****Run Time Analysis****

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Binary Tree |
| Loading data | 0(1) | 0(1) to 0(N) | 0 (log N) |
| Search | 0(n) | 0(1) to 0(N) | 0 (log N) to 0(N) |
| Sort/Print | 0(N log N) | 0(N) | 0(N) |

All three data structures have their own advantages and disadvantages. Loading data into unsorted vector using an append method is incredibly fast but sorting is later has slowest performance.

In theory, a hash table can achieve an average constant-time complexity of 0(1) for all its operations if hash function distributes the keys uniformly across the hash table and there are no collisions. However, this is practically impossible to achieve because hash tables have a limited amount of memory, and the number of elements to be stored in the hash table can exceed its capacity. Therefore, a hash table must handle some collisions to avoid exceeding its capacity, which may push the time complexity to 0(n) in the worst-case scenario, where n is the number if elements in hash table.

The height of a binary tree determines the time complexity of operations. The height of a balanced binary tree is O(log N), where N is the number of nodes. If the binary tree becomes heavily unbalanced, such as when sorted data is loaded, the height of the tree increases, and the time complexity of operations can degrade to O(N), where N is the number of nodes in the tree. This is because the unbalanced tree will resemble a linked list, and operations will need to traverse the entire length of the tree to reach the desired node. To maintain the consistent time complexity, it is essential to keep the binary tree balanced. This can be achieved by using self-balancing binary search trees, that automatically adjust the tree’s structure to maintain balance.

Which data structure to choose depends on how data will be accessed. If the data only needs to be loaded infrequently, there are no advantages after the initial load. If data needs to be searched often, the hash table could be better than binary tree assuming an efficient and well designed hash function.