**Group 26**

**Description**

The first function, slowTSP is used for inputs of less than 500 cities. It takes the list of cities, and makes two lists: visited, and mustVisit. The list mustVisit contains all cities, and the first one is moved to visited and popped from mustVisit. Then a while loop is set up to run as long as mustVisit contains any cities. Just inside the while loop, the shortestPath variable is set to infinity. As long as mustVisit is not empty, there are two for loops which will run inside the while loop. The first one loops over the visited array, and the second one over the mustVisit array. Inside the two loops, the distance is set to the distance from the visited[i] city and the mustVisit[j] city. The shortest distance found inside these two loops is taken by the shortestPath variable. So it is looping over the list^2 each time it finds a city to add to the tour.

The second algorithm is using the greedy heuristic approach. Meaning it is only an approximation of, and not the absolute optimal result. The function (greedyFasterTSP) receives a list of cities from the input in main, and marks them “mustVisit,” and a separate list for “visited” is created. A variable tourLength is created and set to zero. A while loop runs as long as mustVisit contains any cities. Inside the while loop, there is a for loop that runs over the visited array. Then, another for loop inside that one which iterates over the mustVisit array. The “distance” variable is set to the distance between the city at visited[i] and mustVisit[j]. When the nearest city in the mustVisit array is found, it is popped from that array and added to the visited array. The tourLength is incremented in this case. The variable “current” is set to this most recent city.

So in simpler words: you take one city, and look at the distances to every other city. Take the nearest city found, and add it to the list of “visited” cities, which is the tour. Then the city you just added, do the same thing: compare distances to each city in the remaining list of cities. Since you are iterating over the list \* the list, this is N^2.

What is the difference between the two?

The first algorithm Is N^3. It does a while loop, and for each iteration, it does TWO MORE loops just to find one city to add. Because each time it adds a city, it is comparing all cities in the tour (visited) to each city in the remaining list (mustVisit).

The faster algorithm doesn’t go this deep. It just has one nested loop, and each time it compares each city in the mustVisit array to ONLY ONE city. So there is a big difference in the two.

**Pseudo Code**

greedyFasterTSP(cities)

mustVisit ←cities

visited ← [0]

current ← mustVisit[0]

visited[0] ← current

mustVisit[0] ←remove

while mustVisit:

shortestPath ← ∞

for 0 < len(mustVisit)

Do{

distance = distance between (current, mustVisit[i])

if distance < shortestPath

shortestPath = distance

nearestCity = mustVisit[i]

}

tourLength += shortestPath

visited ←append nearestCity

mustVisit ← remove nearestCity

current = nearestCity

tourLength += distance between visited[0] and last city in visited array

slowTSP(cities)

while mustVisit

for 0 < length(visited)

for 0 < length(mustVisit)

Do{

distance = distance between (visited[i], mustVisit[j])

if distance < shortestPath

shortestPath = distance

nearestCity = mustVisit[i]

}

tourLength += shortestPath

visited ←append nearestCity

mustVisit ← remove nearestCity

tourLength += distance between visited[0] and last city in visited array

**Best Tours for Three Example Instances**

**Best Tours for the Competition Test Instances**