ASSIGNMENT 8

For this assignment, do your own work; do not collaborate with classmates.

1. [25 marks, 2 each for the questions below and 5 for formatting of references]
   Consider the following problem: You have a set of strings \( T = \{s_1, \ldots, s_n\} \), and you want the shortest string \( s \) such that every string \( s_i \) in \( T \) appears as a substring of \( s \), i.e., \( s \) is a superstring of all the \( s_i \)'s. Note that “substring” means that the letters must be consecutive.
   For example, \( at \) is a substring of \( fate \), but it is not a substring of \( mallet \).
   Answer each of the following questions by searching any sources you like. For each question, give your source. Reliable sources are better than unreliable ones. Journal papers are typically more reliable than Wikipedia, or course notes found on the web, or queries on StackExchange. Note: only include sources that you yourself looked at. For example, if wikipedia says there is a polynomial time algorithm and refers to a paper by Tarjan, then you cannot say that your source is Tarjan's paper unless you've looked at Tarjan's paper and found a statement in the paper that there is a polynomial time algorithm. Similarly, if you find something in a textbook, then your source is the textbook, unless you look at the paper that the textbook refers to. There will be 1 mark for each answer and 1 mark for the reference given for each answer.
   Suggestion: use Google Scholar—it lets you find later papers that refer to a given paper. Give your reference list as research papers in CS do. Latex and Bibtext (or the newer Biblatex) are great for this task, although you are not required to use them.
   (1) What is the problem usually called?
   (2) What is a main application of the problem?
   (3) The problem is NP-hard. Find a source for this result. What is the reduction from? (Don't give the NP-completeness proof.)
   (4) There is a greedy algorithm for this problem. Describe the greedy algorithm at a high level. Who is it due to?
   (5) What is the best approximation factor known for the greedy algorithm? Who proved it? (Don't give the proof!)
   (6) What is the best lower bound known for the approximation factor of the greedy algorithm? Give an example that achieves the lower bound.
   (7) It is not obvious how to implement the greedy algorithm efficiently. What is the best running time known? Who is it due to?
   (8) What is the best approximation factor known for a polynomial time algorithm for the problem? Who is it due to?
   (9) Is there a PTAS for the problem?
   (10) If you had to code up a solution to this problem, which algorithm would you use? This is a judgement call. You don't need a source, but you should briefly justify the reasons for your choice.