## Problem 1 (0.7 points)

The famous traveling salesmen problem is defined as follows:
Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

1) The most direct way to solve this problem is using BRUTAL FORCE SEARCH, which is to find all the possible routes through every city and find the shortest one. However, it is an almost impossible way to solve the problem. Explain why this is not a good idea.
2) If not using the direct way, it is also very hard to find a way to implement the algorithm. So one of the laziest guys came up with an algorithm called NEAREST NEIGHBOR ALGORITHM. The algorithm is just like its name and defined as follows:
i) start on an arbitrary vertex as current vertex.
ii) find out the shortest edge connecting current vertex and an unvisited vertex V .
iii) set current vertex to $V$.
iv) mark V as visited.
v) if all the vertices in domain are visited, then terminate.
vi) Go to step 2 .

With this algorithm, the runtime can be compressed down to $\mathrm{O}(\mathrm{n})$, where n is the number of cities needed to be visited. This algorithm was proven to be non-efficient but a possible quick way to find a path. Explain why this algorithm cannot always find the shortest path.

## Problem 2 ( 0.7 points)

Please read the following algorithm and answer the two questions:

1. What is the goal of the algorithm?
2. If we want to recover the array after the function is executed, what should we do?

There is a function that accepts an array(unsign int * array) and two int ( $\mathrm{n}, \mathrm{k}$ ) as parameters. The k is an integer that is greater than the biggest element in the array, and the n is the length of the array.

1. Iterate though input array $\operatorname{arr}[]$, for every element $\operatorname{arr}[i]$, increment $\operatorname{arr}[\operatorname{arr}[i] \% k]$ by $k$
2. Find the maximum value in the modified array and return the corresponding index.

## Problem 3 ( 0.6 points)

A node in a binary tree is an only-child if it has a parent node but no sibling node. The "loneliness-ratio" of a given binary tree T is defined as the following ratio: $\mathrm{LR}(\mathrm{t})=$ (the number of nodes in T that are only children)/(the number of nodes in T ).
Prove that for any nonempty AVL tree $T$ we have that $\operatorname{LR}(\mathrm{t})<=1 / 2$.

