|  |
| --- |
| Vector algorithm Runtime complexity bigO notation: O(N4)  Space complexity: S(N+2) |
| Hashtable algorithm Runtime complexity bigO notation: O(N3)  Space complexity: S(N+3) |
| BinaryTree algorithm Runtime complexity bigO notation: O(N2log(N))  Space complexity: S(N+2) |

Vectors - Vectors are a nice data structure when it comes to simplicity in structure and use. It doesn’t take a lot of code to search or iterate through a list. Adding items to the list can be done simply with a single command. It’s also simple to pull single items from the list.

The downsides to vectors are that searching for anything in a vector takes either a possibly long process of searching each item in the list one by one, or by several extra lines of code to search in a different pattern instead. Sorting through a vector list can also be annoying process since it involves both searching through and temporarily pulling the item out of the list and swapping it where it needs to go.

Hashtable – Hashtables are great for presorting and storing information. When you know the data beforehand it even becomes easy to keep everything in separate buckets. When it comes to inserting and searching through a hashtable it’s very likely you only need to do 1-3 comparisons.

Hashtables have the disadvantage of having issues when it comes to clashes. These clashes are resolved by making a list at each bucket, but this just adds on the sorting and maintaining of each list in each bucket.

BinaryTree – BinaryTree’s are another good structure for presorting and storing large amounts of data. Trees allow for much faster searching since it repeatedly cuts the list in half with each iteration through the tree. Insertion also makes sure that new data is put in the proper spot for numerical sorting. BinaryTree’s also have an overall faster runtime compared to Hashtables and vectors when it comes to large volume data lists.

BinaryTree’s have the distinct disadvantage of having an overly complicated removal process for inner nodes. They also require a good understanding of recursive calls to properly utilize the tree’s structure.

I personally would recommend the BinaryTree structure for this assignment. The tree structure should help with easy sorting of the classes as they’re being added to the tree. The search structure and overall better runtime should also help with the program speed and efficiency. The recursive calls for some of the functions should help minimize the amount of code needed for the insertion and call of items from the tree.