# Linear Search

## Time Complexity Analysis

The time complexity of linear search varies based on the position of the key in the list:

- **Best Case**: O(1)- The key is found at the first position.
- **Average Case**: O(n) - On average, the key is found halfway through the list.
- **Worst Case**: O(n) - The key is at the last position or not present at all.

# Binary Search

## Time Complexity Analysis

Binary search has different time complexities depending on the scenario, but it is generally efficient compared to linear search for sorted lists.

- **Best Case**: O(1)- The key is located at the middle of the list in the first comparison.
- **Average Case**: O(logn)- Binary search repeatedly divides the list, so the average number of comparisons is proportional to logn\log nlogn.
- **Worst Case**: O(logn)- In the worst case, binary search will check each division until the list is reduced to one element, which takes approximately  nlogn steps.

# Comparison between Linear Search and Binary Search

| **Criterion** | **Linear Search** | **Binary Search** |
|---|---|---|
| **Time Complexity** | Best: O(1), Avg/Worst: O(n) | Best: O(1), Avg/Worst: O(logn) |
| **Space Complexity** | O(1) (iterative) | O(1) (iterative), O(logn)recursive) |
| **Efficiency on Large Datasets** | Works on unsorted data | Requires data to be sorted |
| **Efficiency on Large Datasets** | Inefficient, scales linearly with n | Efficient, scales logarithmically with n |
| **Best Use Case** | Small or unsorted datasets | Large, sorted, and mostly static datasets |