Big O Calculations

1. Vector
   1. Step 1
      1. O(1) + O(1) = O(2)
   2. Step 2
      1. O(1)
   3. Step 3
      1. O(N) +O(1) + O(1) = O(N)
   4. Step 4
      1. O(1) + O(1) +O(N) = O(N)
   5. Step 5
      1. O(1) + O(1) + O(1) + O(1) = O(4)
   6. Step 6
      1. O(N) \* O(N) = O(N^2)
   7. Step 7
      1. O(N)\* O(N) +O(1) +O(3) = O (N^2)
   8. Step 8
      1. O(1)
   9. Total
      1. O(2) + O(1) + O(N) +O(N) + O(4) +O(N^2) + O(N^2) = O(3+2N + 4 + 2N^2) = O(7 + 2N + 2N^2) = N^2
2. Hash
   1. Step 1
      1. O(1) + O(2) = O(3)
   2. Step 2
      1. O(1) + O(1) + O (1) + O(1) + O(1) = O(5)
   3. Step 3
      1. O(N) +O(1) + O(1) = O(N)
   4. Step 4
      1. O(1) + O(1) +O(N) = O(N)
   5. Step 5
      1. O(N)\* O(N) +O(1) +O(3) = O (N^2)
   6. Step 6
      1. O(1)
   7. Total
      1. O(3) + O(5) + O(N) + O(N) + O (N^2) = O (8 + 2N + N^2) = N^2
3. BST
   1. Step 1
      1. O(1) + O(2) = O(3)
   2. Step 2
      1. O(1)
   3. Step 3
      1. O(N) +O(1) + O(1) = O(N)
   4. Step 4
      1. O(1) + O(1) +O(N) = O(N)
   5. Step 5
      1. O(1) + O(1) + O(1) + O(1) = O(4)
   6. Step 6
      1. T(N) = O(1) + T(N/2) = O (log N)
   7. Step 7
      1. O(N^2) +O(1) = O(N^2)
   8. Step 8
      1. T(N) = O(1) + T(N-1) == O(N)
   9. Step 9
      1. O(1)
   10. Total
       1. O(1) + O(N) + O(N) + O(4) O(N^2) + O(1) + O(1) = O (7 + 2N + N^2) = N^2

Per my analysis the cost per line is very close between all 3 options, however Hash tables did more in fewer lines of code but with the most overhead both in speed and with the inherent flaw of needing more memory space to overcome the collision of keys caused by the fundamental nature of how hashing works. The most complicated by far was the binary search tree. One of the reasons the search tree was complicated was due to added features like recursion, which many would say is a feature not a bug. The recursion while it complicated the code was elegant in its own way and seems to fit better with the goal of Object-Oriented Programming allowing the writing of one piece of code and simply referring to it repeatedly. The search trees also have a great strength and weakness that of the logical flow. You need to get to d you have to go through a, then b, then c. But you always can find d by its value and those around it.

Vectors were easy to implement reasonably fast and easy to sort. They also have an index that does not require tons of overhead or collisions. And nesting vectors is relatively easy. That is why I am recommending vectors. Their speed of creation, inherent index, lower overhead than hash tables, ease of addition at any point, and user friendliness make them an obvious choice.

1. Main Menu

INTRODUCE choice(integer)

choice = 0﻿

WHILE choice != 9

DISPLAY “Main Menu”

DISPLAY “1. Load Data File”

Display “2. Print all courses”

DISPLAY “3. Print the title and prerequisites for an individual course”

DISPLAY “9. Exit”

DISPLAY “Enter Choice”

INPUT choice

SWITCH (choice)

CASE 1

loadCourses(courses.txt)

BREAK

CASE 2

printAllCourses(courses)

BREAK

CASE 3

searchCourses(courses)

BREAK

CASE 4

END WHILE

DISPLAY “Goodbye”

RETURN 0

1. Print All Courses Functions
   1. Vector
      1. Sort All Courses Function

FUNCTION sortAllCourses (courses)

INPUT: Courses (vector)

OUTPUT: Course Information printed

BOOL sortCourses(Course A, Course B)

Return a.courseNumber < b.courseNumber

SORT(courses.begin, courses.end, sortCourses())

* + 1. Print All Courses

FUNCTION printAllCourses (courses)

INPUT: sortCourses()

OUTPUT: Course Information printed

sortedCourses = sortAllCourses (courses)

FOR (const auto& courseData: sortedCourses)

FOR (string element: courseData)

PRINT element

END FOR

END FOR

* 1. Hash Table
     1. Hash Selection Sort

FUNCTION HashCourseSort (courses)

INPUT: Courses

OUTPUT: Course Information printed

FOR (All courses i)

FOR (All courses j)

IF (courses[j].number < courses[indexSmallest].number)

indexSmallest = j

END IF

END FOR

END FOR

* + 1. Print All Courses

FUNCTION printAllCourses (courses)

INPUT: HashCourseSort()

OUTPUT: Course Information printed

sortedCourses = HashCourseSort (courses)

FOR (const auto& courseData: sortedCourses)

FOR (string element: courseData)

PRINT element

END FOR

END FOR

* 1. Binary Search Tree

FUNCTION printAllCourses (courses)

INPUT: File Tree, Courses

OUTPUT: Course Information printed

IF (node is NOT NULL)

printAllCourses(node.left)

PRINT course Information

printAllCourses(node.right)

                END IF

END FUNCTION