2) (10 pts) ALG (Sorting)

(a) (6 pts) In quick sort, when running the partition function, the first step is to choose a random partition element. In some implementations, instead of just choosing a random element, 3 or 5 random elements are chosen and the median of those elements is then selected as the partition element, as opposed to making the partition element a single randomly selected item. What is the potential benefit of using this strategy (median of 3 or median of 5) versus the default strategy of just choosing a single random element?

The potential benefit of this strategy is that the median of 3 or 5 randomly selected items is very likely to be closer to the actual median of the data and this is precisely when Quick Sort will run faster. Quick Sort achieves its best case run-time if every partition element chosen is the median element and its run-time deteriorates as the partition element choices stray further from the median. In the worst case, the partition puts all elements either to the left or right of it, leaving an array almost as big (n-1 elements) to sort as it started with (n). Thus, by taking a bit of extra time up front to increase the probability of getting an element closer to the actual median of the array, Quick Sort has a better chance of achieving a run time closer to its best case. This trade-off is worth it when the array being sorted is very large. An optimized implementation may only choose to use the median of 3 or 5 strategy if it's sorting more than some number of elements.

Grading: A response does NOT need to be nearly as verbose as the one above. Give them full credit simply if they mention that the chance of getting something closer to the median of the array goes up by doing the median of 3 or 5 strategy. Decide partial credit as you see fit. You may want to read a few papers to get a gauge of the common answers students give before figuring out how much partial credit to give for certain types of responses.

(b) (4 pts) The best case run time of an insertion sort of n elements is O(n) and the worst case run time of an insertion sort is $O(n^2)$. Describe how to (a) construct a list of n distinct integers that, when sorted by insertion sort, gets sorted in the best case run time, and (b) construct a list of n distinct integers that, when sorted by insertion sort, gets sorted in the worst case run time.

We simply keep the list of integers in sorted order to achieve a best case run time. As insertion sort attempts to insert each new item, it will see that it's immediately in place and the inner loop won't ever run, allowing for the best case run time.

To achieve the worst case run time, just put each item in reverse sorted order. When insertion sort attempts to insert each new item, it will be forced to swap the new item with every single other previously inserted item, creating the worst case run-time.

(**Grading:** No need for explanation, 2 pts per case and all that needs to be mentioned is that the list needs to be sorted and reverse sorted, respectively.)