## **Runtime Analysis (Big O):**

### **Vector Implementation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Vector<Course> courses = new Vector<Course>() | 1 | 1 | 1 |
| Set<String> courseNumbers = new Set<String>() | 1 | 1 | 1 |
| File file = open(filename) | 1 | 1 | 1 |
| if file is null | 1 | 1 | 1 |
| print "Error: Unable to open file." | 1 | 0 | 0 |
| return null | 1 | 0 | 0 |
| While not end of file | 1 | n | n |
| String line = file.readLine() | 1 | n | n |
| List<String> tokens = line.split(",") | 1 | n | n |
| if length of tokens < 2 | 1 | n | n |
| print "Error: Invalid line format - " + line | 1 | 0 | 0 |
| continue | 1 | 0 | 0 |
| String courseNumber = tokens[0].trim() | 1 | n | n |
| String name = tokens[1].trim() | 1 | n | n |
| List<String> prerequisites = new List<String>() | 1 | n | n |
| courseNumbers.add(courseNumber) | 1 | n | n |
| for i from 2 to length of tokens - 1 | 1 | m \* n | m \* n |
| prerequisites.add(tokens[i].trim()) | 1 | m \* n | m \* n |
| Course course = new Course(courseNumber, name, prerequisites) | 1 | n | n |
| courses.add(course) | 1 | n | n |
| for course in courses | 1 | n | n |
| for prerequisite in course.prerequisites | 1 | m \* n | m \* n |
| if not courseNumbers.contains(prerequisite) | 1 | m \* n | m \* n |
| print "Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " does not exist." | 1 | 0 | 0 |
| file.close() | 1 | 1 | 1 |
| return courses | 1 | 1 | 1 |
| **Total Cost** |  |  | 6n + 3mn + 7 |
| **Runtime** |  |  | O(n + mn) |
| Since m is constant (the number of courses in the system will remain the same for the most part, we can consider this runtime complexity as O(n)). | | | |

### **Hash Table Implementation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| hashTable = new HashTable(100) | 1 | 1 | 1 |
| courseNumbers = createEmptyList() | 1 | 1 | 1 |
| for each line in lines | 1 | n | n |
| tokens = split(line, ",") | 1 | n | n |
| if length(tokens) < 2 | 1 | n | n |
| print("Error: Invalid format in line: " + line) | 1 | 0 | 0 |
| exit(1) | 1 | 0 | 0 |
| courseNumber = tokens[0] | 1 | n | n |
| name = tokens[1] | 1 | n | n |
| prerequisites = tokens[2:] | 1 | n | n |
| course = new Course(courseNumber, name, prerequisites) | 1 | n | n |
| hashTable.insert(course) | 1 | n | n |
| courseNumbers.append(courseNumber) | 1 | n | n |
| for each course in hashTable | 1 | n | n |
| for each prereq in course.prerequisites | 1 | m \* n | m \* n |
| if prereq not in courseNumbers | 1 | m \* n | m \* n |
| print("Error: Prerequisite " + prereq + " for course " + course.courseNumber + " does not exist.") | 1 | 0 | 0 |
| exit(1) | 1 | 0 | 0 |
| **Total Cost** |  |  | 7n + 2mn + 2 |
| **Runtime** |  |  | O(n + mn) |
| Since m is constant (the number of courses in the system will remain the same for the most part, we can consider this runtime complexity as O(n)). | | | |

### **Binary Search Tree:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| BinarySearchTree courseTree | 1 | 1 | 1 |
| file = open(fileName) | 1 | 1 | 1 |
| if file exists | 1 | 1 | 1 |
| while not end of file | 1 | n | n |
| line = readline(file) | 1 | n | n |
| courseData = parseLine(line) | 1 | n | n |
| if courseData is valid | 1 | n | n |
| courseNode = createCourseNode(courseData) | 1 | n | n |
| courseTree.insertCourseNode(courseNode) | 1 | n | n log n |
| else | 1 | 0 | 0 |
| // Handle file format error | 1 | 0 | 0 |
| close(file) | 1 | 1 | 1 |
| else | 1 | 0 | 0 |
| // Handle file not found error | 1 | 0 | 0 |
| **Total Cost** |  |  | 5n + n log n + 5 |
| **Runtime** |  |  | O(n log n) |