An overview of some commonly used sorting algorithms:

1. Bubble Sort:

- Compares adjacent elements and swaps them if they are in the wrong order.
- Repeatedly passes through the list until the list is sorted.
- Simple to implement but not efficient for large lists.
- Time Complexity: O(n^2)

2. Selection Sort:

- Divides the list into a sorted and an unsorted portion.
- Selects the smallest (or largest) element from the unsorted portion and swaps it with the first element of the unsorted portion.
- Repeats this process until the entire list is sorted.
- Time Complexity: O(n^2)

3. Insertion Sort:

- Builds the final sorted list one element at a time.
- Takes each element from the unsorted portion and inserts it into the correct position in the sorted portion.
- Time Complexity: O(n^2), but efficient for small lists or partially sorted lists.

4. Merge Sort:

- Utilizes the divide-and-conquer approach.
- Divides the list into smaller sublists, recursively sorts them, and then merges them to

obtain the final sorted list.
- Efficient for large lists and guarantees a time complexity of O(n log n) in all cases.
- Requires additional space for the merging step.

5. Quick Sort:

- Utilizes the divide-and-conquer approach.
- Selects a pivot element, partitions the list into two sublists (elements less than the pivot and elements greater than the pivot), and recursively sorts the sublists.
- Efficient for large lists, but can have a worst-case time complexity of O(n^2) if the pivot selection is unbalanced.
- In the average case, it has a time complexity of O(n log n).

6. Heap Sort:

- Utilizes a binary heap data structure to sort elements.
- Builds a max-heap from the input list and repeatedly extracts the maximum element to obtain the sorted list.
- Has a time complexity of O(n log n) in all cases and is an in-place sorting algorithm.
- Requires additional space for the heap structure.

7. Radix Sort:

- Sorts elements by processing them digit by digit.
- Groups elements based on the value of each digit using counting sort or any stable sorting algorithm.
- Requires a fixed-length representation for elements (e.g., integers or strings).
- Time complexity depends on the number of digits and the range of digits.