# **Tabular representation of the complexities of various data structures**

| Operation | Data Structure | Add | Search | Insert | Update | Delete | Space Complexity |
| --- | --- | --- | --- | --- | --- | --- | --- |
| List<T> | Dynamic Array | O(1) | O(n) | O(n) | 0(1) | 0(n) | 0(n) |
| Dictionary<TKey, TValue> | Hash Table | O(1) | O(1) | O(1) | O(1) | O(1) | O(n) |
| HashSet<T> | Hash Set | O(1) | O(1) | O(1) | N/A | O(1) | O(n) |
| Queue<T> | Queue | O(1) | O(n) | O(1) | N/A | O(1) | O(n) |
| Stack<T> | Stack | O(1) | O(n) | O(1) | N/A | O(1) | O(n) |
| LinkedList<T> | Linked List | O(1) | O(n) | O(1) | O(1) | O(1) | O(n) |

- O(1) for dynamic arrays, hash tables, and linked lists assumes typical scenarios and average cases. In some cases, there might be resizing or rebalancing operations that can lead to O(n) time complexity.

- For searching, the complexity is O(n) for lists and linked lists because they are linear data structures. Hash tables and sets can achieve O(1) on average.

- For updating and deleting, dynamic arrays and linked lists can have better performance compared to hash tables for specific scenarios (e.g., when dealing with small datasets or in-order traversal is required).

- Space complexity refers to the additional space required for storing the data structure. It doesn't include the space required for the actual data being stored.

It's important to choose the right data structure based on the specific requirements of your application. The "faster" data structure depends on the specific operations and usage patterns in your use case. Additionally, Big-O notation provides an upper bound on the complexity, and actual performance can be influenced by various factors, including the size of the dataset and the specific implementation details.