CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2014**

Assignment 2: Due Wednesday June 25, Midnight

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use **any** sources (textbooks, online notes, friends) please cite them for your own safety.
You can use those data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sorts running time is $\mathcal{O}(n \log n)$.

Every time you are asked to design an efficient algorithm, you should provide both a short high level explanation of how your algorithms works in plain English, and the pseudo-code of your algorithm similar to what we've seen in class. State the running time of your algorithm with a brief argument supporting your claim. You must prove that your algorithm finds an optimal solution!

1 When I was your age...

Remember when we were kids, we played with marbles?[1] We used to also collect them and exchange them for different types. Clearly each type had a different value, you could trade two of these "ugly" ones for this pretty one.

Suppose you have a group of n friends. Each one of them collects one specific type of marbles. Suppose also that for each pair of friends $i \neq j$, you've computed a ratio r_{ij}: one marble from i gives you r_{ij} marbles from j. Notice that r_{ij} can be a fraction; that is if $r_{ij} = \frac{1}{2}$ then if you buy two marbles from friend i, you can exchange them for one marble from friend j.

A marbling cycle is a fixed order $x_1, x_2, ..., x_k$ where you successively exchange marbles from friend x_i with marbles from friend x_{i+1} for all $1 \leq i < k$, and you exchange marbles from x_k with marbles from x_1. More precisely, you buy your favourite type of marbles from the friend who collects them (that's x_1). You then perform a sequence of exchanges from x_1 to x_2, x_2 to x_3 etc until x_k to x_1. And you hope that by the end of your marbling cycle, you end up with more marbles of your favourite type!

This only happens if the product of the ratios along this marbling cycle is above 1! Give a poly-time algorithm that checks whether you will be able to end up with more marbles at the end of your cycle.

2 Sucks to be broke!

You can assume you do not pay for the first move.

Summer is still here but you're broke already and are trying to save money for your next trip! You found a night job at a hospital: Saint Hell's Hospital (SHH). SHH operates two main clinics, one in Toronto (SHHT) and one in Ottawa (SHHO), and by law they have to always have k night guards in total between the two locations. They hired you to meet the law requirements, which makes your job more flexible (or not). For

[1] No? Just me? Maybe I'm too old ...

the n coming months, you have to work at either SHHT or SHHO and you have the freedom to chose the clinic you go to.

For every month i, you make T_i or O_i dollars depending where you work. However, if you decide to switch clinics from month i to month $i+1$, you incur moving expenses, a fixed price E for the move. For simplicity, suppose all your other expenses (rent, food...), except the move, are covered by some generous God. By the end of the n^{th} month, you want to maximize your savings.

Input : A list of values $T_1, T_2, ..., T_n, O_1, O_2, ..., O_n$ where T_i, O_i is your income on month i at $SHHT, SHHO$ respectively, and a value E representing the moving expense.
Output : A sequence S of n locations that maximizes $v(S)$, where location i represents the clinic you will work at during month i and $v(S)$ is the total value of your savings.

Devise an algorithm to solve this problem efficiently. Give a high-level description of how your algorithm works, state the time complexity of your algorithm, and prove that your algorithm is optimal.

3 Piw Piw!

x_i is the number of infected patients you get to see and cure at minute i. Patients who showed up at minute i but were not cured die. You do not get to see them at minute $i+1$.

Working for SHH isn't so bad after all. It sure is quiet and boring at times and not much is happening on Reddit over night, but you manage to keep yourself busy. On a boring night however, you get a call from the emergency room. There was an outbreak and they need to administer the antidote to the affected patients. You were asked to come help [2].

You run upstairs where you are assigned a room, an antidote gun, and a list $\mathcal{L} = \{x_1, x_2, ..., x_n\}$ of number of patients per minute that you will get to *hopefully* save. This means, at minute i, you will have x_i infected patients and you want to save as many of them as possible. The gun you're given pumps the antidote from a heated container, and the whole process works as follows:

- If you let the antidote container heat for j minutes, then you get $f(j)$ dosages (shots) to cure $f(j)$ patients.
- If you use the gun at the i^{th} minute, and it has been j minutes since you last used it, then you will cure $\min(x_i, f(j))$ patients. Of course, if you use $\min(x_i, f(j))$ dosages, then the gun will be completely empty and you will need to wait for more dosages to be available.
- When you first got to the room, the gun was empty. This means, if you started using it at the j^{th} minute, then you will be able to cure up to $f(j)$ patients. In other words, if you first use the gun at minute j, then you will cure up to $f(j)$ patients.

Given the list $\mathcal{L} = \{x_1, x_2, ..., x_n\}$ and the function $f(\cdot)$, you want to select when you will use the antidote gun in order to cure as many patients as possible.

Construct an efficient algorithm that returns the maximum number of patients you can cure. Give a high-level description of how your algorithm works, state the time complexity of your algorithm, and prove that your algorithm is optimal.

[2]I don't know why they believe a night guard is capable of doing this...

4 Everybody deserves good health care!

You saved the most patients out of all the staff! [3] Unfortunately many patients passed away. There just weren't enough staff to help all the patients; so you decided to run for mayor elections again to change this situation. Everybody deserves good health services! This was your selling point at the town elections. You've been elected (Gee you're good at politics) and as a good politician *cough again* you want to improve everyone's access to hospitals and health services in general.

A good access to health services means two things: 1. A household should not travel further than s kilometers to get to a hospital, and 2. no hospital should be responsible for more than t households.

Before investing into building new hospitals, you want to check if the current setup of the town is good[4] (i.e. it meets the requirements above).

Input : A list A of n houses and a list B of m hospitals, where every house and hospital is represented by a pair (x, y) on the plane, and s and t.
Output : "Good" if each household has access to a hospital under the restrictions above, and "Eeek" otherwise.

Think of "Good" as an assignment of houses to hospitals that satisfies the constraints.

Devise an algorithm to solve this problem. Give a high-level description, state the time complexity of your algorithm, and prove that it is optimal.

5 Eeek!

It turned out the town needs one more hospital. You selected the location and hired all the necessary staff, n' people in total $p_1, p_2, ..., p'_n$. There are different tasks (k of them) to be done at the hospital, from surgery to mopping the floor etc. Suppose there are n tasks to be completed in a given day, and assume (clearly) that $n \geq k$ (might have to perform more than one surgery in a given day). Suppose that each task takes one hour to complete, and only one person can work on one instance of any task. Each person p_i works a total of h_i hours per day and can perform a subset S_i of the of task types.

Each job j_i is a pair (t_i, c_i) where t_i is the task type (surgery, etc.) and c_i is an integer representing the number of times we have to do that job in a given day.

Give an algorithm that checks whether it is possible to get all the jobs done in a given day. If it is possible, show how to extract the list of jobs each p_i will have to complete. Prove the correctness of your algorithm.

Example

Suppose the set of tasks is {mopping, surgery, reception work, checking blood pressure}, and suppose on a given day, the jobs are (mopping, 5), (surgery, 10), (checking blood pressure, 6) and (reception work, 2).

Now each staff is represented with (p_i, h_i, S_i) and suppose we have 4 people with:

[3]Not bad for a night guard :D
[4]the breakout night was an exception..

(Alice, 6, (mopping, reception))
(Bob, 3, (mopping, checking blood pressure, reception))
(Sarah, 8, (surgery))
(Eve, 6, (surgery, checking blood pressure))

Now for this specific day, with these specific jobs and hours, it is possible to get all the jobs done as follows:
Alice will spend 4hrs mopping and 2hrs doing reception work.
Bob will do 1hr mopping and 2hrs checking blood pressure.
Sarah will spend 8hrs doing surgery.
Eve will do 2hrs of surgery and 4 hours checking blood pressure.

No person goes over their hours, and all the jobs get to be done.