CPSC 320 2023W1: Assignment 1

This assignment is due on **Wednesday, Sep 27 at 10pm Vancouver time** on Gradescope. Assignments submitted by noon on the day after the deadline will be accepted, but a penalty of 15% will be applied. Please follow these guidelines:

- We strongly recommend that you prepare your solution using LaTeX, and submit a pdf file. Easiest will be to use the .tex file provided, along with the provided tutorial-students.sty file. We will accept solutions that are prepared using other good formatting systems, as long as they are clearly legibile. Solutions that are handwritten or difficult to read will receive a grade of zero.
- For questions where you need to select a circle, you can simply change `\fillinMCmath` to `\fillinMCmathsoln` .
- In problem 3 below, you'll see an example of how you can format pseudocode using latex. If you have your own preferred way to format pseudocode, you can use that, take an image of it, and insert it as a figure into your document using the following lines (changing the file name as needed and adjusting the width so that it displays clearly):

```
\begin{center}
\includegraphics[width=0.3\textwidth]{filename.jpg}
\end{center}
```

- We would also appreciate it if you could typeset your solutions in blue, as it makes it easier for TAs to find the solutions. To do this, all you need to do is use the following lines in your latex file:

```
\begin{soln}
[Your solution here]
\end{soln}
```

  which will produce:

  [Your solution here]

- Whenever possible, keep the solution to a subproblem on a single page, rather than having it span two pages.
- Include in question 1 below the **names** of each member of your team, as recorded on Canvas. If you want an extra double-check on your identity, include your student number. Reminder: groups should include a maximum of three students; we encourage groups of two.
- Start each problem on a new page, using the same numbering and ordering as this handout.
- 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 all the pages of your pdf containing your solution.

Before we begin, a few notes on pseudocode. Your pseudocode should communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. Don't include what we consider to be irrelevant detail.

Remember also to **justify/explain your answers**, unless explicitly indicated otherwise. 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). In this class the words **"prove", "explain", "show"** and **"justify"** are used interchangeably; we're not looking for the formality that might be expected in a Math class. You don't need to be more formal when a proof is requested, but sometimes formal notation or assertions can really help make concepts clear and unambiguous.

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...

1 List of names of group members (as listed on Canvas)

Provide the list here. This is worth 1 mark. Include student numbers as a secondary failsafe if you wish.

2 Statement on collaboration and use of resources

To develop good practices in doing homeworks, citing resources and acknowledging input from others, please complete the following. This question is worth 2 marks.

1. All group members have read and followed the guidelines for groupwork on assignments given on the website (see `https://www.students.cs.ubc.ca/~cs-320/2023W1/coursework.html`, under Assignments). ◯ Yes ◯ No
2. We used the following resources (list books, online sources, etc. that you consulted):
3. One or more of us consulted with course staff during office hours.
   ◯ Yes ◯ No
4. One or more of us collaborated with other CPSC 320 students; none of us took written notes during our consultations and we took at least a half-hour break afterwards.
   ◯ Yes ◯ No

   If yes, please list their name(s) here:
5. One or more of us collaborated with or consulted others outside of CPSC 320; none of us took written notes during our consultations and we took at least a half-hour break afterwards.
   ◯ Yes ◯ No

   If yes, please list their name(s) here:

3 Matching Preferences

Here is an instance of the SMP (stable matching problem) with two applicants and two employers that has at least two stable solutions:

e_1:	a_1, a_2	a_1 :	e_2, e_1
e_2:	a_2, a_1	a_2 :	e_1, e_2

The matching $\{(e_1, a_1), (e_2, a_2)\}$ is stable since both employers are matched with their top choice. Also the matching $\{(e_1, a_2), (e_2, a_1)\}$ is stable since both applicants are matched with their top choice. The matching $\{(e_1, a_1), (e_2, a_2)\}$ is best for both employers, while matching $\{(e_1, a_2), (e_2, a_1)\}$ is best for both applicants.

More generally, let I be an SMP instance with (at least) two different stable matchings M and M'. We say that employer e prefers matching M over M' if e prefers its match in M over its match in M'. That is (using notation from the worksheet) if $(e, a) \in M$ and $(e, a') \in M'$, then $a >_e a'$.

1. (3 points) Show an instance of SMP with four applicants and four employers, and show two stable matchings M and M' for this instance, such that two employers prefer M to M' while the remaining two employers prefer M' to M. Hint: Build on the example given above. (Make sure to provide a short justification of your answer.)

2. (1 point) Let $G(I, M, M')$ be the bipartite graph whose nodes are employers and applicants of I, and with an edge between e and a if either $(e, a) \in M$ or $(e, a) \in M'$. Is the graph $G(I, M, M')$ for your SMP instance of part 1 connected? Give a short justification. (Note: While this question refers to graphs, you don't need any knowledge of graph algorithms to complete the question.)

3. (2 points) In this and the subsequent parts of the problem let I be an instance with $n \geq$ employers and n applicants, with distinct stable matchings M and M'.

   Without loss of generality, let e_1 be an employer that prefers M over M', and let e_1 be matched with a_1 in M, that is $(e_1, a_1) \in M$. Let e_2 be the employer matched with a_1 in M', and let a_2 be the applicant matched with e_2 in M. That is, we have:

M	M'
(e_1, a_1)	$(e_1, ??)$
(e_2, a_2)	(e_2, a_1)
$\ldots$	$\ldots$

   Show that $e_2 >_{a_1} e_1$ and that $a_2 >_{e_2} a_1$.

4. (2 points) Now suppose that there are i distinct employers $e_1, e_2, \ldots e_i$, and i distinct applicants $a_1, a_2, \ldots a_i$, such that in the matchings we have:

M	M'
(e_1, a_1)	$(e_1, ??)$
(e_2, a_2)	(e_2, a_1)
$\ldots$	$\ldots$
(e_{i-1}, a_{i-1})	(e_{i-1}, a_{i-2})
(e_i, a_i)	(e_i, a_{i-1})

Suppose furthermore that $e_1, e_2, \ldots e_{i-1}$ prefer their matches in M to their matches in M'. Show that a_{i-1} must prefer e_i to e_{i-1} and furthermore, that e_i must prefer a_i to a_{i-1}.

5. (3 points) Now, suppose that the graph $G(I, M, M')$ is connected. Show that if one employer prefers M over M', then it must be the case that *all* employers prefer M over M'.

4 Recursive Permutation Generation

In this problem you'll show that the recursive permutation generation algorithm from the first tutorial does generate all permutations of the numbers in an array. Here is the algorithm again.

```
1: function GENERATE-ALL-PERMUTATIONS(p[1..n], i)
2:     ▷ p[1..n] contains n distinct integers, 1 ≤ i ≤ n
3:     if i = 1 then
4:         return p[1..n]
5:     else
6:         for j from i downto 1 do
7:             swap(p[i],p[j])
8:             GENERATE-ALL-PERMUTATIONS(p[1..n], i − 1)
9:             swap(p[i],p[j])
```

For a set S of i distinct numbers and a sequence s of additional distinct numbers, let permut(S, s) be the set of all $i!$ permutations obtained by permuting the numbers in S, and then appending the sequence of numbers s.

For example, if $n = 4$, $S = \{1, 3\}$ and $s = 4, 2$ then

$$\text{permut}(S, s) = \{(1, 3, 4, 2), (3, 1, 4, 2)\}.$$

When the sequence s is empty we denote it by ϵ and we'll shorten the notation by writing

$$\text{permut}(\{a_1, a_2, \ldots, a_i\})$$

to mean permut$(\{a_1, a_2, \ldots, a_i\}, \epsilon)$.

1. (2 points) Give a short explanation as to why, for any i in the range $2 \leq i \leq n$,

   $$\text{permut}(\{a_1, a_2, \ldots, a_i\}, a_{i+1}, a_{i+2}, \ldots, a_n)$$

   is the same as the set

   $$\cup_{1 \leq j \leq i} \text{permut}(\{a_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_i\}, a_j, a_{i+1}, \ldots, a_n).$$

2. (5 points) Use induction on i to prove the following claim:

   Claim: Let $a_1, a_2, \ldots, a_n$ be the sequence of distinct numbers in array $p[1..n]$. Then for any $i, 1 \leq i \leq n$, GENERATE-ALL-PERMUTATIONS$(p[1..n], i)$ returns exactly the $i!$ permutations in the set

   $$\text{permut}(\{a_1, a_2, \ldots, a_i\}, a_{i+1}, a_{i+2}, \ldots, a_n).$$

5 A Number Sieve

The following NUMBER-SIEVE algorithm repeatedly scans an array containing the numbers 1,2, $\ldots, n$ in order. On the first iteration of the **while** loop, the algorithm removes every second number, by replacing the number with -1. On subsequent iterations every third number from the remaining set of numbers (not counting the -1's) is removed; then every fourth number and so on. The algorithm terminates once no number is removed on some iteration. We'll call the numbers that are never removed the *lucky numbers*. The first few lucky numbers in a sufficiently long array are 1, 3, 7, 13, 19.

```
1:  function NUMBER-SIEVE(p[1..n])
2:      ▷ p[1..n] initially contains the numbers 1, 2, ..., n in this order
3:      done ← false
4:      k = 1
5:      while not done do
6:          k = k + 1
7:          j ← 0
8:          done ← true
9:          for i from 1 to n do          ▷ Remove every kth of the remaining positive numbers in p
10:             if p[i] ≠ −1 then
11:                 j ← j + 1
12:                 if j == k then
13:                     p[i] ← −1
14:                     j ← 0
15:                     done ← false
16:     return the list of positive numbers remaining in p
```

1. (4 points) One nice feature of big-O analysis of algorithm runtime is that we can often ignore annoying floors and ceilings. Show that if we ignore these, then for $k \geq 1$ there are n/k positive numbers remaining in the array at the start of the kth iteration of the **while** loop (assuming that the algorithm has not already terminated).

2. (3 points) What is the runtime of NUMBER-SIEVE, as a function of n?

6 Computing Averages

Let $p[1..n]$ be a 1D array of real-valued numbers. We want to create a 2D $n \times n$ array P, where $P[i][j] = 0$ if $j < i$ and otherwise $P[i][j] = \mu(i, j)$, where $\mu(i, j)$ is the average of the numbers in the sub-array $p[i..j]$. That is, if $i \leq j$, then $\mu(i, j) = (p[i] + p[i + 1] + \ldots + p[j])/(j - i + 1)$.

1. (3 points) Here is one algorithm that creates P and computes all entries in the 2D array $P[1..n][1..n]$. Explain why the runtime is $\Theta(n^3)$.

```
1: function COMPUTE-AVERAGES(p[1..n])
2:     create the 2D (uninitialized) array P[1..n][1..n]
3:     for i from 1 to n do
4:         for j from 1 to n do
5:             if j < i then
6:                 P[i][j] ← 0
7:             else
8:                 P[i][j] ← (p[i] + p[i + 1] ... p[j])/(j − i + 1)
```

2. (2 points) Modify the above algorithm to obtain an algorithm that runs in time $\Theta(n^2)$ time. You do not need to provide a justification of your runtime or the correctness of your algorithm.

```
1: function COMPUTE-AVERAGES(p[1..n])
2:     for i from 1 to n do
3:         for j from 1 to n do
4:             if j < i then
5:                 P[i][j] ← 0
6:             else if i == j then
7:                 P[i][j] ← p[i]
8:             else        ▷ i < j
9:                 P[i][j] ← PUT NEW PSEUDOCODE HERE
```