Advanced Algorithms. Assignment 1

Exercise 1.
Let c be any fixed positive integer. We consider the problem of splitting a graph into small pieces by deleting some nodes. Formally, the problem is:
Given a graph $G = (V, E)$, delete a subset $U \subset V$ of nodes (and all their incident edges) from G such that, in the remaining graph, every connected component has at most c nodes, and the size $|U|$ is as small as possible.
The problem appears, e.g., in data analysis, where the nodes represent data items, an edge means similarity, and the data shall be partitioned into small clusters, thereby neglecting as few data as possible.

Give a polynomial-time algorithm with approximation ratio no worse than $c + 1$. That is, if there exists a solution U with $|U| = k$, your algorithm should delete at most $(c + 1)k$ nodes. Make sure that you accurately prove the approximation ratio, and argue why you need only polynomial time.

Terminology remark: Note that a connected component C is, by definition, a subgraph that is connected, and also maximal with this property. That is, no edges must exist between nodes in C and nodes outside C.

Exercise 2.
In the course material we have seen the Center Selection problem. Another classic geometric problem in the same application domain is:
Given n points with Cartesian coordinates (x_i, y_i) in the plane, and positive weights w_i, find one(!) point (x, y) that minimizes the weighted sum of the Euclidean distances to the given points. For formal clarity: We want to minimize

$$\sum_{i=1}^{n} w_i \cdot \sqrt{(x - x_i)^2 + (y - y_i)^2}.$$

This is, at least, not very easy to solve exactly. In the following we therefore propose a rough but rather quick and simple approximation algorithm: Instead of the Euclidean distance, take the Manhattan distance and minimize

$$\sum_{i=1}^{n} w_i \cdot (|x - x_i| + |y - y_i|).$$

2.1. Explain how the point that minimizes the weigthed sum of Manhattan distances can be found in polynonial time. That is: Sketch an algorithm and argue why it is correct.

Some hints: Split the problem in two "independent" one-dimensional problems, that is, work separately in x- and y-direction. To get an idea where to find the optimal coordinate x (and y), study what happens to the sum when you slightly increase or decrease x.

2.2. Show that this algorithm has an approximation ratio $\sqrt{2}$. More specifically: The point found by the algorithm has a weighted sum of Euclidean(!) distances that is at most $\sqrt{2}$ times the optimal sum.

Technically this should not be too difficult, just see that you compare the correct things, and prove all geometric claims that you need.