1. Question- **Analyse the worst-case and best-case Time & space complexity of the following algorithms. (Also add the Reason with it)**
    1. Linear Search.
    2. Binary Search.
    3. Bubble Sort.
    4. Selection sort.
    5. Insertion sort
    6. Merge sort.

    1. **Linear Search:**
        A. Best Case Time Complexity: O(1)
        - The best case is when the element being searched for is present at the beginning of the array, requiring only one comparison.

        B. Worst Case Time Complexity: O(n)
        - The worst case is when the element being searched for is present at the end of the array or not present at all, requiring n comparisons.

        C. Space Complexity: O(1)
        - Linear search doesn't require any extra space beyond the input array.

    2. **Binary Search:**
        A. Best Case Time Complexity: O(1)
        - The best case is when the element being searched for is at the middle of the sorted array, requiring only one comparison.

        B. Worst Case Time Complexity: O(log n)
        - The worst case is when the element being searched for is at either end of the sorted array, requiring log n comparisons.

        C. Space Complexity: O(1)
        - Binary search doesn't require any extra space beyond the input array.

    3. **Bubble Sort:**
        A. Best Case Time Complexity: O(n)
        - The best case is when the input array is already sorted, requiring only n comparisons and no swaps.

        B. Worst Case Time Complexity: O(n^2)
        - The worst case is when the input array is in reverse order, requiring n^2 comparisons and swaps.

        C. Space Complexity: O(1)
        - Bubble sort sorts the array in place and doesn't require any extra space beyond the input array.

    4. **Selection Sort:**
        A. Best Case Time Complexity: O(n^2)
        - The best case is the same as the worst case since the algorithm always makes n^2 comparisons and swaps.

B. Worst Case Time Complexity: O(n^2)
    - The worst case is when the input array is in reverse order, requiring n^2 comparisons and swaps.

C. Space Complexity: O(1)
    - Selection sort sorts the array in place and doesn't require any extra space beyond the input array.

5. **Insertion Sort:**
    A. Best Case Time Complexity: O(n)
        - The best case is when the input array is already sorted, requiring only n comparisons and no swaps.

    B. Worst Case Time Complexity: O(n^2)
        - The worst case is when the input array is in reverse order, requiring n^2 comparisons and swaps.

    C. Space Complexity: O(1)
        - Insertion sort sorts the array in place and doesn't require any extra space beyond the input array.
6. **Merge Sort:**
    A. Best Case Time Complexity: O(n log n)
        - The best case is the same as the worst case since the algorithm always makes n log n comparisons and requires n log n space to store the intermediate arrays.

    B. Worst Case Time Complexity: O(n log n)
        - The worst case is when the input array is in reverse order, requiring n log n comparisons and swaps.

    C. Space Complexity: O(n)
        - Merge sort requires extra space to store the intermediate arrays during the merging process. The space complexity is proportional to the size of the input array.