Due 23:59 Sept 21 (Sunday). There are 100 points in this assignment.
Submit your answers **(must be typed)** in pdf file to CourSys
`https://coursys.sfu.ca/2025fa-cmpt-705-x1/`.
Submissions received after 23:59 will get penalty of reducing points: 20 and 50 points deductions for submissions received at $[00:00, 00:10]$ and $(00:10, 00:30]$ of Sept 22, respectively; no points will be given to submissions after $00:30$ of Sept 22.

1. (Chapter 1 Problem 5 of the text book) 15 points

   Consider the following matching with indifferent ranking problem: Given a set M of n men and a set W of n women. Each man and each woman ranks the opposite gender with ties allowed in the ranking. More formally; each $m \in M$ has a rank $f_m(w)$ for every $w \in W$, $f_m : W \to \{1, .., n\}$ is a function, and each $w \in W$ has a rank $g_w(m)$ for every $m \in M$, $g_w : M \to \{1, .., n\}$ is a function; and $f_m(w) = f_m(w')$ is allowed for $w \neq w'$ ($g_w(m) = g_w(m')$ is allowed for $m \neq m'$). There are two notions for stability:
   (a) A *strong instability* in a perfect matching S is a pair $(m, w) \notin S$ such that for $(m, w'), (m', w) \in S$, $f_m(w) > f_m(w')$ and $g_w(m) > g_w(m')$.
   Either give an example of M and W with f_m and g_w defined for which every perfect matching has a strong instability or give an algorithm that is guaranteed to find a perfect matching with no instability.

   (b) A *weak instability* in a perfect matching S is a pair $(m, w) \notin S$ such that for $(m, w'), (m'w) \in S$, $f_m(w) \geq f_m(w')$ and $g_w(m) \geq g_w(m')$.
   Either give an example M and W with f_m and g_w defined for which every perfect matching has a weak instability or give an algorithm that is guaranteed to find a perfect matching with no weak instability.

2. (Chapter 1 Problem 6 of the text book) 10 points

   A shipping company has n ships and provides service to n ports. Each ship s_i has a schedule that says for each day of the month (has $m > n$ days), which port s_i is visiting or s_i is at sea; each ship visits each port for exactly one day during the month and no two ships is in the same port on the same day. The company wants to perform maintenance on all ships in a specific month via the following truncated schedule: for each ship s_i, there will be some day when s_i arrives in its scheduled port, s_i remains at the port for the rest of the month for maintenance. Prove that given the schedule of each ship, a truncated schedule for each ship can always be found such that no two ships can be in the same port on the same day, give an algorithm to find the truncated schedules, and analyze your algorithm.

3. (Chapter 2 Problems 1 and 3 of the text book. $\log n = \log_2 n$) 10 points

   Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most one hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within one hour?

(1) n^2, (2) n^3, (3) $100n^2$, (4) $n \log n$, (5) 2^n, (6) 2^{2^n}.

Take the following list of functions and arrange them in ascending order of growth rate, that is, if function $g(n)$ immediately follows function $f(n)$ in your list, then $f(n)$ is $O(g(n))$.

$f_1 = n^{2.5}$, $f_2(n) = \sqrt{2n}$, $f_3(n) = n + 10$, $f_4(n) = 10^n$, $f_5(n) = 100^n$, $f_6(n) = n^2 \log n$.

4. (Chapter 2 Problem 6 of the text book) 20 points

   Consider the following problem: given an array A consisting of n integers $A[1], A[2], ..., A[n]$, output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$, that is, the sum $A[i] + A[i + 1] + ... + A[j]$. (The value of array entry $B[i, j]$ is unspecified for $i \geq j$, so it does not matter what is output for these values.) Here is a simple algorithm to solve this problem.

```
for $i = 1, 2, .., n$ do
    for $j = i + 1, i + 2, .., n$ do
        Add up array entries $A[i]$ through $A[j]$
        Store the result in $B[i, j]$
    endfor
endfor
```

   (a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

   (b) For this same function f , show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

   (c) Give a different algorithm to solve this problem, with an asymptotically better running time, and analyze your algorithm. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

   When $\lim_{n\to\infty} g(n)/f(n) = 0$, $g(n)$ is little-O of $f(n)$, denoted as $g(n) = o(f(n))$.

5. (Chapter 3 Problem 2 of the text book) 10 points

   Give an algorithm to detect whether a given undirected graph contains a cycle, and analyze your algorithm. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(m + n)$ for a graph G with n nodes and m edges represented by an adjacent list.

6. (Chapter 3 Problem 10 of the text book) 15 points

   In a social network, a shortest path between two nodes is a path connecting the two nodes with the minimum number of edges in the network. An interesting problem is to find the number of shortest paths between two nodes of a social network. Given an undirected graph $G(V, E)$, and two nodes v and w in G, design an algorithm that computes the number of shortest paths between v and w in G, and analyze your algorithm. (The algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.

7. (Chapter 3 Problem 11 of the text book) 20 points

   There are n computers $C_1, .., C_n$ connected by networks. You are given a sequence of m triples (trace data), each triple (C_i, C_j, t_k) indicates that C_i and C_j exchange data at time t_k. Assume that the triples are presented in sorted order of time (i.e., $t_i < t_j$ for $i < j$) and each pair of computers communicate at most once during the entire time interval of the trace data. Assume that computer C_i was infected by a virus and communicates with C_j at the time t_k (i.e., if one of (C_i, C_j, t_k) and (C_j, C_i, t_k) appears in the trace data), then C_j becomes infected as well. Infection can thus spread from one computer to another across a sequence of communications, for example, if C_i is infected by time t_k, and the trace data contains (C_i, C_j, t_k) and (C_j, C_q, t_r), where $t_k < t_r$, then C_q becomes infected via C_j. Design an algorithm which given a sequence of trace data, decides whether a virus introduced at computer C_a at time t_x could have infected computer C_b by time t_y in $O(m + n)$ time, and analyze your algorithm.