Worth: 7.5% **Due:** Before 10pm on Tuesday 2 October.

Remember to write the *full name*, *student number*, and *CDF/UTOR email address* of *each group member* prominently on your submission.
*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes, and materials available directly on the course webpage). For example, indicate clearly the **name** of every student with whom you had discussions (other than group members), the **title** of every additional textbook you consulted, the **source** of every additional web document you used,* etc.

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.

---

[15] 1. A busy web server receives many document requests at the same time, and must decide in what order to reply to the requests. Each request takes a certain time for the server to process, and the server can only process one request at a time. But each request also takes a certain time for the web clients to render, independently of the other web clients. Suppose that we are interested in responding to the requests in an order that guarantees all requests will be fully rendered as quickly as possible.

Formally, consider the following "Web Request" problem:

Input: Requests $R_1, R_2, \ldots, R_n$, where each request R_i consists of "server" time s_i and "client" time c_i (both of them positive integers). All server time takes place on a single processor, sequentially. All client time takes place on n independent processors, in parallel.

Output: An ordering of the requests $i_1, i_2, \ldots, i_n$ (a permutation of $[1, 2, \ldots, n]$) that minimizes the overall completion time: $\max\left\{s_{i_1} + s_{i_2} + \cdots + s_{i_k} + c_{i_k} : 1 \leqslant k \leqslant n\right\}$ — where the completion time for request number k is equal to the total server time for all requests processed before k ($s_{i_1} + s_{i_2} + \cdots + s_{i_{k-1}}$), plus the time to fully process and render request k ($s_{i_k} + c_{i_k}$).

(a) Write a greedy algorithm to solve the Web Request problem. Give a detailed pseudo-code implementation of your algorithm, as well as a high-level English description of the main steps in your algorithm.

What is the worst-case running time of your algorithm? Justify briefly.

(b) Write a detailed proof that your algorithm always produces an optimal solution.

[15] 2. Consider the following "MST with Fixed Leaves" problem:

Input: A weighted graph $G = (V, E)$ with integer costs $c(e)$ for all edged $e \in E$, and a subset of vertices $L \subseteq V$.

Output: A spanning tree T of G where every node of L is a leaf in T and T has the minimum total cost among all such spanning trees.

(a) Does this problem always have a solution? In other words, are there inputs G, L for which there is no spanning tree T that satisfies the requirements?

Either provide a counter-example (along with an explanation of why it is a counter-example), or give a detailed argument that there is always some solution.

(b) Let G, L be an input for the MST with Fixed Leaves problem for which there *is* a solution.

Is every MST of G an optimal solution to the MST with Fixed Leaves problem? Justify.

Is every optimal solution to the MST with Fixed Leaves problem necessarily a MST of G (if we remove the constraint that every node of L must be a leaf)? Justify.

(c) Write a greedy algorithm to solve the MST with Fixed Leaves problem. Give a detailed pseudo-code implementation of your algorithm, as well as a high-level English description of the main steps in your algorithm.

What is the worst-case running time of your algorithm? Justify briefly.

(d) Write a detailed proof that your algorithm always produces an optimal solution.

[15] 3. Consider the problem of making change, given a finite number of coins with various values. Formally:

Input: A list of positive integer coin values $c_1, c_2, \ldots, c_m$ (with repeated values allowed) and a positive integer amount A.

Output: A selection of coins $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$ such that $c_{i_1} + c_{i_2} + \cdots + c_{i_k} = A$ and k is as small as possible. If it is impossible to make change for amount A exactly, then the output should be the empty set $\varnothing$.

(a) Describe a natural greedy strategy you could use to try and solve this problem, and show that your strategy does not work.

(The point of this question is **not** to try and come up with a really clever greedy strategy — rather, we simply want you to show why the "obvious" strategy fails to work.)

(b) Give a detailed dynamic programming algorithm to solve this problem. Follow the steps outlined in class, and include a brief (but convincing) argument that your algorithm is correct.

What is the worst-case running time of your algorithm? Justify briefly.