Michael Blackburn
CSCI 230
Dr. William Bares
December 9, 2013

Program 8 Search Analysis

A. Best Case Analysis for Selection Sort
   Best-case list of inputs: alpha, charlie, echo, golf, indigo, kilo, november, romeo, tango, victor
   Best-case Big-O for number of key comparisons: O (n) = 45
   Best-case Big-O for number of assignment operations: O (0) = 0
   Actual number of compares reported: 45
   Actual number of assignments reported: 0

For the first of the two basic sorts, we use a selection sort on a list of ten String elements. In our best-case scenario, the list is pre-sorted in ascending order. We would think our comparisons should only be made once for each element, but our actual comparisons reported end up being $\frac{10!}{8!2!}$, which represents the structure of the nested loops used to iterate through each element in the list. Since we must compare each element in the list to each successive element each time through, this is a constant factor for a 10-element list. Because the list is pre-sorted, no assignments take place in the code, and our result is consistent with this idea.

B. Worst Case Analysis for Selection Sort
   Worst-case list of inputs: victor, tango, romeo, november, kilo, indigo, golf, echo, charlie, alpha
   Worst-case Big-O for number of key comparisons: O (n) = 45
   Worst-case Big-O for number of assignment comparisons: O (n^2) = 100
   Actual number of compares reported: 45
   Actual number of assignments reported: 135

In the worst-case for selection sort, our list is in reverse order. Once again, because of our nested loop structure, we have a constant compare factor of 45, which is again what the program reports. Our assignments, however, exceed our expected result of 100. Based on the code, it appears that either this list is not an actual worst-case, or that our assumption about what our results should be are incorrect.

C. Best Case Analysis for Bubble Sort
   Best-case list of inputs: alpha, charlie, echo, golf, indigo, kilo, november, romeo, tango, victor
   Best-case Big-O for number of key comparisons: O (n) = 45
   Best-case Big-O for number of assignment operations: O (0) = 0
   Actual number of compares reported: 45
   Actual number of assignments reported: 0

In our best-case scenario for bubble sort, we again see a similar structure to the selection sort. Our nested loop structure provides a constant 45 compares to iterate through the list. Our list is again pre-sorted in ascending order. We once again have no assignments made, as all of the elements are sorted, with no need for any swaps throughout the list.

D. Worst Case Analysis for Bubble Sort
   Worst-case list of inputs: victor, tango, romeo, november, kilo, indigo, golf, echo, charlie, alpha
   Worst-case Big-O for number of key comparisons: O (n) = 45
   Worst-case Big-O for number of assignment comparisons: O (n^2) = 100
   Actual number of compares reported: 45
   Actual number of assignments reported: 135

In our worst-case scenario for bubble sort, we see the exact same output as we saw in selection sort. We have an appropriate number of compares at 45, but our worst-case list does not produce the expected results, and instead produces the same number of assignments as selection. These numbers are likely attributable to the code itself, as a countAssign variable is incremented by 3 each time a swap is made, and since our list for selection and bubble sort are the same, we should expect similar numbers, incorrect as they may be.

E. Worst Case Analysis for Quick Sort
   Best-case list of inputs: alpha, charlie, echo, golf, indigo, kilo, november, romeo, tango, victor
   Best-case Big-O for number of key comparisons: O (n^2) = 100
   Best-case Big-O for number of assignment operations: O (N^2) = 100
   Actual number of compares reported: 109
   Actual number of assignments reported: 72

In our worst-case quick sort, our list is pre-sorted in ascending order, giving a list where no assignments should be made. Our expected Big-O for the quick sort is of the (n log n) class, and for a list of 10 elements, that is about 23. Our actual reported comparisons, 109, far exceed this number, and this is also likely attributable to both our recursive function(s) used for the quick sort and our placement of countCompare throughout the function. Likewise, since our list is in order, there should be no reported assignments; since we also increment countAssign for other assignment operations in the function(s) besides the assignment swaps, this likely is the reason we have reported assignments. However, with the list presorted, and the pivot value selected as the low element in the list, the pre-sorted list is actually closer to the worst-case scenario versus the best, as we might think it should be.

F. Best Case Analysis for Quick Sort
   Best-case list of inputs: romeo, november, tango, victor, kilo, indigo, golf, alpha, charlie, echo
   Worst-case Big-O for number of key comparisons: O (n log n) = about 23
   Worst-case Big-O for number of assignment comparisons: O (n^2) = 100
   Actual number of compares reported: 78
   Actual number of assignments reported: 55

Using a low-end selection for the pivot in a quick sort makes it difficult to achieve efficiency close to n log n. Our list is unsorted, and provides a lower number of compares versus other ordering. We still see variations in the number of compares and assigns, likely due to our placement of the statements in our loops.

G. Analysis of Merge Sort
   Without the completion of a functioning merge sort, the numbers to report were unattainable. However, we would expect similar numbers as seen in the quick sort, as merge sort is also on the order of O (n log n).