**Algorithms and Data Structures**
**Winter Term 2015/16**
**Exercises for Unit 27**

1. In a basketball tournament $n$ teams play against each other (possibly multiple times) in a pre-determined sequence. The winner of a game gets one point, the loser gets no points, and there are no draws. We assume that at a certain moment during the tournament, the score of the $i$-th team is $score(i)$, and the number of times the $i$-th and the $j$-th team still need to play against each other is $games(i, j)$.

   a) Given access to the tables *scores* and *games*, design an algorithm to determine whether a certain team still has a chance to win the tournament. What is the running time of your algorithm?

   b) Will the approach also work for a chess tournament, where the winner gets two points, and in a draw both players get one point each? How about a football tournament (winner three points, loser no points, draw both one point each)?

2. The network administrators in your company have diagnosed a failure in their network. The network carries traffic from a source $s$ to a target $t$. We will model it as a directed capacitated network $G$, in which every edge has capacity 1, and every edge lies on at least one $s$-$t$-path. When everything is running smoothly, the maxflow in $G$ has value $k$. You are called for assistance, because the administrators believe (for reasons we won't go into here) an attacker has destroyed the $k$ edges of a min-cut, and we'll assume that they are right.

   They have a monitoring tool *ping* on vertex $s$, and if you issue $ping(v)$, for a given vertex $v$, it will tell you whether or not there is an $s$-$v$-path in the remaining network. Since it is impractical to query every vertex of the network, they want to determine the cut in the network using as few *ping* commands as possible.

   Design an algorithm that reports the full set of vertices that are not reachable from $s$ and uses at most $O(k \log n)$ ping operations. Argue why it is correct and achieves the bound.

3. Your friends have implemented a version of the Ford-Fulkerson algorithm, but they haven't fully mastered the residual-graph idea. In their algorithm, an edge $(u, v)$ in the residual graph exists if and only if $c_f(u, v) = c(u, v) - f(u, v) > 0$ and $c(u, v) > 0$. Hence, their *reduced residual graph* $G_f$ contains only subsets of the edges in $E$, but never opposite edges $(v, u) \notin E$ with $(u, v) \in E$. Now the algorithm searches only for augmenting paths in this reduced residual graph, and it terminates if no such path is found. It is not known to you how they choose among several augmenting paths.

   It's hard to convince them to reimplement the algorithm, since they claim it's incredibly fast. Also, while they are aware that it might not give a maximum flow, they believe it returns a constant-factor approximation: There is an absolute constant $b$ (independent of $n$, $m$ and $c$) such that for every network $G = (V, E, c, s, t)$ and for every choice of augmenting paths in the reduced residual graph, the algorithm returns a flow $f$ with $|f| \geq |f_{\max}|/b$, where $f_{\max}$ is a maximum flow in $G$.

   Prove this statement or give a counterexample to it.