Garrett Mestel
C202
Programming Assignment 6
Due 7-29-2017

The problem itself is the Traveling Saleman Problem of trying to get the best route between cities at the cheapest cost without revisiting any of the cities. However, instead of finding the best, we want to find a good answer that we can get in a reasonable amount of time.

The algorithm itself is not that long. To cut down on time we assume that the starting city is always the first on the list. Then we go through the costs of going from the current city to all the others and find the lowest (making sure not to count the city that we are in). Then we go to that city. As we go to each city we mark it as gone to so the next time we look through the cities we also make sure that we have not been to it. The cost of the path is found just from going through the path and adding up the cost of each trip.

The program is designed to use a 2d matrix that the x index is the starting city and the y index is the ending city and the value is the cost from those two cities. It then starts by pushing the starting city (0) onto a stack and setting 0 as the current city. There is also an arraylist that 0 is added to. It then starts a loop that stops when the stack is empty. There are two resetting variables in each loop. One is the min which is set to the highest value it can be. It represents the smallest cost found so far. The other is a boolean that is based on whether min has been changed in that loop or not. It will then start a new loop that goes through the cost of each city by using the current city as the x in the matrix and test each other number between 1 and the number of cities minus 1. It must meet three criteria. First the cost must not be equal to 0. If it is, it means it is the city we are currently in. Second the new city must not be in the arraylist letting us know that we have not been to it yet. Lastly the cost must be less than the min or it is not the best found path. If something does meet everything then it sets the min to the cost, it remembers that city as the closest one and sets the boolean to true. After the loop that goes through the cities but not the one that ends by the stack being empty is done, if the boolean is true then it will push the remembered city to the stack and add it to the arraylist. It also prints out the city that we are now at. If it is not true then we pop one city off the stack. When the stack is empty, it takes the list of visited cities that is ordered in the way we went to them and finds the cost and prints that out too.

Now we will compare this algorithm's output to the output that finds the best path. The output of the first three test files are (output form is changed so it can better be read but not the numbers):
fast:
number of cities 12 - the cost is 920 - the time for 12 cities is 1015964
number of cities 13 - the cost is 929 - the time for 13 cities is 705744
number of cities 14 - the cost is 1025 - the time for 14 cities is 796111

Best:
The Best for 12 number of cities is 821

the time for 12 cities is 421186498
The Best for 13 number of cities is 864
the time for 13 cities is 1272955826
The Best for 14 number of cities is 958
the time for 14 cities is 7434357272

First off the best algorithm does give us better answers by about 100 in the first round and 60 in the second two. This is what we expected as we know the fast way does not look at all of the possible ways but overall it still does a good job at getting close. But next let's look at the time. The best way takes about 1000 times longer and is getting much bigger each time a city is added. On the other hand the new way barely takes much longer each time a city is added. While the best may give us the cheaper option, the fast way will more than make up for it by actually being able to give us an answer when we use more cities.