```
===========================================================================
CSC 373                Tutorial Exercises for Week  2           Winter 2019
===========================================================================
```

0. (a) Write down the complete greedy activity-scheduling algorithm (the final, correct one), in detailed pseudo-code.

   (b) Trace the greedy activity-scheduling algorithm on the following input -- already sorted by non-decreasing finish times.

       A_1 = (1,4); A_2 = (3,6); A_3 = (5,8); A_4 = (7,10);
       A_5 = (9,12); A_6 = (11,14)

   For each iteration of the algorithm's main loop, write down precisely the partial solution S_i generated by the algorithm up until that point. Also write down a complete list of *every* optimal solution that extends S_i.

   IMPORTANT NOTE:
   This exercise illustrates the type of "active note-taking" you will be expected to do on your own for the rest of the course. There won't always be explicit exercises on everything done in lecture. But you will be expected to have thought about all of the lecture material (concepts and examples). As you do this, you should come up with your own examples to make sure you understand the material, or so that you can at least ask well-grounded questions.

1. Consider the problem of making change: assume you have an unlimited supply of 1c, 5c, 10c and 25c coins, and an amount of money A that you want to make change for. The problem is to come up with a number of coins that adds up to exactly A cents while using as few coins as possible. For example, if you want to make change for 15c, you would clearly use one dime (10c) and one nickel (5c) instead of using 15 pennies (1c)!
   Write a greedy algorithm that attempts to solve this problem, then write a complete proof that your algorithm is correct, following the structure presented in class.

   NOTE: The point of this exercise is NOT that the algorithm requires a clever idea (it does not!) but to let you practice writing a proof of correctness for greedy algorithms using the structure from class.

   Hint: To make the proof of correctness easier to write, think of your algorithm as generating a solution one coin at a time (instead of computing directly the number of coins of each denomination). For example, if you're trying to make change for 85c, instead of thinking "the algorithm starts by using three quarters, then ...", think "the algorithm starts by using one quarter, then for what remains it will use another quarter, ..." (The idea is not that complicated but it involves a number of cases.)

2. Does your algorithm always work even if the coin values change? In other words, if you're given an arbitrary list of coin values (called "denominations") d[0] < d[1] < ... < d[k], with d[0] = 1, does your algorithm always make change using the smallest number of coins? (For example, the Canadian coins are represented by d = [1, 5, 10, 25].)
   Either provide a single counter-example and explain why it is a

counter-example, or explain how to modify your proof above so that it applies in the general case.

*3. If the algorithm fails in general, what can you say about the cases when it works? Try to state the most general property possible that guarantees that the algorithm will produce an optimal solution, and if you can, give a rigorous argument that you are correct.

(This is a "starred" problem: we don't expect you to be able to solve it as part of the course, but it's a nice extension of what you've been working on during this tutorial, in case you finish early and want to explore this problem further.
WARNING! Don't spend any time on this until you have *completely* finished the other problems! It would be a shame if you failed to learn some important aspect of the material -- and maybe even lose marks because of it -- simply because you got distracted by a supplementary problem.)