Algorithms and Data Structures 1

Student number: s215158
Hand-in: Exam

1.1)

The party can be model as a graph where each *person* is a node, the nodes are connected with *friendships relations* described by undirected edges.

1.2)

To solve this problem, given a graph as described above, we can apply the bipartite graph algorithm. The algorithm uses BFS on the graph and for each edge checks if its endpoints are in the same layer, the algorithm should return "True" if the teams can be created (the graph is bipartite) and "False" otherwise. The correctness of the algorithm comes from the proof of the lemma of bipartite graphs "The graph is bipartite iff all cycles have even length"; thus, there is no edges between nodes of the same layer. In our case means there is no friendship between people of the same team. The running time is given by the BFS algorithm and is therefore: $O(x + y)$.

1.3)

To solve this problem regarding invitations we can use the same graph representation of the party and run a slightly modified version of DFS. The algorithm, starting at node p_0, should keep a counter c, initialized to zero, and update it every time a node is marked as visited, in this way at the end, we can check if $c \geq k$ and $c \leq 3x/4$ and return "True" if the conditions are respected and "False" otherwise. The correctness comes from the way DFS runs through the graph, only visiting connected nodes that are not marked (not visited already), therefore this ensures that we don't count the same person twice. The running time is given by DFS, and it's not changed since we only add a counter, so it is: $O(x + y)$.

1.4)

To solve this problem, we can construct a graph of the party as proposed in 1.1 and add the strength of friendships at each edge. Now that we have the representation of the party, we can run a slightly modified version of Prim's algorithm from the node p_0. The algorithm initializes a variable s to one, and checks, at every edge addition, that the edge is indeed the lightest and compares the maximum encountered friendship strength s with the weight of the edge, if it's grater then it updates s and keeps going. The correctness comes from the way Prim's algorithm runs through the graph and tries to find the MST starting from the given node. It's implied in the cut property that states, "for any cut, the lightest cut edge is in the MST", thus adding the lightest cut edge to the tree will result in the MST after x-1 steps. At the end the variable s is returned so we know that we can reach every person in the party, starting from p_0 using only friendships of strength at most s. The running time is given by Prim's algorithm, and it's not changed since we only add a variable, so it is: $O(y \log x)$ following the binary heap implementation of the slides.