Theoretical Time Complexity:

1. **Merge Sort**:
   - Average Case: O(n log n)
   - Worst Case: O(n log n)
   - Best Case: O(n log n)
2. **Quick Sort**:
   - Average Case: O(n log n)
   - Worst Case: O(n^2) (with a poor choice of pivot such as smallest or largest element)
   - Best Case: O(n log n)
3. **Heap Sort**:
   - Average Case: O(n log n)
   - Worst Case: O(n log n)
   - Best Case: O(n log n)

In terms of theoretical time complexity, Merge Sort, Quick Sort, and Heap Sort all have similar average and best-case time complexities of O(n log n), which are considered efficient for comparison-based sorting algorithms. However, there are differences in their worst-case time complexities and space complexities.

- **Merge Sort** has consistent O(n log n) time complexity in all cases.
- **Quick Sort** has an average and best-case time complexity of O(n log n), which is very efficient. However, in the worst case (rare scenarios with poor pivot choices), it degrades to O(n^2).
- **Heap Sort** also has an average and worst-case time complexity of O(n log n), making it efficient.

Experimental Data:

For an array of 12 elements sorted via various 3 different algorithms, The output of program Q3.c is as follows.

```
PS F:\Ashoka University\codes\DS\A3\Q3> notepad Q3a.c
PS F:\Ashoka University\codes\DS\A3\Q3> gcc .\Q3a.c
PS F:\Ashoka University\codes\DS\A3\Q3> .\a.exe
Given array is
8 19 9 11 12 11 13 5 6 7 3 17

Sorted array using Merge Sort is
3 5 6 7 8 9 11 11 12 13 17 19
Merge Sort: Copy count 55, Comparison Count 33

Sorted array using Quick Sort is
3 5 6 7 8 9 11 11 12 13 17 19
Quick Sort: Swap 28 Comparison 38

Sorted array using Heap Sort is
3 5 6 7 8 9 11 11 12 13 17 19
Heap Sort: Swap 72 Comparison 38
PS F:\Ashoka University\codes\DS\A3\Q3>
```

Merge Sort: Copy count 55, Comparison Count 33

Quick Sort: Swap 28 Comparison 38

Heap Sort: Swap 72 Comparison 38