Question 1. Explain in your own words how the Ford-Fulkerson algorithm can be implemented to run in $O(mC)$ time, where m is the number of edges in the network and C is the total capacity of all edges leaving the source node s.

1

Question 2. Let $G = (V, E)$ be a bipartite graph with $V = L \cup R$ (that is, every edge in G has one endpoint in L and one in R). A *matching* in G is a set of edges $M \subseteq E$ such that no two edges in M share an endpoint. A *maximum matching* is a matching with the greatest possible number of edges.

Design an algorithm to find a maximum matching in a given bipartite graph. Describe your algorithm, prove its correctness, and analyze its running time.

2

Question 3 (K&T 7.6)**.** Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations*. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

We wish to connect each client to exactly one of the base stations, but our choice of connections is constrained in the following ways. First, there is a *range parameter*, denoted by r: a client can only be connected to a base station that is within distance r. There is also a *load parameter* L: no more than L clients can be connected to any single base station.

Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters r and L, decide whether every client can be connected simultaneously to a base station, subject to the constraints r and L.

3

The last two questions on this problem set are due to Jeff Erickson: `http://www.cs.illinois.edu/~jeffe/teaching/algorithms`.

Question 4. Suppose we are maintaining a data structure under a series of n operations. Let $f(k)$ denote the actual running time of the kth operation. For each of the following functions f, determine the resulting amortized cost of a single operation. For amortized costs other than $\Theta(1)$, be sure to argue why your cost is also a *lower* bound, *i.e.* why it is not possible to do any better.

1. $f(k)$ is the largest integer i such that 2^i divides k.
2. $f(k) = k$ if k is a power of 2, and $f(k) = 1$ otherwise.
3. $f(k) = k$ if k is a Fibonacci number, and $f(k) = 1$ otherwise.
4. $f(k) = k$ if k is a perfect square, and $f(k) = 1$ otherwise.
5. Let T be a *perfect* binary search tree, storing the integer keys 1 through n. $f(k)$ is the number of ancestors of node k.

4

A perfect binary tree is one in which every node has two children and all leaves have the same depth. Thus, a perfect binary tree with height h has exactly $2^h - 1$ nodes.

Question 5. An *extendable array* is a data structure that stores a sequence of items and supports the following operations:

- ADDTOFRONT(x) adds x to the *beginning* of the sequence.
- ADDTOEND(x) adds x to the *end* of the sequence.
- LOOKUP(k) returns the kth item in the sequence, or NULL if the current length of the sequence is less than k.

Describe and analyze a *simple* data structure that implements an extendable array. Your ADDTOFRONT and ADDTOBACK algorithms should take $O(1)$ *amortized* time, and your LOOKUP algorithm should take $O(1)$ *worst-case* time. The data structure should use $O(n)$ space, where n is the *current* length of the sequence.