CPSC 320 2025S: Assignment 2

This assignment is due **Friday, July 18 at 7 PM**. Late submissions will not be accepted. Please follow these guidelines:

- Prepare your solution using LaTeX and submit a pdf file. Easiest will be to submit using the .tex file provided. For questions where you need to select a circle, you can simply change `\fillinMCmath` to `\fillinMCmathsoln` .
- Enclose each paragraph of your solution with `\begin{soln}Your solution here...\end{soln}`. Your solution will then appear in dark blue, making it a lot easier for TAs to find what you wrote.
- Submit the assignment via GradeScope. Your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**.
- After uploading to Gradescope, link each question with the page of your pdf containing your solution.

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode should communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. You may **neither** include what we consider to be irrelevant coding details **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid language-specific notation like C/C++/Java's ternary (question-mark-colon) operator.)

Remember also to **justify/explain your answers**. We understand that gauging how much justification to provide can be tricky. Inevitably, judgment is applied by both student and grader as to how much is enough, or too much, and it's frustrating for all concerned when judgments don't align. Justifications/explanations need not be long or formal, but should be clear and specific (referring back to lines of pseudocode, for example). Proofs should be a bit more formal.

On the plus side, if you choose an incorrect answer when selecting an option but your reasoning shows partial understanding, you might get more marks than if no justification is provided. And the effort you expend in writing down the justification will hopefully help you gain deeper understanding and may well help you converge to the right selection :).

Ok, time to get started...

Group Members

Please list the CWLs of all group members here (even if you are submitting by yourself). We will deduct a mark if this is incorrect or missing.

1 Take Your Order

[10 points] Take the following functions and arrange them in ascending order of growth rate. That is, if $g(n)$ appears after $f(n)$ in your ordering, it should be the case that $f(n) \in O(g(n))$. (Example: the functions n and n^2 would appear in the order: n, n^2.) Justify your answer.

$$f_1(n) = n$$
$$f_2(n) = 2^n$$
$$f_3(n) = \sqrt{n} \log n$$
$$f_4(n) = \frac{n \log n}{\log(\log n)}$$
$$f_5(n) = (n-2)!$$
$$f_6(n) = n^{\lg n}$$

2 All-Stars

Suppose you're organizing a tennis tournament between n players which we simply label as $1, 2, \ldots, n$. Each player competes against every other player, and there are no repeat matches and no tied scores. We model the results of the competition as a directed graph with n vertices and exactly one directed edge between each pair of vertices: $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$ where either $(i, j)) \in E$ (if i defeated j), or $(j, i)) \in E$ (if j defeated i). Such a graph is called a *tournament*.

You want to determine a way to rank these players, but a challenge here is that cycles **may** exist in your tournament – e.g., i may defeat j in a match, and j could defeat k, but then we could still have k win a match against i; this would lead to a cycle of length three.

A ranking for the tournament is an ordering $(r_1, r_2, \ldots, r_n)$ where node r_1 is established as the overall rank 1 player. Due to the potential existence of cycles it is challenging to find a fair ranking.

1. [4 points] We decide that the top ranked player should always be some node whose out-degree in the directed graph is maximum. We motivate this claim by proving that if player i has maximum degree, then for every other player j either i beat j, or i beat some player k which beat j. Prove this claim.

   Hint: A proof by induction on n works well.

2. [2 points] We decide that a reasonable definition is suggested by the previous result. A sequence of nodes $(r_1, r_2, \ldots, r_n)$ is a *valid ranking* if for each $i = 1, 2, \ldots, n$ the node r_i has maximum out-degree in the sub-tournament induced by the nodes $r_i, r_{i+1}, \ldots, r_n$. Give a polynomial-time algorithm which decides if a given permutation of the nodes $(v_1, v_2, \ldots, v_n)$ is a valid ranking.

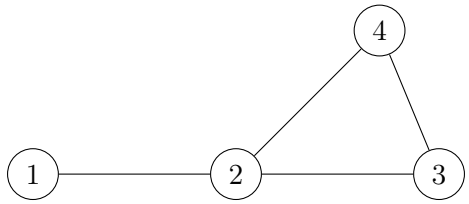
3. [2 points] Describe a tournament of size n which has precisely one valid ranking.

4. [4 points] One brute force method for producing all valid rankings is to consider all permutations of the nodes and then use your algorithm for Part b to check each one if it is a valid ranking. This will run in time $O(n!p(n))$ where $p(n)$ is the runtime of your algorithm in Part b.

   Instead, create a better algorithm which runs in time $O(|Val|poly(n))$ where Val is the set of valid rankings, and $poly(n)$ is a polynomial in n. (You do not need to analyze the runtime of your algorithm.)

3 Colour Me Confused

Given a graph $G = (V, E)$, we consider the problem of assigning a colour to each vertex in G such that no two adjacent vertices share a colour. Recall that the *degree* of a vertex $v \in V$ is the number of edges incident on v. Let d be the *maximum degree* in the graph.



The maximum degree d in the graph above is 3. The graph has a 3-colouring (by using different colours for nodes 1, 2, and 3 and colouring node 4 the same as 1 or 2) and a 4-colouring (by using a different colour for every node), but does not have a 2-colouring.

1. [2 points] For any value of d, describe a graph with maximum degree d that can be coloured with two colours.

2. [2 points] Consider the following greedy algorithm to find a colouring for a graph:

   > Order vertices arbitrarily. Colour the first vertex with colour 1. Then choose the next vertex v and colour it with the lowest-numbered colour that has not been used on any previously-coloured vertices adjacent to v. If all previously-used colours appear on a vertex adjacent to v, introduce a new colour and number it, and assign the new colour to vertex v.

   Prove that this algorithm uses at most $d + 1$ colours.

3. [2 points] Give and briefly explain an example where this algorithm does not produce an optimal colouring (i.e., where it uses more colours than is necessary to colour the graph).

4. [5 points] Now, assume that there is at least one vertex v in V with degree **less than** d. Design a greedy algorithm that will colour vertices of G with at most d colours. You should proceed by first **ordering the vertices** in some way, and then assigning colours using the colouring strategy in question 4.2. You should explain why your algorithm uses no more than d colours.

4 Weekly Meeting Logistics

You're the manager of an animal shelter, which is run by a few full-time staff members and a group of n volunteers. Each of the volunteers is scheduled to work one shift during the week. There are different jobs associated with these shifts (such as caring for the animals, interacting with visitors to the shelter, handling administrative tasks, etc.), but each shift is a single contiguous interval of time. Shifts cannot span more than one week (e.g., we cannot have a shift from 10 PM Saturday to 6 AM Sunday). There can be multiple shifts going on at once.

You'd like to arrange a weekly meeting with your staff and volunteers, but you have too many volunteers to be able to find a meeting time that works for everyone. Instead, you'd like to identify a suitable subset of volunteers to instead attend the weekly meeting. You'd like for every volunteer to have a shift that overlaps (at least partially) with a volunteer who is attending the meeting. Your thinking is that you can't personally meet with every single volunteer, but you would like to at least meet with people who have been working with every volunteer (and may be able to let you know if a volunteer is disgruntled or having any difficulties with their performance, etc.). Because your volunteers are busy people, you want to accomplish this with the fewest possible volunteers.

Your volunteer shifts are given as an input list V, and each volunteer shift v_i is defined by a start and finish time (s_i, f_i). Your goal is to find the smallest subset of volunteer shifts such that every shift in V overlaps with at least one of the chosen shifts. A shift $v_i = (1, 4)$ overlaps with the shift $v_j = (3, 5)$ but not with the shift $v_k = (4, 6)$.

1. [2 points] Your co-worker proposes the following greedy algorithm to select volunteers for your meeting:

   > Select the shift v that overlaps with the most other shifts, discard all shifts that overlap with v, and recurse on the remaining shifts.

   Give and briefly explain a counterexample to prove that this greedy strategy is not optimal.

2. [3 points] Give a greedy algorithm to solve this problem. Give an unambiguous specification of your algorithm using a **brief, plain English description**. Do not write pseudocode or worry about implementation details yet. (You may do this in part 5 if you feel that it's necessary to achieve a particular runtime.)

3. [2 points] In this and the next question, you will prove the correctness of your greedy algorithm. To start, give a "greedy stays ahead" lemma that you can use to prove that your greedy algorithm is optimal. You do not need to prove the lemma (that comes next). Your lemma should compare a partial greedy solution to a partial optimal solution and describe some way in which the partial greedy solution is "ahead."

4. [4 points] Prove the correctness of your algorithm. That is, prove that your set of selected shifts will (a) overlap with all the shifts in V, and (b) contains as few shifts as possible. You should do this by proving the greedy stays ahead lemma you stated in the previous question and using it to prove that the greedy solution is optimal.

5. [4 points] Briefly justify a good asymptotic bound on the runtime of your algorithm. If you prefer to present pseudo-code to help track the runtime incurred, you may do so.