|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Addition** | **Search** | **Deletion** | **Access by index** |
| **Array (T[])** | O(n) | O(n) | O(n) | 1 |
| **Dynamic array (List)** | O(1) | O(n) | O(n) | O(n) |
| **Stack** | O(1) | O(n) | O(1) | O(n) |
| **Queue** | O(1) | O(n) | O(1) | O(1) |
| **Dictionary** | O(1) | O(1) | O(1) | O(1) |
| **Sorted Dictionary** | O(log n) | O(log n) | O(log n) | O(log n) |
| **HashSet** | O(1) | O(1) | O(1) | O(1) |

When taking a look at the table, it is evident that both the ArrayT[] and the dynamic array provide constant time when it comes to access by index, but ultimately are slower for addition, search, and deletion. When taking a look at the table, it is also clear that there is a massive difference between the dictionary and the sorted dictionary. It is clear that dictionary is much quicker and it is ideal when performance is critical, since it has O(1) for addition, search, deletion, and access by index. Furthermore, hash sets have the fastest performance alongside dictionaries, having the same time complexity.

From the perspective of functional perspective, dictionaries with their average-case constant time complexity of O(1) provide efficient key-value management, which is ideal for functional programming tasks like mapping, grouping, and filtering. Furthermore, immutable variants also enable efficient updates while preserving immutability.