Searching and Sorting

Searching:

Searching refers to the process of finding a specific element in a collection of data.

- **Linear Search:**
  - **Description:** Linear search, or sequential search, involves scanning the elements of a collection one by one until the desired element is found or the end of the collection is reached.
  - **Time Complexity:** O(n) in the worst case, where "n" is the number of elements in the collection.
- **Binary Search:**
  - **Description:** Binary search is applicable to sorted collections. It repeatedly divides the search interval in half and compares the middle element with the target, narrowing down the search until the target is found or the search space is empty.
  - **Time Complexity:** O(log n) in the worst case for a sorted collection, where "n" is the number of elements.

Difference Between Linear Search and Binary Search:

Feature	Linear Search	Binary Search

Nature of Data	Works on both sorted and unsorted arrays/lists.	Requires a sorted array/list.
Algorithm Type	Sequential search algorithm.	Divide and conquer algorithm.
Approach	Scans elements one by one until the target is found or the end is reached.	Divides the search space in half at each step.
Time Complexity	Worst Case: O(n) - Linear time complexity.	Worst Case: O(log n) - Logarithmic time complexity for a sorted array.
Best Case	O(1) - If the target is found at the beginning.	O(1) - If the target is found at the middle.
Average Case	O(n/2) - On average, the target is found in the middle.	O(log n) - On average, the target is found in log2(n)log2(n) comparisons.
Use Cases	Suitable for small datasets or unsorted collections.	Efficient for large sorted datasets.
Ease of Implementation	Easier to implement.	Requires a sorted dataset but is relatively straightforward to implement.

Sorting:

Sorting is the process of arranging elements in a particular order, typically in ascending or descending order based on certain criteria.

1. **Bubble Sort:**
    - **Description:** Iteratively compares adjacent elements and swaps them if they are in the wrong order. The process is repeated until the entire list is sorted.

- **Time Complexity:** O(n^2) in the worst case.

2. **Selection Sort:**
   - **Description:** Divides the list into a sorted and an unsorted region. In each iteration, the minimum (or maximum) element from the unsorted region is selected and moved to the sorted region.
   - **Time Complexity:** O(n^2) in the worst case.
3. **Insertion Sort:**
   - **Description:** Builds the sorted list one element at a time by repeatedly taking elements from the unsorted part and inserting them into their correct position in the sorted part.
   - **Time Complexity:** O(n^2) in the worst case.
4. **Merge Sort:**
   - **Description:** Divides the list into two halves, recursively sorts each half, and then merges them back together. It is a divide-and-conquer algorithm.
   - **Time Complexity:** O(n log n) in the worst case.
5. **Quick Sort:**
   - **Description:** Chooses a pivot element and partitions the list into two sublists, elements smaller than the pivot and elements greater than the pivot. Recursively applies the same process to the sublists.
   - **Time Complexity:** O(n^2) in the worst case (rarely occurs), but O(n log n) on average.

Differences between Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort:

Feature	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort

Algorithm Type	Comparison-based Sorting Algorithm	Comparison-based Sorting Algorithm	Comparison-based Sorting Algorithm	Comparison-based Sorting Algorithm (Divide and Conquer)	Comparison-based Sorting Algorithm (Divide and Conquer)
Approach	Iterative	Iterative	Iterative	Recursive (Divide and Conquer)	Recursive (Divide and Conquer)
Worst Time Complexity	O(n^2)	O(n^2)	O(n^2)	O(n log n)	O(n^2) (rarely occurs, but possible)
Average Time Complexity	O(n^2)	O(n^2)	O(n^2)	O(n log n)	O(n log n)
Best Time Complexity	O(n)	O(n^2)	O(n)	O(n log n)	O(n log n)
Space Complexity	O(1)	O(1)	O(1)	O(n) (additional space for merging)	O(log n) (additional space for recursive calls)
Stability	Stable (equal elements maintain their relative order)	Unstable (equal elements may change their order)	Stable (equal elements maintain their relative order)	Stable	Unstable
Adaptability	Can be adaptive with optimized implementations.	Not adaptive.	Can be adaptive with optimized implementations.	Not adaptive.	Can be adaptive with optimized implementations.