David Ng Questions 5, 6, and 7

Question 5

Total Runtime to Sort 1000 Random Arrays of Indicated Length

	Array of Length 128	Array of Length 1024	Array of Length 16384	Array of Length 65536
Insertion Sort	58.1	233 ms	17183 ms	281493 ms
Heap Sort	286 ms	307 ms	4003 ms	27869 ms
Quick Sort	219 ms	433 ms	35074 ms	510087 ms
Quick Sort Improved	213 ms	585 ms	97013 mss	1541546 ms
Java Sort Function	30.7 ms	61.4 ms	148 ms	466 ms

The results were fairly close to what I would expect. Java’s sorting algorithm is the fastest in every case tested. This is likely due to the fact that Java’s algorithm is an implementation of Timsort, which is a sorting algorithm which uses elements of both merge sort and insertion sort. Insertion sort is also faster in practice for smaller array sizes compared to heap sort and the two quick sort algorithms. When the array sizes increase, we can see that heap sort surpasses insertion sort from array size of 1024 to 16384. The results I did not expect were the fact that this implementation of quick sort was slower at increasing array lengths. The improved quick sort was even slower than quick sort, even though we had theoretically made improvements to the algorithm. This may be due to the fact that the cutoff point was could have been chosen to be a better value which would be more optimal in terms of deciding when to use insertion sort instead of continuing with quick sort. These disparities in what I expected to the actual results may be due to the fact that we had implemented quick sort using deterministic partitioning. While it seems contrary to my expectations that quick sort would be slower at these increasing array lengths, it is not theoretically implausible, since it may be the case that this particular implementation of quick sort is slower than better implementations.

Question 6

Total Runtime to Sort 1000 Sorted Arrays of Indicated Length

	Array of Length 128	Array of Length 1024	Array of Length 16384
Insertion Sort	49.5 ms	66.4 ms	574 ms
Heap Sort	187 ms	382 ms	4811 ms
Quick Sort	230 ms	642 ms	44293 ms
Quick Sort Improved	106 ms	479 ms	59630 ms
Java Sort Function	17.7 ms	16.7 ms	30.4 ms

We can see that quick sort functions slower than when the array was not already sorted. This matches closely with what I had expected, as theoretically, the worst case for quick sort is when the array is already sorted. We can see that as the array size increases, the time difference between the time it takes quick sort to sort the sorted array increases more and more from the time it takes quick sort to sort the random arrays. In these tests, it was seen that the improved quick sort improved drastically from the previous results when testing random arrays. This is most likely due to the fact that improved quick sort uses insertion sort when the array sizes are smaller. Thus, we are taking advantage of the fact that insertion sort performs better than quick sort for sorted arrays. Overall, we also saw a slight increase in the speeds of insertion sort, heap sort, and java's sorting algorithm. These results are consistent with what I had expected, as insertion sort performs best when the array is already sorted, in which case insertion sort has linear running time.

Question 7

A modification to quick sort that allows it to run in O(n log n) time in the worst case would require a selection algorithm for finding the medians. SELECT would takes an array A, the lower and upper bounds of the subarray in A, and the number i, which is (high-low+1)/2, and in linear time, returns the ith smallest element in A[p ..r].

For an n-element array, the largest subarray that the improved version of quick sort recurses on therefore has n/2 elements. This situation occurs when n = r – p + 1 is even; then the subarray A[q+1..r] has n/2 elements, and the subarray A[p..q–1] has n/2–1 elements. Because this version of quick sort always recurses on subarrays that are at most half the size of the original size, the recurrence for the worst-case running time is T(n) <= 2T(n/2) + (n) = O(n log n).

When testing the program, I would expect it to be slightly slower than the normal implementation of quick sort on random arrays, as it includes additional instructions to find the medians. However, when it is used on sorted arrays, I would expect it to be slightly faster, as it has worst case n log n compared to n^2 of quick sort.

References: Introduction to Algorithms, 3rd Edition. Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.