CS 300 Runtime Analysis Document

Runtime Analysis

Code PrintCourse(CourseNumber)	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	n
`if the course is the same as courseNumber`	1	n	n
`print out the course information`	1	1	1
`for each prerequisite of the course`	1	n	n
`print the prerequisite course information`	1	n	n
		Total Cost	4n + 1
		Runtime	O(n)

Code Vector.addCourse(course)	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	n
`If no course in vector add course`	1	n	n
`for i=0; i<length(courses); i++`	3	n*(n +3)	n*(n + 3)
`If courses[i].id == course.id`	1	n	n
`Course.insert(i, course)`	1	n	n
		Total Cost	n^2 + 7n
		Runtime	O(n^2)

Code HashTable.addCourse(course)	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	n
`key = hash(course.id)`	1	n	n
`node = node[key]`	1	n	n
`If node == nullptr`	1	n	n
`Nodes[key] = node`	1	n	n
`ELSE`	1	n	n
`If node.key = -1`	1	n	n
`node.key = key`	1	n	n
`node.course = course`	1	n	n

Code HashTable.addCourse(course)	Line Cost	# Times Executes	Total Cost
node.next = nullptr	1	n	n
Else	1	n	n
If node.key = -1	1	n	n
node.key = key	1	n	n
node.course = course	1	n	n
node.next = nullptr	1	n	n
Else	1	n	n
WHILE node-> next != nullptr	1	n^2	n^2
node = node->next		n^2	n^2
node-> next = Node(bid,key)	1	n	n
		Total Cost	2n^2 + 17n
		Runtime	O(n^2)

Code BST.Insert(course)	Line Cost	# Times Executes	Total Cost
for all courses	1	N	n
If root = nullptr	1	N	n
root = New node(course)	**1**	**N**	**n**
else	**1**	**N**	**n**
addNode(root,course)	**1**	**n^2**	**n^2**
Total Cost	**5**	**4n + n^2**	**4n + n^2**
Runtime		**Runtime**	**O(n^2)**

Code Menu()	Line Cost	# Times Executes	Total Cost
Input = 0	**1**	**1**	**1**
While input != 9	**1**	**Infinite**	**Infinite**
OUTPUT Menu	**1**	**Infinite**	**Infinite**
GET input	**1**	**Infinite**	**Infinite**
IF input == 1	**1**	**Infinite**	**Infinite**
BST.LoadCourses()	**1**	**Infinite**	**Infinite**
ELSE IF input == 2	**1**	**Infinite**	**Infinite**
GET courseId	**1**	**Infinite**	**Infinite**
course = BST.Search(CourseId)	**1**	**Infinite**	**Infinite**

Code Menu()	Line Cost	# Times Executes	Total Cost
PrintCourse(course.courseId)	1	Infinite	Infinite
ELSE IF input == 3	1	Infinite	Infinite
Bst.PrintAll()	1	Infinite	Infinite
ELSE IF input == 9	1	Infinite	Infinite
OUTPUT goodbye	1	1	1
Exit()	1	1	1
		Total Cost	Infinite + 3
		Runtime	O(infinity)

Each data structure has a worst case time complexity of O(n^2) to build and search because for each line in the input file (n) up to n prerequisites may need to be added. The best case search complexity for each is O(1) because they could each match on the first comparison. Vector inserts have an average of O(n/2) = O(n). But Binary Search trees inserts and searches have an average complexity of log n for a perfectly balanced tree. The added memory cost of a Binary Search Tree is more than a hash table which is more than a vector. The hash table is usually unordered, so an additional n log n time complexity would need to be added to sort for printing (based on c++ time complexity requirements of the .sort() function). A vector can be sorted in n time while it is being built, and a Binary Search tree can be traversed in in order in an average of log n and worst case of n.

In this scenario on a standard modern computer, memory space isn't much of an issue because the memory cost is sufficiently small for each data structure. Therefore speed is the deciding factor and the average speed of a Binary Search Tree is the most desirable of these data structures, therefore I recommend using a Binary Search Tree data structure to implement the course list.