|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times**  **Executes** | **Total**  **Cost** |
| **Vector Load Course** |  |  |  |
| Initialize variables | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Read Line | 1 | n | 1 |
| Parse course data | 1 |  |  |
| Create course | 1 | n | n |
| Add prerequisites to course | 1 |  |  |
|  |  | Total cost | 4 |
|  |  | runtime |  |
|  |  |  |  |
| **Binary Search Tree Load Course** |  |  |  |
| Initialize variables | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Count lines in file | 1 |  |  |
| Resize BST | 1 | 1 | 1 |
| Read line | 1 | n | n |
| Declare variables | 1 | n | n |
| Parse course data | 1 |  |  |
| Create course | 1 | n | n |
| Insert course | O(n) | O(n) | O(n) |
| Return BST | 1 | 1 | 1 |
|  |  | Total cost |  |
|  |  | runtime |  |
|  |  |  |  |
| **Hash Table Load Course** |  |  |  |
| Initialize variables | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Set line count to zero | 1 | 1 | 1 |
| Read line | 1 | n | n |
| Count line | 1 | n | n |
| Resize hashtable to size of linecount | 1 | 1 | 1 |
| Declare row variable | 1 | 1 | 1 |
| Read line | 1 | n | n |
| Read parse data in line | 1 |  |  |
| Create course | 1 | n | n |
| Insert course into hashtable | O(n) | O(n) | O(n) |
| Return courses Hashtable | 1 | 1 | 1 |
|  |  | Total cost |  |
|  |  | runtime |  |

## **Data Structure Analysis**

As the above table shows, each data structure has a worse case runtime of . However, the vector data structure also has an average insert runtime of . A vector has the same runtime on average for searching, inserting, and deleting elements in that vector. Vectors are a very efficient way to store data. This is because a vector is a partitioned section of sequential memory. Therefore, every insertion and deletion must resize the vector meaning it creates a copy of every element in the vector with the desired modification. You can use an index to access an element in a vector with a constant runtime of . However, unless the index is known the vector must be searched for the desired element which on average is also a linear runtime of . is a linear runtime meaning the length of time a function takes to execute is linearly proportional to the size of data that function interacts with. If the size of the vector doubles this causes the runtime for an insertion, search, or deletion operation to double. A quick sort operation for a vector has a time complexity of which is noticeably slower than a linear runtime.

A Binary Search Tree (BST) has a worse case insertion, deletion, and search runtime of . However, the average runtime of these functions in a BST is . Therefore, the runtime doubles when the size of data is squared, . This results in a very slow growth of runtime and exceedingly fast manipulation and querying of data. This is because a BST is constructed with nodes storing data and pointing to other nodes. The BST uses a value system to decide if a new node is more or less than each node until it reaches the bottom of the tree and then inserts the new node. This results in more memory consumption to hold datasets. In worst case scenarios a BST can become a linked list if the data is inserted in order of lowest value to high or reversed order. The result is a Linked list which changes the runtime to worst case scenario and consumes more memory than a vector. Therefore, creating a BST in a randomized order is crucial to achieving an efficient and balanced Binary Search Tree. Because of the nature of BST retrieving elements in order is easily accomplished by traversing the tree from left to right which results in a runtime of .

A Hash Table is a vector surrounded with logic to manage insertions, deletions and searches. The logic for a Hash Table primarily computes a key from a portion of the data to create an index to choose where that data will be stored in the vector “bucket”. Meaning on average insertion, deletion, and search operations have a runtime of , meaning all processes take place at a constant regardless of data size. This is extremely useful for large sets of data. However, collisions can happen if the key computation does not evenly distribute data elements. This hash table handles collisions with a linked list. In a worse case runtime, all elements are indexed in the same bucket which results in a linked list data structure with a runtime for the three general operations. Because of the logic behind a Hash Table the data structure does require more space than a simple vector. Also, since a vector is the underlying data structure of a Hash Table it must be sorted the same way. However, changing the index of an element in the vector would cause improper queries of the hash table sorting the vector can only be accomplished in a copy which would require more almost double the amount of memory.

## **Recommendation**

A Binary Search Tree’s runtime for sorting, inserting, deleting, and searching operations is best suited for the academic advisors in the Computer Science department at ABCU. The BST has the best sorting runtime. With careful implementation logic a well-balanced BST can be created. Because the sorting operation is built into the data structure memory is conserved. The memory requirements for a BST are still well within modern system requirements.