Binary Search

Thursday, December 25, 2025 6:05 PM

What is it?

A highly efficient searching algorithm that uses a "Divide and conquer" strategy.

Pre-requisite:

The list/array must be sorted (ascending (or descending) before applying this algorithm.

Method:

It repeatedly divides the search interval in half to narrow down the possible location of the target.

How it Works

1. **Find the middle**: calculate the middle index of the current range: $\boxed{mid = (low+high)//2}$
2. **Compare**: check the element at the mid against the target value.
   - **Case A (Match)**: If $\boxed{list[mid] == target}$, return the mid index. Stop.
   - **Case B (Too high)**: If $\boxed{list[mid] > target}$, discard the right half. move to the left half $\boxed{high = mid-1}$
   - **Case C (Too low)**: If $\boxed{list[mid] < target}$, discard the left half. move to the right half $\boxed{low = mid+1}$
3. **Repeat**: Continue steps 1-2 until the target is found.
4. **Not found**: If the low index becomes greater than the high index, the target is not in the list.

Algorithm Complexity (Big O)

- **Time Complexity**:
  - **Best case**: $O(1)$
    The target is exactly at the middle index on the first try.
  - **Worst case**: $O(\log n)$
    The target is at the ends (or) not present.
  - **Average case**: $O(\log n)$

- **Space Complexity**:
  - **Iterative Implementation**: $O(1)$ (constant space)
  - **Recursive Implementation**: $O(\log n)$ (due to stack space).

Key Characteristics

- **Efficiency**: Extremely fast for large datasets. Searching a dictionary of 1,000,000 words takes only ~20 comparisions max.
- **Constraints**: Can't be used on unsorted data.
- **Access**: Requires random access (like an array); not efficient for linked lists.