1. Two problems. (5+5 Pts)
   
   (a) Design an algorithm to find the largest and the second largest of
       \( n \) numbers using comparisons. How many comparisons does your
       algorithm use?
   
   (b) What are the types of edges one encounters while doing DFS on a
       directed graph. Give an example graph where these edges occur.
2. Given array $A$ and $B$, both sorted in increasing order, and an integer $k$, the problem is to find the $k$th smallest element in the union of both arrays.

(a) (3 Pts) Design an $O(k)$ time comparison-based algorithm to solve this problem.

(b) (7 Pts) Design a faster algorithm by binary search (or divide and conquer). How much time (number of comparisons) does your algorithm take?

(c) (Extra Credit 10 Pts) Given a two dimensional array $A[1 \cdots n, 1 \cdots n]$ in which each row is sorted in increasing order and each column is sorted in increasing order, how much time do you think it takes to find the $k$th smallest number? Justify.

3. (10 Pts) Suppose you are in a dark room and are given $n$ nuts and $n$ bolts of different sizes. Each nut matches exactly one bolt and vice versa. Since the room is dark and the nuts and bolts are all nearly the same size, you can’t tell whether one bolt is smaller than the other, or if one nut is bigger than the other. Only thing you can do is to try to “compare”, ie., match a nut with a bolt, it would be either too big or too small for the bolt or it would be just right for the bolt. Find the correct matching. How many comparisons does your algorithm need?

Hint. You have to use a randomized algorithm. Use nuts to partition bolts and vice versa.
4. (10 Pts)
Let $T = (V, E)$ be an undirected tree with weights $w(u, v)$ on the edges $(u, v) \in E$. A subset $M \subseteq E$ of edges in $G$ is a matching, if no pair of edges of $M$ share an endpoint. Design an algorithm for computing the maximum weight matching in $T$.

5. (10 Pts) Consider the min-cut algorithm that performs random edge contractions. Design an algorithm to find a smallest and a second smallest network cut; a second smallest may or may not have the same number of edges as a smallest network cut. Use either of the two methods below.

(a) Repeatedly run our mincut algorithm keeping track of a smallest and second smallest cut. First show that when the number of vertices exceeds two, there is always a smallest cut $(C; C')$ and a second smallest cut $(D; D')$ for which one of $C \cap D, C \cap D', C' \cap D,$ and $C' \cap D'$ is empty. Then perform a sequence of random edge contractions until a small, fixed number (which must be at least three) of vertices remain, at which time use a deterministic method to find a smallest and second smallest (surviving) cut. Analyze the probability that a specific selection of smallest and second smallest cuts survives, and hence determine the expected number of times the process can be run to have probability at least $1/2$ of having found the desired cuts.

(b) An alternative method is to first determine (with appropriate probability) a minimum cut. Then, repeatedly select an edge of the cut at random, contract the edge, and find a minimum cut in the resulting graph (this is a candidate for a second smallest cut in the original graph). Specify the details of this approach, and determine the expected time to have probability at least $1/2$ of having found the desired cuts.