less than length THEN | 1 | 1 | 1 |
| OUTPUT “, “ | 1 | 1 | 1 |
| OUTPUT transferVector[index1].preReq[index2] | 1 | 1 | 1 |
| IF index2 does NOT equal (length minus 1) THEN | 1 | 1 | 1 |
| OUTPUT "," | 1 | 1 | 1 |
| INCREMENT index2 | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| INCREMENT index1 | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 5  Worst Case: 3 + N(5 + N(6) + 2) + 1 = 4 + 13N^2 |
| **Runtime** | | | Best case: O(1)  Worst case: O(N^2) |

**MAIN**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INIT integer type choice as 0 | 1 | 1 | 1 |
| INIT string type csvName as the name of the CSV file being loaded | 1 | 1 | 1 |
| INIT HashTable\* courseTable | 1 | 1 | 1 |
| INIT Course course | 1 | 1 | 1 |
| INIT courseTable as a new HashTable object | 1 | 1 | 1 |
| INIT vector <Course Type> sortVector | 1 | 1 | 1 |
| WHILE choice is NOT 9 THEN | 1 | N | N, 1 |
| DISPLAY “Menu:” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| DISPLAY “1. Load Courses” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| DISPLAY “2. Print all Courses” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| DISPLAY “3. Find Course” | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| ASSIGN choice with input from the user. | 1 | 1 | 1 |
| SWITCH (choice) | 1 | 1 | 1 |
| CASE 1 THEN | 1 | 1 | 1 |
| CALL loadTable with csvName, courseTable | 1 | 1 | Best Case: 10  Worst Case: 9 + N(8 + N(2) + 4 + N(5) + 10 + N(2)) + 2 = 11 + 31N^2 |
| BREAK | 1 | 1 | 1 |
| CASE 2 THEN | 1 | 1 | 1 |
| CALL loadVector with sortVector | 1 | 1 | Best Case: 3  Worst Case: 2 + N(4 + N(3) + 1) + 1 = 3 + 8N^2 |
| CALL insertionSort with sortVector | 1 | 1 | Best Case: 8  Worst Case: 3 + N(4 + N(3) + 1) + 2 = 5 + 8N^2 |
| CALL printVector with sortVector | 1 | 1 | Best Case: 4  Worst Case: 3 + N(5 + N(6) + 2) + 1 = 4 + 13N^2 |
| BREAK | 1 | 1 | 1 |
| CASE 3 THEN | 1 | 1 | 1 |
| INIT string type courseNum | 1 | 1 | 1 |
| DISPLAY “Please input the Course Number you are searching for. | 1 | 1 | 1 |
| DISPLAY NEWLINE | 1 | 1 | 1 |
| INPUT user inputs a Course Number | 1 | 1 | 1 |
| ASSIGN courseNum with INPUT | 1 | 1 | 1 |
| ASSIGN course with CALL to courseTable->printCourse(courseNum) | 1 | 1 | Best Case: 9  Worst Case: 8 + 4N + 7 + 6N + 1 = 16 + 10N |
| BREAK | 1 | 1 | 1 |
| CASE 9 THEN | 1 | 1 | 1 |
| EXIT | 1 | 1 | 1 |
| END WHILE | 1 | 1 | 1 |
| **Total Cost** | | | Best Case: 6 + 16N  Worst Case: 6 + N(12 + 11 + 31N^2 + 1) = 6 + 55N^3 |
| **Runtime** | | | Best case: O(N)  Worst case: O(N^3) |