Binary Search Recursive approach

This C++ code implements the Binary Search algorithm to find a target element in a sorted array. Binary Search is an efficient search algorithm that repeatedly divides the search space in half, narrowing down the possible location of the target element until it is found or the subarray becomes empty.

Recursive function to calculate the sum of digits in a number

1. The code begins by defining the **binarySearch** function, which is a recursive function. It takes four arguments:
    - **arr**: A pointer to the array to be searched (assumed to be sorted in ascending order).
    - **start**: The starting index of the current range being searched.
    - **end**: The ending index of the current range being searched.
    - **target**: The element to be found in the array.
2. The function contains three main parts:
    - The base case: If the **start** index becomes greater than the **end** index, it means the target element is not found in the current range. In this case, the function returns **-1**.
    - The calculation of the **mid** index: It is determined as the average of **start** and **end** to divide the array into halves.
    - Two recursive cases: If the element at the **mid** index is equal to the **target**, the function returns the **mid** index. Otherwise, if the element at the **mid** index is greater than the **target**, the search is performed in the left half of the array (before the **mid** index). If it is less than the **target**, the search is performed in the right half of the array (after the **mid** index). The search space is adjusted accordingly in each recursive call.

Time Complexity:

- **The time complexity of the binary search function is O(log N), where N is the size of the input array. At each recursive call, the search space is reduced by half, leading to a logarithmic time complexity**. This makes binary search much more efficient than linear search, especially for large arrays.

Space Complexity:

- **The space complexity of the binary search function is O(log N), where N is the size of the input array. This is because each recursive call creates a new function call**

frame, and the recursion depth depends on the number of times the array can be divided in half.

Recursive Call stack for the Binary Search function