COP 3502 Suggested Problems/Program Edits: Sorting (Week 9)

1) Edit Bubble Sort so that if a whole iteration completes without making any swaps, then you don't run any more iterations and return.

2) Rewrite insertion sort so that the sorted list is built from the right side and not the left. So, for example, if the array first has:

3 2 8 1 6

It would look like this after each iteration:

3	2	8	1	6	(inserted 1 into list with 6)
3	2	1	6	8	(inserted 8 into list 1,6)
3	1	2	6	8	(inserted 2 into list 1,6,8)
1	2	3	6	8	(inserted 3 into list 1,2,6,8)

3) In selection sort the way it was taught in class, we place the maximum element of the array at the end first. Reverse the logic so that the first item that is selected is the smallest, the second item selected is the second smallest, etc.

4) Rewrite merge sort so that it splits the array into the left size that is one third (roughly) the size of the full array and the right side so that it is two thirds the size of the fully array. Run this version against the usual version and time the code on large arrays. Is this version faster or slower? Why?

5) Edit the posted file with quick sort and try a base case of various sizes ranging from 5 to 100, where in the base case, you sort the subarray using insertion sort. Time the various versions on large arrays. Which one works the best?

6) The median of 3 idea was discussed in lecture to pick a partition element for the partition portion of Quick Sort. Edit the posted Quick Sort to implement this idea.

7) The posted code for Quick Sort adds a call to the is_sorted function before the usual recursive code. Removing this slows down the execution of the code on an array with all the same values. Why?

8) The while loop in the merge function of the posted Merge Sort has an if-else inside of it. This is only two separate branches of code. Many students write this with more branches (either 3 or 4). Why might writing this code with more branches be less error prone? Usually, writing more code lends itself to more errors, but in this case, it's likely that separating out the logic into 4 branches reduces the chance of implementation error. Why?