ERIN UTHE

Display Menu() {

Print out option 1

Print out option 2

Print out option 3

Print out option 9

} (Runtime:4)

Menu(){

Display Menu(): =4

Get menu option =1

If option one: =1

Funtion to open/read file =(12n+6)

If option 2: =1

Print courses() = (2n)

If option 3: =1

Print courseInformation() = (4n)

If option 9: =1

Print goodbye message = (1)

Break/exit program =(1)

} (Runtime: 18n + 17)

PrintAlphanumerical order(tree.root) {

If node is null: =1

return =1

Print(node) =n

Go to left child node =n

Go to right child node =n

} (Runtime: 2n+ 2)

A vector has a runtime of O(n), where n is the number of elements. Vectors have a linear search and allows direct indexing if the index of the element is known. Vectors are easy to edit and resize, however they are difficult to perform insertion and deletion within the middle of the vector rather than the front/end and when dealing with large datasets vectors may need to be resized often causing redundant coding, on top of the already present high usage of memory. Hashtables also have a runtime of O(n), however, hashtables are more suitable for large datasets with an easy way to insert and delete data within the hashtable buckets, along with this comes the easy access to elements. However, with hashtables, once a dataset becomes filled, if the dataset is larger, this can cause more collisions within the hashtable causing the runtime to increase. Hashtables also do not order elements nor does a hashtable accept NULL data. Finally the binary search tree has the fastest runtime so far with a O(n logn) runtime. Binary trees are efficient for retrieving, deleting and inserting data into a node, size changes automatically and is not predefined, and binary search trees are also known for their recursive abilities, and using hierarchy. Although binary trees have the best runtime, if a tree becomes unbalanced, this can increase runtime of the tree from O(n logn) to O(n), it can be difficult to compare node information, and they also have a higher memory consumption.

Given the information of the data in the file to be loaded into the tables, I believe a binary search tree is the most appropriate fit. Due to the efficiency of manipulating the tree with additional nodes or node removal, I believe a binary search tree would load the data into the table most efficiently. Due to the file not having thousands of courses, but rather a few, this can give the program the most effective runtime while avoiding large memory consumption and the tree becoming unbalanced due to the large dataset.