The Time Complexity and Space Complexity of Data Structures

Data Structure	Update	Insert	Delete	Add
List	O(1) for random access, O(n) for sequential access	O(1) for pre-allocated space, O(n) for reallocation	O(1) for random access, O(n) for sequential access	O(1)
Hash Set	O(1)	O(1)	O(1)	O(1)
Sorted Set	O(log n)	O(log n)	O(log n)	O(log n)
Stack	O(1) for push and pop operations, O(n) for peek operation	O(1)	O(1)	O(1)
Queue	O(1) for enqueue and dequeue operations, O(n) for front operation	O(1)	O(1)	O(1)
Linked Lists	O(1) for random access, O(n) for sequential	O(1)	O(1)	O(1)

Data Structure	Update	Insert	Delete	Add
	access			
Dictionary	O(1)	O(1)	O(1)	O(1)

List

A list is a collection of elements that are ordered by their position in the list. Lists are efficient for access, insertion, and deletion when the elements are accessed by their position in the list (i.e., random access). However, they could be more efficient for access, insertion, and deletion when the elements are accessed sequentially (i.e., sequential access).

Hash Set

A hash set is a collection of elements stored in a hash table. Hash sets are efficient for all operations, including access, insertion, and deletion. This is because the elements are stored in a hash table, allowing constant access to any element in the set.

Sorted Set

A sorted set is a collection of elements that are stored in a sorted order. Sorted sets are efficient for access and deletion but inefficient for insertion. This is because the elements are stored in a sorted order, which requires the elements to be rebalanced after an insertion.

Stack

A stack is a collection of elements ordered by LIFO (Last In, First Out). Stacks are efficient for push and pop operations but inefficient for peek operations. This is because the elements are stored in a stack, allowing constant access to the top element of the stack.

Queue

A queue is a collection of elements ordered by FIFO (First In, First Out). Queues are efficient for enqueue and dequeue operations but inefficient for front operations. This is because the elements are stored in a queue, allowing constant access to the front element of the queue.

Linked Lists

A linked list is a collection of elements stored in a linear order. Linked lists are efficient for access, insertion, and deletion when the elements are accessed by their position in the list (i.e., random access). However, they could be more efficient for access, insertion, and deletion when the elements are accessed sequentially (i.e., sequential access).

Dictionary

A dictionary is a collection of key-value pairs. Dictionaries are efficient for all operations, including access, insertion, and deletion. This is because the key-value pairs are stored in a hash table, which allows for constant access to any element in the dictionary.

Here is a table summarizing the time and space complexity of different data structures:

Data Structure	Time Complexity	Space Complexity
List	O(1) for random access, O(n) for sequential access	O(n)
Hash Set	O(1)	O(n)
Sorted Set	O(log n)	O(n)
Stack	O(1) for push and pop operations, O(n) for peek operation	O(n)

Data Structure	Time Complexity	Space Complexity
Queue	O(1) for enqueue and dequeue operations, O(n) for front operation	O(n)
Linked Lists	O(1) for random access, O(n) for sequential access	O(n)
Dictionary	O(1)	O(n)