COSC 3020 Name: Jacob Tuttle
Algorithms

Lab 04

October 3, 2019

1. Permutation Sort: Solutions to the permutation sort problem are provided within the lab file `lab_04_tuttle.html`.

2. Runtime Analysis: The runtime of this algorithm is more difficult than others to evaluate since the runtime of sorting a single array can vary depending on what order the array is in when it is passed to the sorting function. So let us first examine the worst case scenario. Imagine that throughout the search tree of the permutation space, the sorted list is the last one (this would occur when the array is nearly sorted and the only change is that the first element has moved to the end). This is certainly the worst case scenario because there must be $n!$ other arrays searched first. This isn't even $n!$ operations, but $n!$ other leafs must be checked. Calculating every single leaf node within the tree would take at least 1 swap per leaf, driving up the worst case time complexity to $O(n * n!)$.

Now, considering the best case scenario the number of permutations to be calculated will be only 1 (meaning that the array was already sorted) in a time of $O(n)$.

In general, assuming a randomized distribution of the values to be sorted, the number of leaves needed to explore will range between 1 and $n!$ leaves. From this, we can conclude that the average runtime will be $\Theta(n * \frac{n!}{2})$ which, in asymptotic analysis, is $\Theta(n * n!)$

Consider now that instead of systematically searching the permutation space to find the correctly sorted solution, the Permutation Sort algorithm simply randomized the elements within the array and checked to see if they were sorted. In the best case, this would be able to able to generate the sorted list in a time of $O(n)$. In the worst case, it would never find the correctly sorted list.