You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Recall that a ***tree*** is a connected graph with no cycles. A graph is ***bipartite*** if we can color its vertices black and white, so that every edge connects a white vertex to a black vertex.

   (a) ***Prove*** that every tree is bipartite.

   (b) Describe and analyze a fast algorithm to determine whether a given graph is bipartite.

2. Describe and analyze an algorithm SHUFFLE($A[1..n]$) that randomly permutes the input array A, so that each of the $n!$ possible permutations is equally likely. You can assume the existence of a subroutine RANDOM(k) that returns a random integer chosen uniformly between 1 and k in $O(1)$ time. For full credit, your SHUFFLE algorithm should run in $O(n)$ time. *[Hint: This problem appeared in HBS 3½.]*

3. Let G be an undirected graph with weighted edges.

   (a) Describe and analyze an algorithm to compute the *maximum* weight spanning tree of G.

   (b) A ***feedback edge set*** of G is a subset F of the edges such that every cycle in G contains at least one edge in F. In other words, removing every edge in F makes G acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of G.

   [Hint: Don't reinvent the wheel!]

4. Let $G = (V, E)$ be a connected directed graph with non-negative edge weights, let s and t be vertices of G, and let H be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe and analyze an algorithm to choose the best edge to reinsert. For full credit, your algorithm should run in $O(E \log V)$ time. *[Hint: This problem appeared in HBS 6¾.]*

5. Describe and analyze an efficient data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i.

   - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
   - Given an index i, return $X[i]$.
   - Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

   For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible.