Data Structure operations and time complexity:

Arrays:

Set, Check element at a particular index: O(1)

Searching: O(n) if array is unsorted and O(log n) if array is sorted and something like a binary search is used,

As pointed out by Aivean, there is no Delete operation available on Arrays. We can symbolically delete an element by setting it to some specific value, e.g. -1, 0, etc. depending on our requirements

Similarly, insert for arrays is basically Set as mentioned in the beginning

---

Linked List:

Inserting: O (1), if done at the head, O(n) if anywhere else since we have to reach that position by traversing the linked list linearly.

Deleting: O (1), if done at the head, O(n) if anywhere else since we have to reach that position by traversing the linked list linearly.

Searching: O(n)

Stack:

Push**:** O(1)

Pop**:** O(1)

Top**:** O(1)

Search **(Something like lookup, as a special operation):** O(n)

Queue/Deque/Circular Queue:

Insert**:** O(1)

Remove**:** O(1)

Size**:** O(1)

Binary Search Tree:

Insert, delete and search**: Average case:** O(log n)**, Worst Case:** O(n)

HashMap/Hashtable/HashSet:

Insert/Delete**:** O(1) **amortized**

Re-size/hash**:** O(n)

Contains**:** O(1)

Heap/PriorityQueue (min/max):

Find Min/Find Max: O(1)

Insert: O(log n)

Delete Min/Delete Max: O(log n)

Extract Min/Extract Max: O(log n)

Lookup, Delete (if at all provided): O(n), we will have to scan all the elements as they are not ordered like BST