Introduction to Algorithms	**Problem Set 10**
CS 4820, Spring 2019	**Due 11:59pm Sunday, May 5**

Hand in your solutions electronically using CMS. Each solution should be submitted as a separate file. Collaboration is encouraged while solving the problems, but:

1. **list the names of those with whom you collaborated;**
2. **you must write up the solutions in your own words;**
3. **you must write your own code.**

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size.

(1) *(15 points)* Consider the problem of MAX-SPREAD, where we are given as input a list of locations $\{1, \ldots, n\}$, and a distance function d (supplied as a $n \times n$ matrix), where $d(i, j)$ denotes the distance between location i and location j, and an integer C. The distance function is restricted to be a metric (see below for a definition). For a set $S \subseteq \{1, \ldots, n\}$, define $spread(S) = \min_{i,j \in S, i \neq j} d(i, j)$. Your task is to output a subset $S \subseteq \{1, \ldots, n\}$ of cardinality C that **maximizes** $spread(S)$.

(1a) *(5 points)* Consider the decision version of MAX-SPREAD, denoted by D-MAX-SPREAD, where the input, in addition to the input of MAX-SPREAD, contains an integer α, and it is a YES instance if and only if there exists a subset $S \subseteq \{1, \ldots, n\}$ of cardinality C such that $spread(S)$ **is at least** α. Prove that D-MAX-SPREAD is NP-complete.

(1b) *(10 points)* Design an efficient 2-approximation algorithm for MAX-SPREAD.

[*Definition of a metric:* The distance function $d : \{1, \ldots n\} \times \{1, \ldots, n\} \to \mathbb{R}$ is a metric if it satisfies the following constraints: for all $i, j, k \in \{1, \ldots, n\}$,

- $d(i, j) \geq 0$
- $d(i, j) = 0$ if and only if $i = j$,
- $d(i, j) = d(j, i)$,
- $d(i, j) + d(j, k) \geq d(i, k)$.]

(2) *(15 points)*
Recall that in the Knapsack Problem, one is given a set of items numbered $1, 2, \ldots, n$, such that the i-th item has value $v_i \geq 0$ and weight $w_i \geq 0$. Given a total weight constraint W, the problem is to choose a subset $S \subseteq \{1, 2, \ldots, n\}$ so as to maximize the combined value, $\sum_{i \in S} v_i$, subject to the weight constraint $\sum_{i \in S} w_i \leq W$.

(a) Consider the following *greedy algorithm*, GA.

1. Eliminate items whose weight w_i is greater than W.
2. For each remaining i, compute the *value density* $\rho_i = v_i / w_i$.
3. Sort the remaining items in order of decreasing ρ_i.
4. Choose the longest initial segment of this sorted list that does not violate the size constraint.

Also consider the following *even more greedy algorithm*, EMGA.

1. Eliminate items whose weight w_i is greater than W.
2. Sort the remaining items in order of decreasing v_i.
3. Choose the longest initial segment of this sorted list that does not violate the size constraint.

For each of these two algorithms, give a counterexample to demonstrate that its approximation ratio is not bounded above by any constant C. (Use different counterexamples for the two algorithms.)

(b) Now consider the following algorithm: run GA and EMGA, look at the two solutions they produce, and pick the one with higher total value. Prove that this is a 2-approximation algorithm for the Knapsack Problem, i.e. it selects a set whose value is at least half of the value of the optimal set.

(c) By combining part (b) with the dynamic programming algorithm for Knapsack presented in class, show that for every $\varepsilon > 0$, there is a Knapsack algorithm with running time $O(n^2 s/\varepsilon)$ whose approximation ratio is at most $1 + \varepsilon$. Here, s denotes the maximum number of bits in the binary representation of any of the numbers v_i, w_i ($i = 1, \ldots, n$). [We will see an algorithm in class with running time $O(n^3 s/\varepsilon)$.] In your solution, it is not necessary to repeat the proof of correctness of the algorithm that will be presented in class.