1. Kleinberg & Tardos Chapter 11, question 4

2. As with satisfiability, we can also consider optimization problems based on subset sum. Here is one we will call MAX-SUBSET-SUM: you are given a set of non-negative integers \( X \) and a target \( t \), find a subset \( X' \) of \( X \) with the largest sum \( \leq t \).

Here is a heuristic for this problem. First, let \( X_2 \subseteq X \) be the subset of elements of \( X \) that are \( > t/2 \). Let \( S \) be the set consisting of the largest element of \( X_2 \) if it is nonempty, or the empty set otherwise. Now sort the remaining elements of \( X - X_2 \) in non-increasing order.

For each element in this list, add it to \( S \) if doing so would not cause \( S \)'s sum to exceed \( t \).

(a) Show that the above heuristic is a \( \frac{1}{2} \)-approximation for MAX-SUBSET-SUM.

(b) Show how to extend this heuristic into a \( \frac{k}{k+1} \)-approximation for any \( k \geq 2 \). What is the running time of your method for a given \( k \)?

3. A self-organizing data structure is reorganized during execution in response to a sequence of operations, with the goal of achieving good performance on the actual, initially unknown, sequence. Often, the majority of operations concern a small number of elements, and so optimizing the access time to these elements can improve performance overall. In this problem, we will analyze a simple self-organizing linked list. In this linked list, whenever we access a list node, we move that node to the head of the list (pushing the rest of the elements back one position). We’ll say that it costs one operation (\$1, if you like) to access a node of the list and check its value, so that walking a list to access the \( k \)th element of that list costs \( k \) operations.

(a) Consider some arbitrary list containing \( n \) distinct elements, and suppose we have used the self-organizing algorithm when accessing \( k \) out of the \( n \) elements. For each of the remaining \( n - k \) elements, exactly how many more operations would it take to access that element in the self-organizing list than in the original list?

(b) Now, show that for our arbitrary list containing \( n \) distinct elements, and any sequence of \( T \) accesses to the list elements, the self-organizing algorithm uses at most twice as many operations as if we had kept the list order static.

(c) Finally, show that for every possible static ordering of the \( n \) distinct elements and sequence of \( T \) accesses, if these accesses cost \( C \) operations in total, then the self-organizing algorithm uses at most \( 2C + n^2 \) operations in total, for a competitive ratio of at most \( 2 + \frac{n^2}{T} \) as \( T \to \infty \) against all static lists.