Project 5 rough_draft_algorithm explanation
Group 4: Emma Paul, Ian Paul, Abdulhalim Bambang

We first approached this problem with time complexity in mind. Knowing that a brute force algorithm is O(n!), and the fact that the competition part makes a parameter of our algorithm performing under 5 minutes, we wanted to create something that could perform fairly quickly with a very large number of cities. First we scaled the problem down by putting 10 random points on graphing paper to try and think of a way we could quickly find a route. We were trying to find an algorithm that would allow us to find the closest next city without having to visit all the cities and compare the distances. One solution we came up with was to create two arrays, one where the city id's were ordered by the x-coordinate, and another that was ordered by the y-coordinate. We observed that often times the closest city was a point that was either the city above or below the city we were at on either the ordered x-coordinate array or ordered y-coordinate array. So, we created a function that would take in the city id we were at, and would look at the city index that was directly above and below the city in the ordered arrays. Therefore it would have 4 cities in question that it would compute distances for, and select the city id that was the shortest of all of them. One thing we observed was that sometimes at the end of the tour, the last couple of cities would have long distances between them in order for all of them to be visited. To fix this problem, we divided the x-y plane into 4 quadrants. We then ran our function on one quadrant at a time, so that all the cities in that quadrant would be visited before moving on to the next. In a sense, that caused our algorithm to go in a somewhat circle. This reduced our tour distance, however when we ran it on the test cases, our result was only about 65%. It would run quickly because the time complexity of our algorithm was around O(n^2) at worse due to the sorting. Originally, when we created our algorithm in the first place, our city indexes that we had on graphing paper were more spread out, so usually one city would be on an x or y coordinate. For example, only one city would have x = 4, or y = 4. Because most of the points had a unique x or y coordinate, our algorithm seemed to work fairly well on paper. However, when we ran our algorithm on the test cases and mapped out our tour on graphing paper and noticed something we had not come across before. Many of the cities on the test cases were on the same x or y plane, so when our algorithm sorted and looked at the city index above and below, it was often a city index that had the same x coordinate (if looking at the ordered x-coordinate array) or y coordinate (if looking at the ordered y-coordinate array). Therefore, there were many cases where the next city our algorithm found was a city directly above/below or to the left/right, that was a long distance away compared to the shorter distanced indexes by the city in question. This was not something we had considered when we first began, so we had to start over. Because we worked so long and hard on the algorithm described, we included it in our project files as "rough_draft_algorithm.c".