Table of Contents

1. Introduction

Binary search is an efficient algorithm for finding a specific target value within a sorted list or array. It follows a divide-and-conquer approach by repeatedly dividing the search space in half until the target value is found or it determines that the target value does not exist in the list.

2. Algorithm

Here's a step-by-step description of the binary search algorithm:

1. Given a sorted list or array, determine the leftmost (`start`) and rightmost (`end`) indices of the search space.
2. Calculate the middle index as `mid = (start + end) // 2`.
3. Compare the value at the middle index with the target value:
   - If they are equal, the target value is found, and the algorithm can terminate.
   - If the value at the middle index is greater than the target value, update `end = mid - 1` to search in the left half of the list.
   - If the value at the middle index is less than the target value, update `start = mid + 1` to search in the right half of the list.
4. Repeat steps 2 and 3 until the target value is found or the search space is empty (`start > end`). In the latter case, the target value does not exist in the list.

Binary search has a time complexity of O(log n), where n is the size of the list or array. This makes it significantly faster than linear search, which has a time complexity of O(n) for an unsorted list.

3. Example Usage

An example implementation of the binary search algorithm in Python:

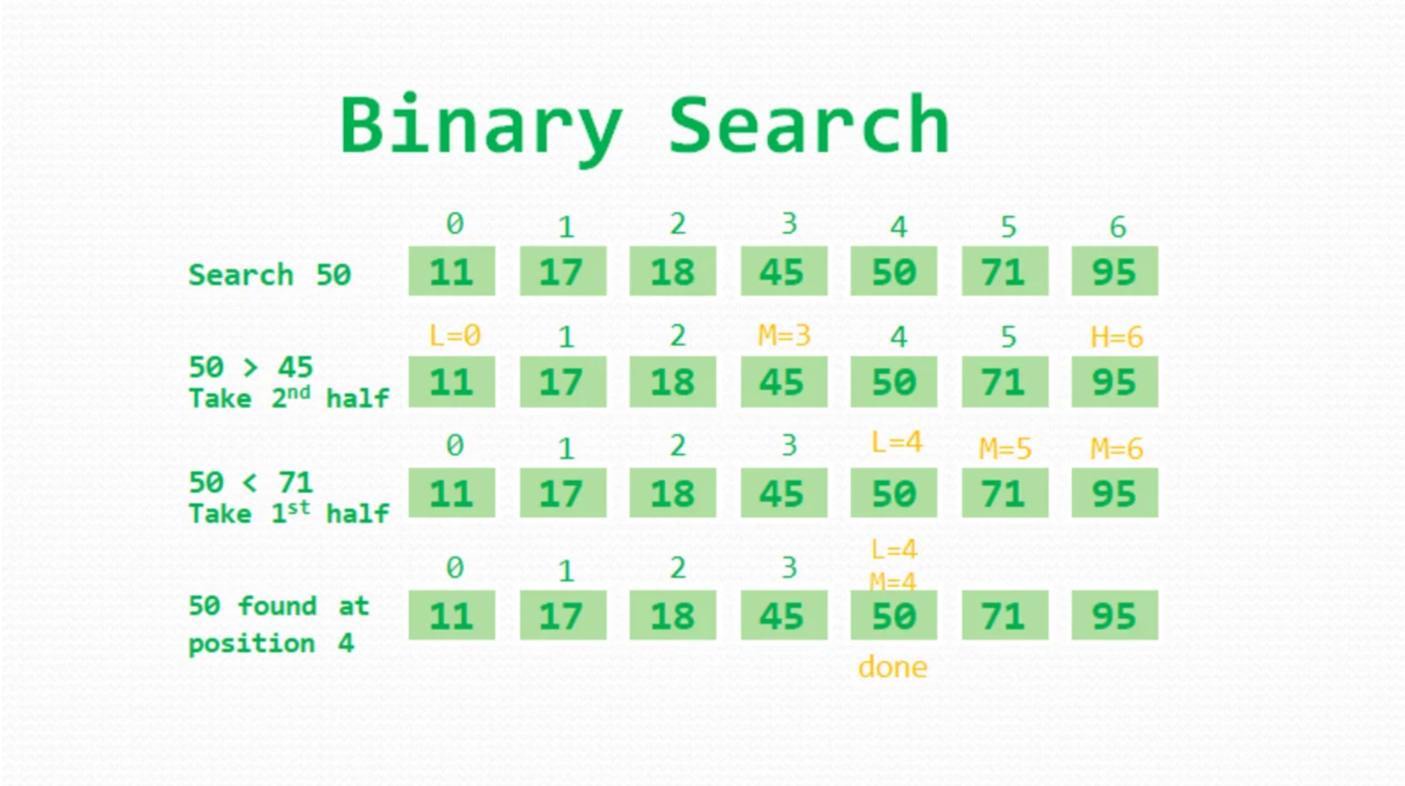
```python
def binary_search(nums, target):
    start = 0
    end = len(nums) - 1

    while start <= end:
        mid = (start + end) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            start = mid + 1
        else:
            end = mid - 1

    return -1  # Target value not found
```

We can use this `binary_search` function to search for a target value in a sorted list or array. It will return the index of the target value if found, or -1 if the target value is not present in the list.

4. Advantages

Using a heap binary search algorithm offers several advantages:

1. **Efficiency**: Binary search is highly efficient with a time complexity of O(log n). This is much faster than linear search, especially for large datasets.
2. **Simplicity**: The algorithm is relatively simple to understand and implement.
3. **Versatility**: Binary search is not limited to arrays; it can be adapted for use with other data structures like trees.
4. **Optimal for Sorted Data**: Binary search is particularly effective when working with sorted data since it exploits the ordered nature of the elements.
5. **Reduced Number of Comparisons**: Binary search eliminates half of the remaining elements in each step, reducing the number of comparisons needed to find the target value compared to linear search.


Binary Search
0 1 2 3 4 5 6
Search 50
11 17 18 45 50 71 95
L=0 1 2 M=3 4 5 H=6
50 > 45
Take 2nd half
11 17 18 45 50 71 95
0 1 2 3 L=4 M=5 M=6
50 < 71
Take 1st half
11 17 18 45 50 71 95
0 1 2 3 L=4 M=4
50 found at position 4
11 17 18 45 50 71 95
done