# Cs 300: Project 1

# Ian Viguera

**Cs 300: Analysis and Design**

**Prof. Griffith**

1. **VectorData Pseudocode**

// Opening and reading files

myFile.open(“courses.txt”)

While (myFile exists)

{

For() // read through each line

{

If (values < 2 )

{

Print “Invalid Input”

}

Else if (values > 3)

{

dataReader() // function for reading parameters

}

Else

{

Print “Error on opening the file”

}

}

}

\\ Creating Course Objects

Initialize Vector<Course> courses

For() // read through vector

{

For() // read through file

{

If(value < 3)

{

Push\_back value to vector

}

Else

{

Push\_back value to file

}

}

}

̴̴

// Search and Print Prerequisite information

Prompt for user to input “searchNumber”

For() // read through vector

{

If (searchNumber == courseNumber)

{

Print stored data

}

Else

{

Print “Input not found”

}

}

1. **HashTable Pseudocode**

// Opening and reading files

myFile.open(“courses.txt”)

While (myFile exists)

{

For() // read through each line

{

If (values < 2 )

{

Print “Invalid Input”

}

Else if (values > 3)

{

dataReader() // function for reading parameters

}

Else

{

Print “Error on opening the file”

}

}

}

\\ Creating Course Objects

Initialize Vector<Course> courses

Create HashTable Class

Create Method to save items to HashTable Class

For() // read through vector

{

For() // read through file

{

If(value < 3)

{

Push\_back value to vector

}

Else

{

Push\_back value to file

}

}

}

// Search and Print Prerequisite information

**Prompt for “input”**

**Assign “input” to “key”**

**If (Key exists)**

**{**

**Print (Course Information)**

**Print (Prerequisites)**

**}**

**Else**

**{**

**Print “Invalid Input”**

**}**

1. **TreeDataStructure Pseudocode**

// Opening and reading files

myFile.open(“courses.txt”)

While (myFile exists)

{

For() // read through each line

{

If (values < 2 )

{

Print “Invalid Input”

}

Else if (values > 3)

{

dataReader() // function for reading parameters

}

Else

{

Print “Error on opening the file”

}

}

}

\\ Creating Course Objects

Initialize Vector<Course> courses

For() // read through vector

{

For() // read through file

{

If(value < 3)

{

Push\_back value to vector

}

Else

{

Push\_back value to file

}

}

}

// Search and Print Prerequisite information

**Create Binary Tree class**

**Assign “root” to nullptr**

**If(“Statement that compares root to bid” equals 0)**

**{**

**If (“root->left” equals nullptr)**

**{**

**Set “root->left” to new node**

**}**

**Else**

**{**

**Call this->addNode(root->left, bid)**

**}**

}

Else

{

**If (“root->right” equals nullptr)**

**{**

**Set “root->right” to new node**

**}**

**Else**

**{**

**Call this->addNode(root->right, bid)**

**}**

}

1. **Menu Pseudocode**

Create “Choice” variable

While (choice not equal 9)

{

Print Menu options

Prompt for “choice”

Switch (choice)

{

Choice 1:

Load courses through file

Choice 2:

Display all bids

Choice 3:

Search for bids

Choice 4:

Delete bids

Choice 9:

Terminates program

}

}

1. **Runtime Analysis**

| VectorData  \\ Opening and reading files | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While “course.txt” exists | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 2 | 1 | n | n |
| Print “Invalid input” | 1 | 1 | 1 |
| Else if “value” is greater than 3 | 1 | n | n |
| Read through parameters | 1 | n | n |
| Else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| VectorData  \\ Creating course objects | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Initialize vector | 1 | 1 | 1 |
| For “read through vector” | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 3 | 1 | n | n |
| Push\_back value to vector | 1 | n | n |
| Else |  |  |  |
| Push\_back value to file | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| VectorData  \\ Search and Print Prerequisite information | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Prompt for searchNumber | 1 | 1 | 1 |
| For “read through vector” | 1 | n | n |
| If searchNumber equal to courseNumber | 1 | n | n |
| Print course information | 1 | n | n |
| Print prerequisites | 1 | n | n |
| Else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| HashTable  \\ Opening and reading files | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While “course.txt” exists | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 2 | 1 | n | n |
| Print “Invalid input” | 1 | 1 | 1 |
| Else if “value” is greater than 3 | 1 | n | n |
| Read through parameters | 1 | n | n |
| Else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| HashTable  \\ Creating course objects | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Initialize vector | 1 | 1 | 1 |
| Create HashTable Class | 1 | n | n |
| Save contents to HashTable | 1 | n | n |
| For “read through vector” | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 3 | 1 | n | n |
| Push\_back value to file | 1 | 1 | 1 |
| Else |  |  |  |
| Push\_back value to file | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| HashTable  \\ Search and Print Prerequisite information | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Prompt for input | 1 | 1 | 1 |
| Assign input to key | 1 | n | n |
| If Key exists | 1 | n | n |
| Print course information | 1 | 1 | 1 |
| Print prerequisites | 1 | 1 | 1 |
| Else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

| TreeDataStructure  \\ Opening and reading files | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While “course.txt” exists | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 2 | 1 | n | n |
| Print “Invalid input” | 1 | 1 | 1 |
| Else if “value” is greater than 3 | 1 | n | n |
| Read through parameters | 1 | n | n |
| Else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| TreeDataStructure  \\ Creating course objects | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Initialize vector | 1 | 1 | 1 |
| Create HashTable Class | 1 | n | n |
| Save contents to HashTable | 1 | n | n |
| For “read through vector” | 1 | n | n |
| For “read through file contents” | 1 | n | n |
| If “value” is less than 3 | 1 | n | n |
| Push\_back value to file | 1 | 1 | 1 |
| Else |  |  |  |
| Push\_back value to file | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| TreeDataStructure  \\ Search and Print Prerequisite information | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create BinaryTree Class | 1 | n | n |
| Assign root to NULL | 1 | n | n |
| If root compared to bids equals to 0 | 1 | n | n |
| If root using left, compared to bid is NULL | 1 | n | n |
| Assign root using left to new node | 1 | n | n |
| Else |  |  |  |
| Call this using addNode(root using left, bid) | 1 | n | n |
| Else |  |  |  |
| If root using right, compared to bid is NULL | 1 | n | n |
| Assign root using right to new node | 1 | n | n |
| Else |  |  |  |
| Call this using addNode(root using right, bid) | 1 | n | n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

1. **Advantages, disadvantages, and recommendations**

The same program was tested multiple times with different coding structures that produced the same outcome. The program consisted of utilizing saved data from a specific course data and output it in order, as well as adding, searching, and deleting information from the file. The Vector Data structure is the easiest structure to implement and code, but its storage capabilities and order arrangements are very limited. The Hash Table structure can expand upon storage capabilities and can hold twice the data a vector structure can hold; however, it is not limitless. The downside of this advantage is that with larger data it becomes progressively slower to run search functions. The Binary Search Tree traverses its data in almost any order, and it makes excellent use of variable data storage. The disadvantage of such structure is that it needs comprehensive and smart coding strategies to implement correctly, and smart time management.

For a program with this level of data, all three structures fulfilled their requirements. However, for a much higher demanding program, the Binary Tree data structure handles more than the necessary functions to run the program. It is more complex and more demanding than the other two structures, but it effectively improves upon the functions that this program offers. With smart time management, and smart programming strategies, this structure could be helpful for most of the programs that offer the same, or similar, experience.