Joseph Limbert

Runtime Analysis

void loadCourses(DataStructure courses, string filePath)

Code	Line Cost	# Times Executes	Total Cost
`Ifstream inputFile(filename)`	1	1	1
Course course	1	1	1
If file is not open	1	1	1
throw error	1	1	1
return	1	1	1
String line	1	1	1
While getLine(inputFile), line	1	n	n
Vector<string> result	1	n	n
Split string by comma into result vector	1	n	n
If result length is less than 2	1	n	n
Throw error	1	1	1
return	1	1	1
String courseNumber = result.at(0)	1	n	n
String courseName = result.at(1)	1	n	n
If result length is greater than 2	1	n	n
Vector<string> prerequisites(result length – 2)	1	n	n
For i = 2 if i less than result length	1	n	n
If result at i is in courses	1	O(n)*n	O(n)*n

Add result at i to prequisites	1	O(n)*n	O(n)*n
Course equals Course(courseNumber, courseName,prerequisites)	1	n	n
Else	1	n	n
course equals Course(courseNumber, courseName) to courses	1	n	n

Final cost for Vector

AddCourse(courses, course)	O(1)	1	O(1)
Total Cost			(12n + 8) * 2n + O(1)
Runtime			O(n^2)

Void addCourse(Vector<Course> courses, Course course)

Code	Line Cost	# Times Executes	Total Cost
`Insert course at end of courses`	1	1	1
Total Cost			1
Runtime			O(1)

Final cost for Hash Table

AddCourse(courses, course)	O(n)	1	O(n)
Total Cost			(12n + 8) * 2n + O(n)
Runtime			O(n^2)

Void addCourse(HashTable<Course> courses, Course course)

Code	Line Cost	# Times Executes	Total Cost
Key equals courseNumber	1	1	1

Hashed key equals courseNumber mod tableSize	1	1	1
New node equals Node(course, key)	1	1	1
Existing node equals courses at hashed key	1	1	1
If existing node is null	1	1	1
Existing node equals new node	1	1	1
Else	1	1	1
Current equals existing node	1	1	1
While current does not equal null	1	n	n
if current.next equals null	1	n	n
current.next equals new node	1	1	1
return	1	1	1
current equals current.next	1	n	n
		Total Cost	10 + 3n
		Runtime	O(n)

Final cost for Tree

AddCourse(courses, course)	O(n)	1	O(n)
		Total Cost	(12n + 8) * 2n + O(n)
		Runtime	O(n^2)

Void addCourse(Tree<Course> courses, Course course)

Code	Line Cost	# Times Executes	Total Cost
If courses root node is null	1	1	1

root node equals new Node(course)	1	1	1
Else	1	1	1
addNode(root node, course)	O(n)	1	O(n)
		Total Cost	3 + O(n)
		Runtime	O(n)

Void addNode(Node node, Course course)

Code	**Line Cost**	**# Times Executes**	**Total Cost**
If nodes course number is greater than courses course number	1	1	1
if node.left equal null	1	1	1
node.left equals new Node(course)	1	1	1
else	1	1	1
addNode(node.left, course)	1	n	n
Else	1	1	1
if node.right equals null	1	1	1
node.right equals new Node(course)	1	1	1
else	1	1	1
addNode(node.right, course)	1	n	n
		Total Cost	8 + 2n
		Runtime	O(n)

The advantages of using a vector is that it is simple to implement and doesn't require as much code. The disadvantages are that the runtime can become very large with larger datasets. The advantages of a hash table is that it has great performance for adding courses. The disadvantages are that it doesn't maintain any ordering so when we need to sort the data it can add a lot of runtime and complexity. The advantages for a binary tree is that it maintains an order so looking up and sorting the data can be much faster. Its disadvantages are that it adds a lot of complexity and can be slow if the tree is unbalanced.

For this project I recommend to use the binary tree structure. Creating the data structure from the input file may not be as fast as a hash table however there are other benefits that outweigh this. We also need to search and print a course as well as print all the courses in alphanumeric order. The binary tree structure will be much faster at these operations than the other structures.