CS 535 Homework 8
Due: November 13 (W), in class.

Everyone, please answer problems 1, 2, 3.

1. **Revisiting the scheduling problem.** Recall our algorithm for scheduling tasks in as few machines as possible:

   **GreedySchedule**((s₁, f₁), (s₂, f₂), . . . , (sₙ, fₙ))

   Sort the tasks according to their start times so that s₁ ≤ s₂ ≤ . . . ≤ sₙ.
   Initially set the number of machines m to 0.
   Initialize array A[1 . . . n] that will store the machine assigned to each task.
   For i = 1 to n
     if task i can be processed in some machine j
       A[i] ← j
     else
       m ← m + 1 /* That is, increase the number of machines. */
       A[i] ← m
   Return A[1 . . . n]

   In class, I proved that the above scheduling algorithm will always output a schedule that uses the fewest number of machines. Will the above algorithm still output an optimal schedule if we make one of these changes?

   a. Instead of sorting the tasks according to their start times, we sort them according to their finish times.

   b. Instead of sorting the tasks according to their start times, we sort them according to their processing times (i.e. fᵢ − sᵢ).

   c. We do not sort at all. You can think of it this way: the tasks arrive one at a time. As soon as a task arrives, you schedule it in a machine that can process it. If none of the current m machines can process it, you open a new one and schedule it there.

   For each of the changes, if you answer “yes”, then prove that the algorithm still works. If you answer “no”, provide a counterexample and explain how the proof presented in class “breaks” – i.e., why it does not work anymore.

2. Your one-person take-out restaurant just received n orders. To keep your sanity, you always process the orders one at a time. Let tᵢ minutes be the time it takes to prepare the food for order i. So if, for example, you want to process order 1 first, then order 2, then order 3, etc., then customer 1 waits for t₁ minutes, customer 2 waits for t₁ + t₂ minutes, customer 3 for t₁ + t₂ + t₃ minutes, etc. Your goal is to
minimize the total waiting time of the customers:

\[ T = \sum_{i=1}^{n} \text{time spent waiting by customer } i. \]

a. Warm up! Suppose there are 5 orders and \( t_1 = 10, t_2 = 5, t_3 = 20, t_4 = 15, \) and \( t_5 = 1. \) In what sequence should the customers be served so that the total waiting time is minimized?

b. Now, given \( n \) orders, describe a greedy algorithm that solves the general problem. That is, the algorithm should output the sequence in which you should process the orders so that the total waiting time of the customers is as small as possible. What is its running time?

c. Prove/explain why your algorithm works in a manner similar to the ones we did in class.

3. The Minimum Coverage Problem. Let \( L \) be a set of \( n \) line segments on the \( x \)-axis. The \( i \)th line segment is \([l_i, r_i]\), where \( l_i \) and \( r_i \) denote the left and right endpoints of the line segment respectively. For some positive number \( M \), we want to cover the segment \([0, M]\) completely using the fewest number of line segments from \( M \).

a. Warm up! Let \( L = \{[-1.5, 1.5], [-4, 2], [2, 3], [3, 5], [1, 4.5]\} \) and \( M = 4.2. \) What is the fewest number of line segments from \( L \) that cover \([0, 4.2]\). What are they?

b. Given a set \( L \) and a positive number \( M \), solve the Minimum Coverage Problem using a greedy algorithm. In particular, if the line segments in \( L \) can cover \([0, M]\), the algorithm should output the fewest number of line segments that can do so; otherwise, it should indicate that the line segments from \( L \) cannot do so. What is its running time?

c. Prove/explain why your algorithm produces the correct answer.