Marisa Kuyava
CS 300
Project One

The purpose of this document is to analyze three types of data structures (vector, hash table and tree), explain the advantages/disadvantages of each structure and provide a recommendation on which data structure I will be using for project two.

The runtime analysis of each data structure type did not give me as definitive of an answer as I expected. The worst-case complexity for all three are O(n) so this prompted me to look at the other advantages and disadvantages of these data structures and the project I am building to make my final decision. Based on the pros and cons listed below in conjunction with the runtime analysis and having some experience in working with all three structures my recommendation on which data structure I will be using for project two is the Binary Search Tree.

Vectors:

- Dynamic allocation – This might be beneficial as classes are added; however it does not play a huge role for the initial build of this project.
- Easy implementation
- Adding and removing the last element is easy and quick – But only the last one - Inserting or removing data from the middle of the structure is not recommended
- Searching can become very slow as the list grows because we can only use a linear search

Hash:

- Elements can be accessed directly with the key
- May take up more room than what it needs
- Collisions must be delt with
- Sorting data in alphanumeric order within the table is not possible – As displaying the data as such is a requirement for this project this make hash tables an unlikely candidate.
- Searching is more difficult if the key is unknown.

Tree:

- Tree must be balanced to perform well
- Tree can end up being very tall/unbalanced if data is loaded when it is already sorted.
- Easy to traverse
- Access speed is superior
- Maintains a sorted order (smaller to the left, larger to the right)

VECTOR : LINEPARSER PSEUDOCODE

-Used to validate data looking for formatting errors before course is inserted

```
lineParser(vector<string> oneLine)
        if oneLine.size() is equal to 2 the line has required format and can be added
                Create new course
                Set courseNumber equal to line 0
                Set courseName equal to line 1
                Return new course
        Else if line size is greater than 2
                Create a new course
                Set courseNumber equal to line 0
                Set courseName equal to line 1
                for each additional line until the end of the vector
                        pushback each line greater than 1 to prerequisite vector
                Return new course
        Else if line size is less than 2
                PRINT There is an error in the file format. Every course must have a course
                number and course name
```

Vector : LineParser Pseudocode	**Line Cost**	**# Times Executes**	**Total Cost**
if oneLine.size() is equal to 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
Return new course	1	1	1
Else if line size is greater than 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
for each additional line until the end of the vector	1	n	n
Pushback each line	1	n	n
Return new course	1	1	1
Else if line size is less than	1	n	n
Print error	1	1	1
		Total Cost	5n + 9
		Runtime	O(n)

VECTOR : INSERT PSEUDOCODE

-Insert Course into Vector

```
Insert(Course* courseNumber)
        Courses.push_back(courseNumer)
```

Vector : Insert Pseudocode	Line Cost	# Times Executes	Total Cost
pushback	1	1	1
		Total Cost	1
		Runtime	O(n)

VECTOR : LOAD FILE PSEUDOCODE

-Code for file loading into vector

```
loadFile(file FileName)
        Create vector of Courses
        Create vector of strings to hold file data
        String variable to hold each line
        Open file with Ifstream
        while get line finds a next line in the file
                stringstream stst (line)
                while stst.good() is to true
                        create variable to store substring of line
                        Use get line to break substring from string using comma delimitator
                        Push substring to temporary <string> vector
        Insert temp line vector to Courses vector using Insert Function and lineParser function
        Clear temporary vector
```

Vector : Load File Pseudocode	Line Cost	# Times Executes	Total Cost
Create vector of Courses	1	1	1
Create vector of Strings	1	1	1
Create string variable	1	1	1
Open file with Ifstream	1	1	1
While getLine finds a line	1	n	n
Stringstream (line)	1	1	1
Create variable to hold substring	1	n	n
Use getLine to break string	1	n	n
Push substring to temp vector	1	n	n
Parse with lineParser	1	n	n
Insert to Courses with insert function	1	n	n
Clear temporary vector	1	1	1
		Total Cost	6n + 6
		Runtime	O(n)

HASHTABLE : LINEPARSER PSEUDOCODE

-Used to validate data for formatting errors before course is inserted

```
lineParser(vector<string> line)
        if line.size() is equal to 2 line can be added as it has required format{
                Create new course
                Set courseNumber equal to line 0
                Set courseName equal to line 1
                Return new course
        Else if line size is greaten than 2
                Create new course
                Set courseNumber equal to line 0
                Set courseName equal to line 1
                for each additional line until the end of the vector{
                        pushback each line greater than 1 to prerequisite vector
                Return new course
        Else if line size is less than 2
                PRINT There is an error in the file format. Every course must have a course
                number and course name
```

Hashtable : LineParser Pseudocode	**Line Cost**	**# Times Executes**	**Total Cost**
if oneLine.size() is equal to 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
Return new course	1	1	1
Else if line size is greater than 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
for each additional line until the end of the vector	1	n	n
Pushback each line	1	n	n
Return new course	1	1	1
Else if line size is less than	1	n	n
Print error	1	1	1
		Total Cost	5n + 9
		Runtime	O(n)

HashTable: Insert Pseudocode

-Insert Course into HashTable

```
Insert(Course* courseNumber)
        Using hash function create key from courseNumber
        Create keyNode to retrieve node via key created
        If keynode is empty/null
                Add course at current empty node
        Else if keyNode is not empty
                While loop through keyNodes linked list until an empty node is found
                        Add course at empty node found
```

HashTable: Insert Pseudocode	Line Cost	# Times Executes	Total Cost
Using hash function create key from courseNumber	1	1	1
Create keyNode to retrieve node via key created	1	1	1
If keynode is empty/null	1	n	n
Add course at current empty node	1	1	1
Else if keyNode is not empty	1	n	n
While loop until an empty node is found	1	n	n
Add course at empty node	1	1	1
		Total Cost	3n + 4
		Runtime	O(n)

HashTable: Load File Pseudocode

-Code for file loading into HashTable

```
loadFile(file FileName)
```

Create hashtable
Create vector of strings to hold file data
String variable to hold each line
Open file with Ifstream
while get line finds a next line in the file
    stringstream stst (line)
    while stst.good() is to true
        create variable to store substring of line
        Use get line to break substring from string using comma delimitator
        Push substring to temporary <string> vector
Insert temporary line vector to hashtable using Insert Function and lineParser function
Clear temporary vector

Hash Table : Load File Pseudocode	**Line Cost**	**# Times Executes**	**Total Cost**
Create hashtable of Courses	1	1	1
Create vector of Strings	1	1	1
Create string variable	1	1	1
Open file with Ifstream	1	1	1
While getLine finds a line	1	n	n
Stringstream (line)	1	1	1
Create variable to hold substring	1	n	n
Use getLine to break string	1	n	n
Push substring to temp vector	1	n	n
Parse with lineParser	1	n	n
Insert to Courses with insert function	1	n	n
Clear temporary vector	1	1	1
		Total Cost	6n + 6
		Runtime	O(n)

BINARY SEARCH TREE : LINEPARSER PSEUDOCODE

-Used to validate data for formatting errors before course is inserted

lineParser(vector<string> line)

if line.size() is equal to 2 line can be added as it has required format
    Create new course
    Set courseNumber equal to line 0
    Set courseName equal to line 1
    Return new course
Else if line size is greater than 2
    Create new course
    Set courseNumber equal to line 0
    Set courseName equal to line 1
    for each additional line until the end of the vector
        pushback each line greater than 1 to prerequisite vector
    Return new course
Else if line size is less than 2
    PRINT There is an error in the file format. Every course must have a course number and course name

Binary Tree : LineParser Pseudocode	Line Cost	# Times Executes	Total Cost
if oneLine.size() is equal to 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
Return new course	1	1	1
Else if line size is greater than 2	1	n	n
Create new course	1	1	1
Set courseNumber equal to line 0	1	1	1
Set courseName equal to line 1	1	1	1
for each additional line until the end of the vector	1	n	n
Pushback each line	1	n	n
Return new course	1	1	1
Else if line size is less than	1	n	n
Print error	1	1	1
		Total Cost	5n + 9
		Runtime	O(n)

Binary Search Tree : Insert Pseudocode

-Insert

void Insert(Bid bid)

if the root is null

Make a newNode with bid and set equal to the node root

Else (if the root is not null)

Make a call to public function ‘addNode’ and as a parameter pass the root

Binary Tree : Insert Pseudocode	**Line Cost**	**# Times Executes**	**Total Cost**
if the root is null	1	n	n
Create new Node with bid	1	1	1
Set equal to the root node	1	1	1
Else (if the root is not null)	1	n	n
Function call to addNode pass the root as parameter	1	1	1
		Total Cost	2n + 3
		Runtime	O(n)

BINARY SEARCH TREE : LOAD FILE PSEUDOCODE

-Code for file loading into HashTable

```
loadFile(file FileName)
    Create BinaryTree
    Create vector of strings to hold file data
    String variable to hold each line
    Open file with Ifstream
    while get line finds a next line in the file
        stringstream stst (line)
        while stst.good() is to true
            create variable to store substring of line
            Use get line to break substring from string using comma delimitator
            Push substring to temporary <string> vector
    Insert temporary line vector to BinaryTree using Insert Function and lineParser function
    Clear temporary vector
```

Binary Tree : Load File Pseudocode	**Line Cost**	**# Times Executes**	**Total Cost**
Create Tree of Courses	1	1	1
Create vector of Strings	1	1	1
Create string variable	1	1	1
Open file with Ifstream	1	1	1
While getLine finds a line	1	n	n
Stringstream (line)	1	1	1
Create variable to hold substring	1	n	n
Use getLine to break string	1	n	n
Push substring to temp vector	1	n	n
Parse with lineParser	1	n	n
Insert to Courses with insert function	1	n	n
Clear temporary vector	1	1	1
		Total Cost	6n + 6
		Runtime	O(n)

MENU PSEUDOCODE

Define data structure to hold all of the courses
Create integer variable to hold user's choice
Create string variable for case 3 - courseID

While choice does not equal 4 (this is the number selected to exit)
    Print:
        Menu: What would you like to do?
        1. Load Data Structure
        2. Print Course List (alphanumerically ordered list of all the courses)
        3. Print Course (print course title and the prerequisites for any individual course)
        4. Exit

    Get selection input and store in choice variables

    Switch(choice)

    Case 1:
        Instantiate data structure
        Function call to loadFile()
        Break
    Case 2:
        If data structure is not null
        Function call to print courses in order
        Break
    Case 3:
        If data structure is not null
        Print: Please enter the course number
        Get input and store in courseID
            If courseID.length() does not equal zero
                Function call to PrintCourse()
        Break
    Case 4:
        Print goodbye message
        Break
    Default:
        Print Not a valid selection – Please try again
        Break

Print Course List (Alphanumeric) Pseudocode

-Traverse the tree in order

```
InOrder()
    call inOrder fuction and pass root


inOrder(Node* node)
    if node is not equal to null ptr
        recursive call to inOrder on left
        Print CourseID
        Print CourseName
        recursive call to inOrder on right
```

Print Course Pseudocode

-Print the course title and the prerequisites for any individual course.

```
PrintCourse(string courseId)
    Function call to SearchCourse() which will return a course
    Print CourseID
        Print CourseName
        Print Prerequisite One
        Print Prerequisite Two
```

Search Pseudocode

-Used for the Print Course function

```
Search(string courseId)
    Create new Node ‘current’

  while current node is not null
    Continue looping downwards until matching coursed is found or bottom reached
    If current.courseId matches courseId
      return current course
    else if courseId is smaller than current.courseId
      traverse the tree’s left side
   else ( courseId is greater than current. Coursed)
      traverse the tree’s right side
return course
```