Important Note: You are required to turn in only your solution to Problem 1. The other problems are optional and won’t be graded. However, you are strongly advised to solve those problems to improve your problem-solving skills.

Problem 1: Let $S$ denote the set $\{1, 2, \ldots, n\}$. Suppose we have a function $f$ from $S$ to $S$. The function $f$ may not be one-to-one. For any subset $X$ of $S$, the restriction of the function $f$ to $X$, denoted by $f_X$, is defined as follows: for each $j \in X$, $f_X(j) = f(j)$.

Given the set $S$ with $n$ elements and a function $f$ from $S$ to $S$, give a polynomial time algorithm for finding a subset $X$ of $S$ that satisfies the following two conditions:

(i) The range of $f_X$ is $X$ itself.

(ii) $|X|$ is a maximum among all sets that satisfy Condition (i).

Assume that the function $f$ is given by an array $F[1..n]$ so that the value $F[i]$ represents $f(i)$.

Example: Suppose $S = \{1, 2, \ldots, 7\}$ and the function $f$ is defined by $f(1) = 3$, $f(2) = 1$, $f(3) = 1$, $f(4) = 5$, $f(5) = 5$, $f(6) = 4$, and $f(7) = 6$. One solution to this instance is $X = \{1, 3, 5\}$.

Your answer must include a high level description of the algorithm, a proof of its correctness and an estimate of its running time. Try to make the algorithm as efficient as you can. (It is possible to obtain an $O(n)$ algorithm for this problem.)

Problem 2: Let $G(V, E)$ be a connected undirected graph and let $s$ be a node in $V$. Carrying out a BFS of $G$ starting at $s$, leads to a spanning tree $T$. Further, when a DFS of $G$ is carried out starting at $s$, we get the same spanning tree $T$. Prove that $G = T$. In other words, prove that $G$ cannot contain any edges which are not in $T$.

Problem 3: You are given an array $X[1..n]$ containing integer values. These values may be positive, negative or zero. We want to know whether there is a subarray $X[p..q]$, where $1 \leq p \leq q \leq n$, such that the sum of the elements in the subarray is equal to zero. Your task is to design an algorithm with a running time of $O(n \log n)$ for this problem.

Your answer must include a clear description of the algorithm, a proof of its correctness and a proof that its running time is $O(n \log n)$.

Note: The solution that your instructor has for this problem is not based on divide-and-conquer or dynamic programming or the greedy approach.