Instructions:

- Answer the questions directly on the exam pages.
- Show all your work for each question. Providing more detail including comments and explanations can help with assignment of partial credit.
- If the answer to a question is a number, unless the problem says otherwise, you may give your answer using arithmetic operations, such as addition, multiplication, “choose” notation and factorials (e.g., “9 × 35! + 2” or “0.5 × 0.3/(0.2 × 0.5 + 0.9 × 0.1)” is fine).
- If you need extra space, use the back of a page.
- No books, notes, calculators or other electronic devices are allowed. Any cheating will result in a grade of 0.
- If you have questions during the exam, raise your hand.

<table>
<thead>
<tr>
<th>Question</th>
<th>Value</th>
<th>Points Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
**Question 1.** (10 points) True or False? Indicate whether each of the following statements is TRUE or FALSE. No justification required.

1.1 (2 points): 3-COLORING can be solved by breadth first search and therefore is in $P$.

1.2 (2 points): $\sum_{i=1}^{n} 2^i = \Theta(2^n)$.

1.3 (2 points): A dynamic program that implements the following recursive form can be used to solve the subset sum problem, which asks to find a subset $S$ of numbers from $x_1, \ldots, x_n$ (all non-negative) with maximum weight subject to not exceeding a given number $W$.

$$Val(i, w) = \max\{Val(i-1, w), x_i + Val(i-1, w-x_i)\}$$

1.4 (2 points): For any flow network, and any two vertices $s, t$ there is always a flow of at least 1 from source $s$ to target $t$.

1.5 (2 points): The recurrence $T(n) = 2T(n-1) + O(1)$ solves to $\Theta(n^2)$.
Question 2.  *(20 points)* Short Answer. Answer each of the following questions in at most two sentences.

2.1 *(4 points):* In a weighted graph $G$ where all edges have weight 1, how can we use Dijkstra’s algorithm to find a minimum spanning tree?

2.2 *(4 points):* Solve the recurrence $T(n) = 3T(n/2) + O(n)$.

2.3 *(4 points):* Suppose a dynamic programming algorithm creates an $n \times m$ table and to compute each entry of the table it takes a minimum over at most $m$ (previously computed) other entries. What would the running time of this algorithm be, assuming there is no other computations.
2.4 (4 points): Why is MaxFlow in \( \mathcal{NP} \cap \text{co-NP} \)?

2.5 (4 points): Suppose \( A \) is a randomized algorithm that finds the optimal solution to some minimization problem with probability at least \( p \in (0,1) \). More precisely, if we run \( A \) on some input, it returns a candidate solution \( O \) along with the cost of \( O \), and with probability at least \( p \), we are guaranteed that \( O \) minimizes the cost function. For another parameter \( \delta > 0 \), how can we use \( A \) to find an optimal solution with probability at least \( 1 - \delta \) and what is the running time of this new algorithm?
**Question 3.** (30 points) Consider the longest increasing subsequence problem defined as follows. Given a list of numbers $a_1, \ldots, a_n$ an increasing subsequence is a list of indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $i_1 < i_2 < \ldots, i_k$ and $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_k}$. The longest increasing subsequence is the longest list of indices with this property.

3.1 (2 points): What is the longest increasing subsequence of the list 5, 3, 4, 8, 7, 10?

3.2 (4 points): Consider the greedy algorithm that chooses the first element of the list, and then repeatedly chooses the next element that is larger. Is this a correct algorithm? Either prove its correctness or provide a counter example.

3.3 (4 points): Consider the greedy algorithm that chooses the smallest element of the list, and then repeatedly chooses the smallest element that comes after this chosen one. Is this a correct algorithm? Either prove its correctness or provide a counter example.
3.4 (5 points): Consider a divide and conquer strategy that splits the list into the first half and second half, recursively computes $L = (\ell_1, \ldots, \ell_{k_L}), R = (r_1, \ldots, r_{k_R})$ the longest increasing subsequences in each half, and then, if the last chosen element in the first half is less than the first chosen index in the second half (i.e. $a_{\ell_{k_L}} \leq a_{r_1}$) returns $L \cdot R$, otherwise it returns the longer of $L$ and $R$. Is this a correct algorithm? either prove its correctness or provide a counterexample.

3.5 (15 points): Design a dynamic programming algorithm for longest increasing subsequence. Prove its correctness and analyze its running time.
Question 4. (10 points) In this problem we investigate the feedback arc-set problem which generalizes the topological ordering. Given a directed graph (which may contain cycles), the goal in feedback arc-set is to find an ordering of the vertices that minimizes the number of back edges. More precisely, if \( G = (V, E) \) is a directed graph, and \( (a_1, \ldots, a_n) \) with \( a_i \neq a_j \) is an ordering of the vertices, we define the cost as

\[
\text{cost}(a_1, \ldots, a_n) = \sum_{i<j} 1[(a_j, a_i) \in E]
\]

Here \( 1[\cdot] \) is a function that is 1 if the argument is true and zero otherwise. This is the number of edges going from right to left (backward) if we ordered the vertices with \( a_1 \) on the left and \( a_n \) on the right.

![Graph Diagram]

4.1 (1 points): In the above graph, what is the cost of \((v_1, v_2, v_3, v_4, v_5, v_6)\)?

4.2 (1 points): In the above graph, what is the cost of \((v_6, v_5, v_4, v_3, v_2, v_1)\)?

4.3 (2 points): True or False. A directed acyclic graph always has an ordering \( O \) with \( \text{cost}(O) = 0 \).

4.4 (6 points): Prove that the decision version of feedback arc set is NP-complete. That is given a directed graph and an integer \( k \), decide whether the graph has an ordering with at most \( k \) back-edges.
**Question 5.** (20 points) In this problem we investigate vertex-capacitated flow networks. We are given a directed graph $G = (V, E)$ with source $s$ and sink $t$ and a capacity $c_v$ for each $v \in V$. We want an $s - t$ flow $f$ that satisfies the usual conservation of flow constraints, but instead of satisfying edge-capacity constraints, satisfies the vertex capacity constraints $f(v) \leq c_v$. Here $f(v) = \sum_{(u,v) \in E} f_{u,v}$ is the total flow entering the node $v$. The goal is to design an algorithm for computing a maximum $s - t$ flow in a vertex-capacitated network.

5.1 (5 points): Draw a directed graph $G$ with clearly labeled source $s$ and sink $t$, where if we consider the usual edge-capacitated version of the problem (with edge capacities $c_e = 1$) we get a maximum flow with a different value than if we consider the vertex capacitated version of the problem (with vertex capacities $c_v = 1$).

5.2 (15 points): Design a polynomial time algorithm for computing the maximum flow in a node-capacitated network. Prove that the algorithm is correct and analyze its running time.
Additional space.
**Question 6.** (10 points) Consider a variant of the subset sum problem where we are given a set of numbers $x_1, \ldots, x_n$ and need to partition them numbers into sets $S_1, \ldots, S_K$ such that for each $k \in \{1, \ldots, K\}$, $\sum_{i \in S_k} x_i \leq W$ for some target $W$. The goal is to minimize $K$, the number of sets in the partition. We will study a simple approximation algorithm for this problem. The algorithm considers the items in order, and forms the first set $S_1$ by repeatedly adding the numbers $x_1, x_2, \ldots$ until the next number would exceed the target $W$. Then it proceed to construct the next set.

6.1 (2 points): Give an example input where this algorithm does not use the minimum number of sets.

6.2 (2 points): Derive a lower bound on $K^*$ the smallest possible number of sets in the partition in terms of the target $W$ and the total weight $X = \sum_{i=1}^{n} x_i$.

6.3 (6 points): Use this lower bound to prove that this greedy algorithm always produces a number of sets $K$ that is at most $2K^*$. 