1. Comparing the sorts at a general level, is one sort always better than the others?

No, because while the merge sort performs significantly better than the insertion sort and selection sort for larger lists, it performs way slower than the other two for smaller lists. In other words, the merge sort's time complexity is O(n log (n)) for all cases, whereas the insertion sort's time complexity is O(n) for the best case (better for smaller lists) and O(n^2) for both average and worst cases (worse for bigger lists). Similar to insertion sort, selection sort also performs O(n^2) for larger lists. However, the merge sort might outperform the selection sort for smaller lists because the selection sort runs O(n^2) for all cases, including the best case because even if a list is sorted, it still has to iterate until the end.

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)?

The insertion sort is better than the other two because it compares one value to the previous value and if the previous value is not bigger than the current value, it will stop the while loop and move on to the next comparisons. On the other hand, the selection sort has to iterate through the entire list to search for minimum value, even if the list is sorted. Similarly, the merge sort still has to divide the entire list and put them all together.

3. You probably found that insertion sort has about half as many comparisons as selection sort. Why?

Because the selection sort always performs quadratic numbers of comparisons whereas the insertion sort performs quadratic numbers of comparisons in the average and worst cases. In the best case, it performs O(n). Also, the insertion sort is adaptive to data, unlike the selection sort, which moves each value at most once.

4. Given the above observation, why are the times for insertion sort not half what they are for selection sort? (For part of the answer, think about what insertion sort has to do more of compared to selection sort.)

What makes the selection sort inefficient compared to the insertion sort is that it has to search for the smallest value throughout the entire list. However, once the minimum value is found, it's instantly swapped with the first unsorted value. On the other hand, while the insertion sort takes lesser comparisons, it has to compare every previous value in order to properly insert the value in its place. This is why it takes a similar amount of time compared to the selection sort.