2) (5 pts) DSN (Sorting)

In both Merge Sort and Quick Sort, in class we are taught to break down the sorting problem recursively such that the base case is a subarray of size 1 (or 0). It turns out that for both, on average, the implementation is *faster* if we have a base case with a subarray of size in between 20 and 50 and use a $O(n^2)$ sort (typically insertion sort) to sort the base case subarray. Even though insertion sort is $O(n^2)$, why does this modification to the algorithm result in a speed up for both Merge Sort and Quick Sort?

There is a great deal of overhead with recursive calls. Namely, when a new function call is executed, memory is allocated for that function on the call stack and parameters are passed (actual values copied into formal parameter slots), then the function can start running. A vast majority of the total # of recursive calls in the call branches of either of these functions occurs for small arrays. For an array of size 32, at least 31 recursive calls get made. While for large arrays an insertion sort is slower than Merge or Quick sort, for small arrays, the insertion sort if faster because of the overhead of all of these recursive calls. Also, insertion sort only does quick local array accesses so though it does more steps, they are generally faster steps. Thus, if we make our base case larger, what we are doing is substituting something that is slower (a Merge Sort or Quick Sort of 30 or so values) with something that is faster (an Insertion Sort of 30 or so values). Naturally, if we have a set of steps in an algorithm and substitute some of those steps with faster ones, our new algorithm is faster. The key is to set this base case right near that tipping point of the optimal difference between the two sorts for small values.

Grading: There are quite a few ways to explain this that are valid. The crux of it is that for small arrays, the overhead of the recursion slows the algorithm down so much, it's slower than a simple sort that does more steps but does them without extra function calls and has quick array accesses. Making these substitutions speeds up the overall algorithm since we are substituting something slower for something faster. Give credit based on how complete and convincing the argument given is. Read several responses before calibrating the grading scale.