Exam Logistics:

- Answer any 13 of 14 questions. If all 14 are attempted, only the first 13 will be corrected and considered for the total score.
- Each question is worth 5 marks, and requires roughly 14 minutes' worth of effort.

Instructions for the Exam:

- Use appropriate indentation when writing pseudocode. This is strongly encouraged.
- What is expected when providing an algorithm:
  1. Algorithm must be written as pseudocode, and not descriptive format.
  2. Add comments wherever necessary to convey your ideas clearly.
  3. Do not provide code in any specific programming language.
  4. The following must be provided whenever asked to analyze an algorithm:
     (a) Proof of correctness.
     (b) Justification of running time.
     (c) As required, space complexity.

Questions (5 marks each):

1. In the `EXACT 4SAT` problem, the input is a set of clauses, each of which is a disjunction of exactly four literals, and such that each variable occurs at most once in each clause. The goal is to find a satisfying assignment, if one exists. Prove that `EXACT 4SAT` is **NP**-complete.

2. The procedure `PERMUTE-BY-CYCLE` is used to generate a uniform random permutation. Show that each element $A[i]$ has a $\frac{1}{n}$ probability of winding up in any particular position in B. Also, show that the resulting permutation is not uniformly random, despite the goal of the procedure.

```
PERMUTE-BY-CYCLE(A,n)
1 let B[1:n] be a new array
2 offset = RANDOM(1,n)
3 for i=1 to n
4     dest = i + offset
5     if dest > n
6         dest = dest - n
7     B[dest] = A[i]
8 return B
```

3. Given two strings $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_m$, the goal is to find the length of their
longest common substring, that is, the largest k for which there are indices i and j with:
$$x_ix_{i+1} \dots x_{i+k-1} = y_jy_{j+1} \dots y_{j+k-1}.$$
Show the *dynamic programming* algorithm to achieve this goal runs in time $O(mn)$.

4. Show how to implement the stingy algorithm using a *greedy strategy* for Horn formula satisfiability (`HORN-SAT`) in time that is linear in the length of the formula (the number of occurrences of literals in it). (*Hint*: Use a directed graph, with one node per variable, to represent the implications.)

5. For the given network in Figure 1, with edge capacities as shown, find the maximum flow from S to T using Ford-Fulkerson algorithm, along with a matching cut.

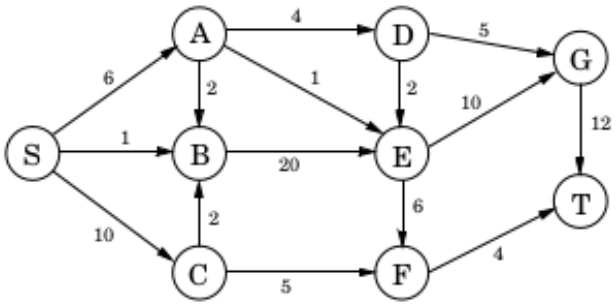

Figure 1: A given network to solve for max-flow problem

6. Another way to topologically sort a directed acyclic graph $G = (V, E)$ is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph. Explain how to implement this idea so that it runs in time $O(V + E)$. What happens to this algorithm if G has cycles?

7. State the Master Theorem, and use it to give asymptotically tight upper and lower bounds for $T(n)$ for each of the following recurrence examples.
   (a) $T(n) = 7.T(n/3) + n^2$
   (b) $T(n) = 7.T(n/2) + n^2$
   (c) $T(n) = 2.T(n/4) + \sqrt{n}$
   (d) $T(n) = T(n-2) + n^2$

8. Write the pseudocode for binary search of a search key in a sorted array, for both iterative and recursive variants of the algorithm. Argue that the running time of binary search is $\Theta(\lg n)$.

9. In the `MULTIWAY CUT` problem, the input is an undirected graph $G = (V, E)$ and a set of terminal nodes $s_1, s_2, \dots, s_k \in V$. The goal is to find the minimum set of edges in E whose removal leaves all terminals in different components.

(a) Show that this problem can be solved exactly in polynomial time when $k = 2$.

(b) Give an approximation algorithm with ratio at most 2 for the case $k = 3$.

10. Horner's algorithm:

Algorithm 1 Horner's Algorithm

Input: Array A with coefficients of $P(x)$ of degree n as $a_0, \ldots, a_n$
Input: Variable x
Output: p (evaluation of the polynomial $P(x)$ for a given value of x)

```
1: p ← 0
2: for i ← n downto 0 do
3:     p ← A[i] + x.p
4: end for
5: return p
```

(a) What is the running time of the algorithm in Ω-notation?

(b) Write the pseudocode to implement the naïve polynomial evaluation algorithm that computes each term of the polynomial as *products* and then "reduces" these terms using a *sum* operator. Compare the running time of the naïve implementation with that of Horner's algorithm.

11. A server has n customers waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. Thus, if the customers are served in the order of increasing i, then the i^{th} customer has to wait $\sum_{j=1}^{i} t_i$ minutes.

The goal is to minimize the total waiting time, given by:

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i).$$

Give an efficient algorithm using a *greedy strategy* for computing the optimal order in which to process the customers.

12. Duckwheat is produced in Kansas and Mexico and consumed in New York and California. Kansas produces 15 shnupells of duckwheat and Mexico 8. Meanwhile, New York consumes 10 shnupells and California 13. The transportation costs per shnupell are \$4 from Mexico to New York, \$1 from Mexico to California, \$2 from Kansas to New York, and \$3 and from Kansas to California. Write a *linear program* that decides the amounts of duckwheat (in shnupells and fractions of a shnupell) to be transported from each producer to each consumer, so as to minimize the overall transportation cost.

13. *Proving* **NP**-*completeness by generalization* – For each of the problems below, prove that it is **NP**-complete by showing that it is a generalization of some **NP**-complete problem, covered in the lecture.

(a) `SUBGRAPH ISOMORPHISM`: Given as input two undirected graphs G and H, determine whether G is a subgraph of H (that is, whether by deleting certain vertices and edges of H we obtain a graph that is, up to renaming of vertices, identical to G), and if so, return the corresponding mapping of $V(G)$ into $V(H)$.

(b) `LONGEST PATH`: Given a graph G and an integer g, find in G a simple path of length g.

14. How can *dynamic programming* solution be used to solve any <u>one</u> of these problems?

    (a) Chain Matrix Multiplication

    (b) 0-1 Knapsack Problem

    State the problem and explain the dynamic programming table in your answer.
