Binary Search Algorithm

Binary Search is a highly efficient algorithm used to find the position of a target element within a sorted array or list. By repeatedly dividing the search interval in half, Binary Search minimizes the number of comparisons needed to locate the target. This algorithm operates in O(logn)time complexity, making it much faster than linear search for large datasets.

How Binary Search Works

1. **Initial Setup**:
    - Start with two pointers, `low` and `high`, representing the bounds of the array. Initially, `low = 0` and `high = n - 1` (where n is the size of the array).
2. **Iterative Steps**:
    - Compute the middle index, `mid = (low + high) / 2`.
    - Compare the target value with the element at index `mid`:
        - If the target equals `arr[mid]`, the target is found, and the index `mid` is returned.
        - If the target is smaller than `arr[mid]`, narrow the search to the left half by setting `high = mid - 1`.
        - If the target is greater than `arr[mid]`, narrow the search to the right half by setting `low = mid + 1`.
3. **Termination**:
    - If `low` exceeds `high`, the target is not in the array, and the search returns an indicator (e.g., “not found”).

Algorithm (Pseudocode)

```
function binary_search(array, target):
    low ← 0
    high ← length(array) - 1

    while low ≤ high do:
        mid ← (low + high) // 2

        if array[mid] == target:
            return mid
        else if array[mid] < target:
            low ← mid + 1
        else:
            high ← mid - 1

    return -1  # Target not found
```

Key Features

1. **Preconditions**:
    - The array must be sorted before applying Binary Search.
2. **Time Complexity**:
    - Best case: O(1)(if the target is found at the middle index on the first attempt).
    - Average and worst case: O(logn) .
3. **Space Complexity**:
    - Iterative implementation: O(1) (no additional space required).
    - Recursive implementation: O(logn)(due to the recursion stack).

Example

Consider the sorted array `[1, 3, 5, 7, 9, 11, 13]` and the target value `7`:

1. Initial values: `low = 0`, `high = 6`, `mid = 3`. Compare `array[3]` with `7`.
2. Since `array[3] == 7`, the target is found at index `3`.

Applications

- Searching in large databases or datasets.
- Efficiently locating elements in sorted collections, such as dictionaries or phone books.
- Basis for many advanced algorithms and data structures (e.g., binary search trees).

Conclusion

Binary Search is a fundamental algorithm that combines simplicity and efficiency. Its logarithmic time complexity makes it ideal for scenarios where quick look ups in sorted data are necessary.
Understanding Binary Search is crucial for mastering algorithms and problem-solving in computer science.