Algorithms
TSP Project [100 pts]

1 Introduction

The objective of this project is to correctly implement two approaches to solving TSP and to measure their run times on a few different inputs. Assume the input file is organized as follows:

```
4
3 4
6 8
50 8
11 7
```

The first line contains n, the number of points. Each of the next n lines contains the (x, y) coordinates of the n points. Your output should report both the TSP tour (the order in which the points are visited) and the length of the TSP tour.

1. The first approach is the nearest-neighbor heuristic discussed in class. Pseudocode may be found on Page 6 of the text book. This approach is fast, but is not guaranteed to find an optimal tour.
2. The second approach is the exhaustive algorithm that we discussed in class that tries all permutations. Pseudocode may be found on Page 8 of the text. This approach is guaranteed to find an optimal solution, but is very slow. *You may consult the internet or any other source to find code or an algorithm to generate all permutations.*

Recall that the definition of TSP allows the tour to start at any of the n points. However, for consistency with our solutions, your algorithm should start at the first point listed in the input. [In the example above, this would be $(3, 4)$.]

2 Deliverables

The deliverables will consist of (1) a brief report as described below and (2) a verification of your software.

2.1 Report

1. [15 pts] Explain the details of your two implementations. Specifially, in both cases, discuss how you *efficiently* implementied high-level "english" statements provided in the pseudo-code.
2. [15 pts] Determine the worst-case time complexity of your algorithms in terms of n. (This will depend on your implementation.)
3. [20 pts] Use a random number generator to devise inputs for your algorithms for **at least** four different values of n. The values of n may need to be different for the two approaches and should be chosen with the following in mind:

(a) n should be large enough so that you can reliably determine the runtime of your algorithm by using an appropriate timing function call such as clock() to time your program; i.e., the minimum run time should be at least 10 times more than the resolution of your clock function (the smallest unit of time that it measures).

(b) Also choose n so that you can experimentally verify the theoretical runtime you derived above.

For each n, determine the run time by taking the average of three runs on the same input. This reduces the likelihood of inaccuracies due to system load. Display your results in a table. Explain your choice of n.

4. [10 pts] Match theory and practice: Argue/demonstrate that your experimental runtimes are consistent with the theoretical complexities you derived.

2.2 Verification

1. [40 pts] Verification: we will ask you to run both of your algorithms on inputs supplied by us after you submit your reports. Stay tuned for announcements about this.