Due: Wednesday 9/12 before 9:15

1. **(4.2)** For each of the following statements, decide whether it is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

   (a) Suppose we are given an instance of the Minimum Spanning Tree problem on a graph G, with edge costs that are all positive and distinct. Let T be a minimum spanning tree for this instance. Now suppose we replace each edge cost c_e by its square c_e^2, thereby creating a new instance of the problem with the same graph but different costs.
   True or false? T must still be a minimum spanning tree for this new instance.

   (b) Suppose we are given an instance of the Shortest s-t Path problem on a directed graph G. We assume that all edge costs are positive and distinct. Let P be a minimum-cost s-t path for this instance. Now suppose we replace each edge cost c_e by its square c_e^2, thereby creating a new instance of the problem with the same graph but different costs.
   True or false? P must still be a minimum-cost s-t path for this new instance.

2. **(4.13)** A small business—say, a photocopying service with a single large machine— faces the following scheduling problem. Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take time t_i to complete. Given a schedule (i.e., an ordering of the jobs), let C_i denote the finishing time of job i. For example, if job j is the first to be done, we would have $C_j = t_j$; and if job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent of the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

   Design an efficient algorithm to solve this problem. That is, you are given a set of n jobs with a processing time t_i and a weight w_i for each job. You want to order the jobs so as to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

   Example. Suppose there are two jobs: the first takes time $t_1 = 1$ and has weight $w_1 = 10$, while the second job takes time $t_2 = 3$ and has weight $w_2 = 2$. Then doing job 1 first would yield a weighted completion time of $10 \cdot 1 + 2 \cdot 4 = 18$, while doing the second job first would yield the larger weighted completion time of $10 \cdot 4 + 2 \cdot 3 = 46$.

3. **(4.25)** Suppose we are given a set of points $P = \{p_1, p_2, \ldots, p_n\}$, together with a distance function d on the set P: d is simply a function on pairs of points in P such that

   - $d(p_i, p_j) = d(p_j, p_i) > 0$ if $i \neq j$, and
   - $d(p_i, p_i) = 0$ for each i.

   We define a *hierarchical metric* on P to be any distance function τ that can be constructed as follows. We build a rooted tree T with n leaves, and associate with each node v of T (both leaves and internal nodes) a *height* h_v. These heights must satisfy the properties that

- $h(v) = 0$ for each leaf v, and
- if u is the parent of v in T, then $h(u) \geqslant h(v)$.

We place each point in P at a distinct leaf of T. Now, for any pair of points p_i and p_j, their distance $\tau(p_i, p_j)$ is defined as follows. We find the least common ancestor v in T of the leaves containing p_i and p_j, and define $\tau(p_i, p_j) = h_v$.

We say that a hierarchical metric τ is *consistent* with our distance function d, if for all pairs i, j, we have $\tau(p_i, p_j) \leqslant d(p_i, p_j)$.

Give a polynomial-time algorithm that takes the distance function d and produces a hierarchical metric τ such that

(a) τ is consistent with d, and

(b) if τ' is any other hierarchical metric consistent with d, then $\tau'(p_i, p_j) \leqslant \tau(p_i, p_j)$ for each pair of points p_i, p_j.

4. **[Implementation question]** (Help Hermione Granger!) Hermione is back to Hogwarts for her third year, and wants to take many classes, several of which unfortunately overlap in schedule. (The schedule is designed to accomodate average students.) To help her, Prof. McGonagall gives Hermione a time-turner, that is, a device that allows its owner to turn back time and thus appear in more than one place at a time. Time-turners are carefully controlled by the Ministry of Magic, and so Hermione should not use hers any more than strictly necessary. Write a function `schedule` that takes as input a single argument, a list of the form `[ ('Potions', 10), ('Transfigurations', 11), ('Charms', 10.5) ]`. In this list, every entry is a pair (tuple) of a string (describing the name of the class) and a number (describing the start time of the class). (For simplicity, let the numbers be floats, and interpret, for example, 10.5 as 10:30. Also for simplicity, assume that every class lasts exactly 50 minutes and the remaining 10 are sufficient to reach wherever the next class is held: thus two classes do not conflict if and only if their start time differs by at least 1 hour.) Your function should compute a schedule for Hermione by grouping classes into a minimum number of groups such that classes in each group do not overlap, and thus can be taken without using the time-turner. Then Hermione only needs to use the time-turner between different groups of classes. (In the three-class list above, she could group Potions and Transfigurations together, then use the time-turner to go back and take Charms as well, because that class period overlaps the other two.) Your function should output the groups of classes in the form of a list of lists: for the example above, the function should return the list `[ [ 'Potions', 'Transfigurations' ], [ 'Charms' ] ]`. (It doesn't matter how long each list is; even though a time-turner can be turned once to go back an hour, but requires two turns for two hours etc., we won't worry about the total number of turns necessary, just the total number of discrete times the device is used.) Within each list, the classes should be sorted alphabetically. For the sorting, you may use the built in list `sort` method.