Assignment 1

**Time Complexity Analysis**:

* **Best-case time complexi**ty: O(n) - This occurs when the input array is already sorted. In this case, Insertion Sort performs very efficiently as it only needs to check each element once.
* **Average-case time complexity**: O(n^2) - In the average case, it involves nested loops to compare and shift elements.
* **Worst-case time complexity**: O(n^2) - This occurs when the input array is sorted in reverse order. In this case, Insertion Sort performs the slowest as it needs to compare and shift elements for each element in the array.

**Stability**:

Insertion Sort is a stable sorting algorithm, which means that it preserves the relative order of equal elements. If you have two equal elements in the input, their order will remain the same in the sorted output.

**Performance on Different Input Data**:

* Already sorted input: Performs well with a best-case time complexity of O(n).
* Reverse sorted input: Performs the slowest with a worst-case time complexity of O(n^2).
* Random data: Performs with an average-case time complexity of O(n^2). It is not the most efficient choice for large random datasets.

**Comparison with Merge Sort**:

Merge Sort has a better average and worst-case time complexity of O(n log n). It is more efficient than Insertion Sort, especially for larger datasets. Merge Sort is a stable sort as well. However, Merge Sort uses additional memory space for its operations, making it less memory-efficient compared to Insertion Sort.

**When to Prefer Insertion Sort or Merge Sort**:

* Use Insertion Sort when dealing with small datasets or nearly sorted data, as it has a good best-case performance.
* Use Merge Sort when dealing with larger datasets or when you need a stable sorting algorithm. Merge Sort is generally a better choice for general-purpose sorting.