Linear Search

**Approach 1: Using Iteration method**

This approach uses a for loop to iterate through the array arr from the beginning to the end.

Inside the loop, it checks if the current element arr[i] is equal to the target value.

If a match is found, it returns the index i.

If the loop completes without finding a match, it returns -1 to indicate that the target value is not present in the array.

**Space Complexity: The iterative approach has a constant space complexity of O(1) since it doesn't require any additional data structures proportional to the input size.**

**Time Complexity: The time complexity of the iterative linear search is O(n) in the worst case, where n is the size of the array.** This is because, in the worst case, the target value might be located at the last position or be absent, requiring a complete traversal of the array.

**Approach 2: Using Recursive method**

This approach uses a recursive function to perform the search.

It checks the base case, which occurs when the index reaches the size of the array. In this case, it returns -1 to indicate that the target is not found.

If the base case is not met, it checks if the current element arr[index] is equal to the target value.

If a match is found, it returns the index.

If a match is not found, it makes a recursive call to the function, passing the array, size, target, and an incremented index.

**Space Complexity: The recursive approach has a space complexity of O(n) in the worst case. This is because each recursive call adds a new frame to the call stack, which consumes memory.** In the worst case, the recursion depth could be equal to the size of the array.

**Time Complexity: The time complexity of the recursive linear search is also O(n) in the worst case.** Similar to the iterative approach, it may need to traverse the entire array to find the target value or determine its absence.