2optSwap Writeup

Algorithms and Data Structures

For my implementation of the 2optSwap I had runtimes that were surprisingly fast. I believe that my code works from what I understand about the algorithm, but I never had runtimes of over a minute. Even arrays of size 700 take around 5,000ms to complete, and an array of that size going through and checking if the array is correct would take forever. When run on smaller arrays it seems to output the correct array to get the smallest route. I struggled with selecting the i and k values at first, having the user input them did not seem to work with how I was storing the routes. In searching online I found come c++ implementations that seemed to have a helper function outside that did the cutting, reversing, and concatenating back together. So I used this to have a second function that simply did the modifications and then called that function inside a nested for loop using the i and k from the for loops as the i and k in the 2optSwap. After getting the i and k values I push the modified array to a cache and store all possible permutations of routes based off i and k. I then pop arrays off cache and check their values and store the smallest array, once I have popped all arrays out of cache and checked their size I return the array with the shortest route. From my runtime plot, it looks like the empirical runtime of my program is running in exponential time. Where the worst-case asymptotic time complexity would be O(n3). I get this because on the inside of the function I have a while loop running from 0 to cache.length where cache is dependent on the total route size, inside of this while loop I have two for loops iterating over arrays within cache. These two for loops are also dependent on cache.length and cache.length is dependent on routes so I have an outermost loop, middle loop, inner loop all running at most n times making the worst case time complexity for my program O(n3). The space complexity however is O(n2), this is because an array takes up size O(n) space and the only time that I have more space allocated than for one total array is when I call the 2optSwap function and push it to cache, I am creating the space for the arrays of 2opt and then the space of cache to store that space so n \* n means the space taken up is O(n2).