Algorithms and Complexity 2024
Mastery Test 1: Algorithms

Mastery Test 1 must be solved individually. No collaboration or outside help is allowed: you are not allowed to discuss the test with others until after the oral presentations are completed, and taking help from generative AI to solve the problems is not allowed. See further the EECS Code of Honour.

Written solutions are submitted through Canvas and are *due on Wednesday, February 28th, 19:00.* The written report should be in English and the file must be in pdf format.

The mastery test is a mandatory and graded part of the course. The test consists of four tasks of varying difficulties. **The problems are sorted by name, not by difficulty!**

Key parts of your algorithms should be described with pseudo code, and you should also explain the core ideas of your algorithms in text. For instance if you are using one of the algorithm design paradigms discussed in the course (e.g., divide and conquer or dynamic programming), you should clearly say so and explain how you are using the paradigm. In all problems you should argue for the correctness of your algorithm and give an analysis of its time complexity. You may use standard algorithms and data structures from Ch 1-7 of the book and Lecture 1-7 as "black boxes". For instance if you need to sort a list you can use the fact that sorting takes $O(n \log n)$ time and do not have to explain an algorithm achieving this, or if you need to find a maximum flow in a graph you may assume access to a function solving this using one of the known algorithms for the problem, and do not need to explain how that algorithm works or prove its correctness (but you need to know what its time complexity is).

If you have any questions about any of the problems, send an e-mail to algokomp-help@kth.se. Good luck!

Requirements For each of the four problems, the requirements are as follows. An item labelled "mostly" indicates that this needs to be present, but that there is room to correct minor errors or issues in the oral presentation.

Component	**Requirement for written solutions**	**Requirement for oral presentation**
Clear verbal/illustrated algorithm description	Mostly	Yes
Pseudo code is structured, understandable, and at appropriate level of detail	Mostly	Yes
Algorithm is correct, solves the given problem	Yes	-
Algorithm is efficient enough, satisfies time complexity requirement	Yes	-
Correct analysis of time complexity	Mostly	Yes
General/high-level idea(s)[†] for correctness proof (non-rigorous correctness argument)	Mostly	Yes
Complete rigorous correctness proof	Mostly (but for higher grades only)	Yes (but for higher grades only)

[†] Specific to the problem, not just describing a general proof method like "induction" or "exchange argument".

Grading As decribed in the detailed grading criteria on Canvas, the requirements for the overall grade on this mastery test are then as follows.

Grade	Requirement
A	Solve all four problems, at least three with rigorous correctness proofs.
B	Solve three problems all with rigorous correctness proofs, or solve all four problems with at least two rigorous correctness proofs.
C	Solve three problems with at least one rigorous correctness proof.
D	Solve two problems with at least one rigorous correctness proof, or three problems with only non-rigorous correctness arguments.
E	Solve two problems with only non-rigorous correctness arguments.
Fx	Miss one of the requirements for one of the problems for grade E.

Problems

1 Ad Nauseam

A factory has n machines. These machines are old and unreliable and frequently have to be restarted, a task which falls on you, the Machine Operator. Restarting a machine is relatively time-consuming, and takes a total of r minutes of your time (the same for each machine). After years of working with the machines, you know them like the palm of your hand: for the i'th machine you know that after restarting it, it will function for exactly f_i minutes before it has to be restarted again.

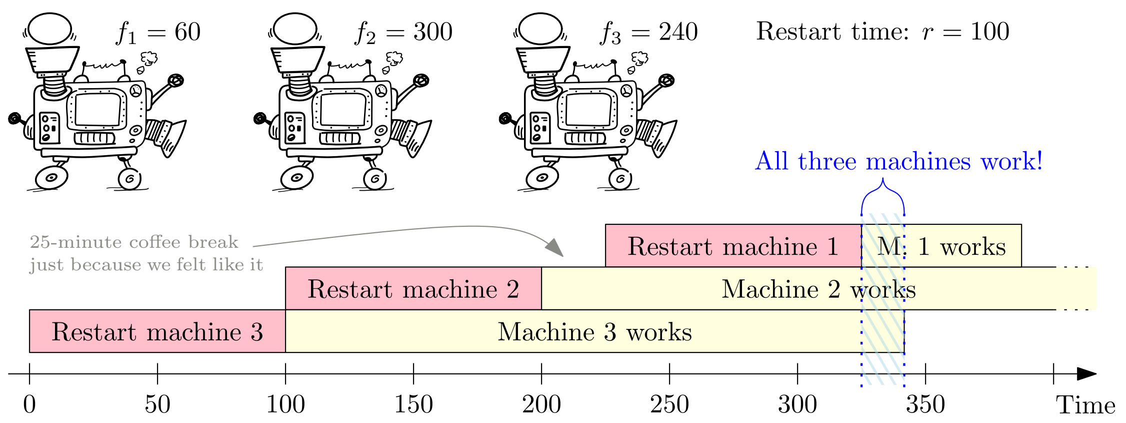
When you arrive at work, all machines are currently inoperable and need to be restarted. It is company policy that the Machine Operator (i.e., you) cannot leave work if there are any machines that have stopped functioning and need to be restarted. You would really like to be able to leave work at some point. Company policy further dictates that all employees must be focused on the task at hand, which means that you are not allowed to work on restarting several machines in parallel: once you begin restarting a machine you must spend the entire r minutes completely restarting that machine before you are allowed to switch to some other task.

Design an algorithm which determines whether you can get out of this Sisyphean task (i.e., if you can get all n machines functioning at the same time) or whether you will be stuck at work indefinitely. The algorithm must have a time complexity of $O(n^{1.5})$, assuming a unit cost analysis (e.g., you may assume that adding/multiplying/comparing two numbers takes $O(1)$ time). The input to the algorithm consists of:

- The integers n and r as described above
- For each $1 \leq i \leq n$, the integer $f_i \geq 1$ as described above.

(See example on next page.)

Example: here is an example of $n = 3$ machines and one possible strategy for restarting them.



(If r was e.g. 200, then it would be impossible to get all three machines working simultaneously.)

2 Caveat Venditor

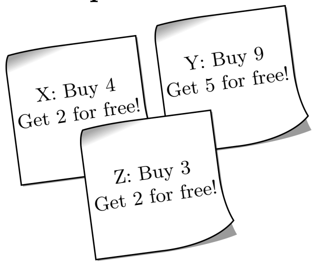
Due to dwindling popularity, the world's last DVD store is closing down and having a sale to get rid of their remaining inventory of movies. Every movie costs the same amount C crowns, and in addition they have distributed a large number of coupons all over town with offers along the lines of "Buy any five movies, get two of them for free". This means that you can get any 5 movies in the store (even 5 copies of the same movie) for a cost of $3 \cdot C$ crowns (instead of $5 \cdot C$ crowns).

An avid DVD collector wants to buy n of the movies available in the store, and to get the best deal they have gone around town and collected a large number m of the discount coupons (for some reason, nobody else was interested in them). The coupons can have different offerings. E.g., one could be "buy three, get one for free", another could be "buy 100, get 5 for free", etc. Now the collector has a conundrum: what is the best way to use these coupons in order to get all n movies as cheaply as possible? It is fine to buy a few additional movies if that leads to a smaller total cost (see example below). Each coupon can only be used once.

Design an algorithm which given the set of available coupons, finds the smallest total cost of buying the n movies. The algorithm must have time complexity polynomial in n and m, assuming a unit cost analysis (e.g., you may assume that adding/multiplying/comparing two numbers takes $O(1)$ time). The input to the algorithm consists of:

- The integers n, m, and C as described above.
- For each $1 \leq i \leq m$, a pair of integers (b_i, f_i) (where $1 \leq f_i \leq b_i \leq n$), indicating that the i'th coupon says "buy b_i, get f_i for free".

Example:



$m = 3$ available coupons X, Y, Z

Each movie costs $C = 99$ kr

Some ways of buying these $n = 6$ movies:

- No coupon: 594 kr
- Using only coupon X: 396 kr
- Using only coupon Y: 396 kr
  (this way we also get 3 additional movies, but we don't care about that)
- Using only coupon Z: 396 kr
- **Using coupons X and Z:** 297 **kr**
  (we get one additional movie that we don't care about)

Smallest cost = 297 kr

3 Coniunctis Viribus

A train company has a sequence of n train carriages arranged in a line along the rail at their railway depot. They would like to combine these carriages into some number of separate trains, without changing the order of the carriages. For instance, maybe the first four carriages are combined into one train, the next seven carriages into another train, and so on.

Just like any complex system, the maintenance needed for a train is more than the sum of its parts. Specifically, a train consisting of k carriages needs a total of k^2 units of maintenance. In order to cover the maintenance need, each carriage i has some maintenance capacity x_i. For a train to function properly, its total maintenance capacity must be at least as large as its maintenance need. In other words, if we choose to form a train consisting of carriages $i, i+1, i+2, \ldots, j-1, j$ for some $i \leq j$, then it must hold that $x_i + x_{i+1} + \ldots + x_j \geq (j-i+1)^2$. Given this requirement, the train company would like to form as few trains as possible using all the n carriages (since having more trains leads to more administrative overhead, but that is the topic for another problem).

Design an algorithm which determines the fewest number of trains that can be formed using all the n train carriages. The algorithm must have time complexity polynomial in n, assuming a unit cost analysis (e.g., you may assume that adding/multiplying/comparing two numbers takes $O(1)$ time). The input to the algorithm consists of:

- The integer n.
- For each $1 \leq i \leq n$, the integer $x_i \geq 1$ indicating the maintenance capacity of the i'th carriage.

Example: suppose $n = 5$ and $x = (4, 1, 6, 1, 2)$. Then we can form two trains: one train using the first two carriages (need $= 2^2 = 4$, capacity $= 4 + 1 = 5$), and one train using the last three carriages (need $= 3^2 = 9$, capacity $= 6 + 1 + 2 = 9$). This is the best possible for this input; we cannot form a single train (since that would have a maintenance need of $5^2 = 25$ while only having a maintenance capacity of $4 + 1 + 6 + 1 + 2 = 14$).

4 Quid Pro Quo

You are playing an open-world computer game where your character trades resources with other characters in order to achieve your life-long goal of having a complete collection of resources.

In the game there are n resource types (wood, gold, bananas, paper clips, transistors, etc) numbered from 1 to n, and m other characters (apart from your own character) numbered from 1 to m. Character j initially owns $r_{i,j}$ units of resource i, and their goal is to gather $w_{i,j}$ units of resource i.

Whenever you have some resource i that another character wants, you can trade that for any surplus resource i' that the character is willing to part with (if they have more of the resource than they want to have). These exchanges always happen at a 1-to-1 ratio, for example 1 unit of gold would be traded for 1 paper clip.

You would like to trade resources with the various characters in such a way that you own a complete collection, i.e., at least one of each resource type. You can trade with each character as many times and whenever you want (for example you could first trade with character 1, then do some other trades, and then come back to character 1 to do some more trades with them). Character j will accept trading one unit of resource i for one unit of resource i' if and only if they currently have less of resource i than they want to have, and currently have more of resource i' than they want to have. The other characters are NPCs and do not interact with each other, only with you.

Design an algorithm that determines whether or not it is possible for you to accumulate at least one unit of each type of resource. The algorithm must run in polynomial time in n and m, assuming a unit cost analysis (e.g., you may assume that adding/multiplying/comparing two numbers takes $O(1)$ time). The input to the algorithm consists of:

- The integers n and m
- For each resource $1 \leq i \leq n$, an integer $h_i \geq 0$ indicating how many units of resource i you initially have.
- For each resource $1 \leq i \leq n$ and character $1 \leq j \leq m$:
  - An integer $r_{i,j} \geq 0$, the number of units of resource i that character j initially has.
  - An integer $w_{i,j} \geq 0$, the total number of units of resource i that character j would like to have.

Example: suppose there are $n = 4$ resources (wood, gold, bananas, paper clips), and $m = 3$ characters. You initially have 15 wood, 1 gold, and 1 banana (so you are only missing a paper clip). One character has 1 wood but want a total of 3, and also has 1 banana. Another character wants 20 bananas and has 10 gold. The last character has a paper clip but is only wants a single banana. In other words, we have

$$h = (15, 1, 1, 0) \qquad r = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 10 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad w = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 20 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

A possible solution here is that you first trade one unit of wood with the first character for a banana. Then you have two bananas and can trade one of them with the last character for a paper clip.