Binary Search

Approach 1: Using the Iteration method

This method uses a while loop to iteratively search for the target element within the given sorted array. It maintains two pointers, left and right, that represent the boundaries of the search space. The middle element mid is computed at each step by averaging the values of left and right. The search space is continuously divided by adjusting left or right based on the comparison of arr[mid] with the target value.

Time Complexity:

The Iterative approach makes the search space is halved with each comparison, resulting in a time complexity of O(log n), where n is the size of the input array. This makes binary search an efficient algorithm for searching in sorted arrays.

Time Complexity:

The iterative approach use a constant amount of extra space for variables, regardless of the input size. Therefore, **the space complexity for both approaches is O(1).**

Approach 2: Using Recursive method

This method utilizes recursion to divide the search space and perform binary search. It takes additional parameters low and high, representing the lower and upper bounds of the current search range. It recursively calls itself, adjusting low and high based on the comparison of arr[mid] with the target value. The base case for recursion is when low becomes greater than high, indicating that the target element is not found.

Time Complexity:

The Recursive approach makes the search space is halved with each comparison, resulting in a time complexity of O(log n), where n is the size of the input array. This makes binary search an efficient algorithm for searching in sorted arrays.

Time Complexity:

The Recursive approach use a constant amount of extra space for variables, regardless of the input size. Therefore, **the space complexity for both approaches is O(1).**