Solutions to Assignment 2

1. You are given a sequence of n songs where the i^{th} song is ℓ_i minutes long. You want to place all of the songs on an ordered series of CDs (e.g. $CD_1, CD_2, \ldots CD_k$) where each CD can hold m minutes. Furthermore,

   (a) The songs must be recorded in the given order, song 1, song 2, ..., song n.

   (b) All songs must be included.

   (c) No song may be split across CDs.

   Your goal is to determine how to place them on the CDs as to minimise the number of CDs needed. Give the most efficient algorithm you can to find an optimal solution for this problem, prove the algorithm is correct and analyse its time complexity.

   Solution Simply put as many songs as they can fit on CD1, and continue in this manner with other songs and CDs. To prove optimality assume that there exists a better solution which uses fewer CDs. Clearly, in such a solution there must exist a CD which violates the greedy approach we described, and let CD_p be the first CD which violates our greedy strategy. On such CD_p let the last song put on it be song(m), then, because our greedy strategy is violated, song(m+1) could have fit on CD_p but was instead put on CD_{p+1}. But now we can move song(m+1) from CD_{p+1} to CD_p and still have all the conditions satisfied. In this way we obtain a solution in which our greedy strategy was respected on CDs 1 to p. Continuing in this manner we can transform the allegedly better solution to our greedy solution without increasing the number of CDs used, which is a contradiction. Note that we just have to keep adding durations of songs until we reach the capacity. This is takes $O(n)$ time.

2. At a trade school, there are N workers looking for jobs, each with a skill level x_i. There are P entry-level job openings, and the i^{th} opening only accepts workers with a skill level less than or equal to p_i. There are also Q senior job openings, the i^{th} of which requires a skill level of at least q_i. Each worker can take at most one job, and each job opening only accepts a single worker. Your task is to determine the largest number of workers you can assign to jobs in time $O(N \log N + P \log P + Q \log Q)$.

   Solution This can be done in several ways with a straightforward greedy method. To start we sort in a non-decreasing order the workers by their skill level x_i; we also sort in an increasing order the junior jobs by their upper bound skill level requirement p_i; we also sort in an increasing order the more

senior jobs by their lower bound skill requirement q_i. This clearly can be all done in time $O(N \log N + P \log P + Q \log Q)$. By re-labelling if necessary, we can assume that $x_{i+1} \geq x_i$ for all $i < N$. Now start with the least qualified worker x_1 and assign him to the job with the lowest p_m such that $p_m \geq x_1$. We continue in this manner for as long as possible. Among the leftover workers, we then start with the most qualified worker j and assign him to job with the largest requirement q_m such that $x_j \geq q_m$. We continue in this manner until there are no more jobs and workers which are compatible. It is easy to see that every q_j and every p_i needs to be considered only once, because the assignments of P jobs are a monotonically increasing sequence and assignments of Q jobs a monotonically decreasing sequence. Thus, this step takes $O(P + Q)$ many steps, so the total complexity of the algorithm is $O(N \log N + P \log P + Q \log Q)$.

3. A city is attacked by N monsters and is defended by a single hero with initial strength of S units. To kill a monster i the hero must dissipate a_i units of strength; if monster i is killed successfully the hero gains g_i units of strength. Thus, if the hero had strength $c \geq a_i$ before tackling monster i he can kill the monster and he will end up with $c - a_i + g_i$ units of strength. Note that for some monsters i you might have $g_i \geq a_i$ but for some other j you might have $a_j > g_j$. You are given S and for each i you are given a_i and g_i. Design an algorithm which determines in which order the hero can fight the monsters if he is to kill them all (if there is such an ordering). In case there is no such ordering the algorithm should output "no such ordering". Assume all values are positive integers.

   Solution While there are monsters alive, consider all monsters which satisfy $a_i \leq S$. Among those choose the one such that $g_i - a_i$ is the largest and kill it first. Let now $S \leftarrow S + g_i - s_i$ and repeat. To estimate the run time of our algorithm we can order the monsters in a list of decreasing values of $g_i - s_i$. This takes $O(n \log n)$ time. Traverse such a list, looking for the first monster j such that $S \geq s_j$. Each traversal takes $O(n)$ time and there are $O(n)$ such traversals (one per monster). Thus the whole algorithm runs in time $O(n^2)$.

4. You are given n stacks of blocks. The i^{th} stack contains $h_i > 0$ identical blocks. You are also able to move for any $i \leq n - 1$ any number of blocks from stack i to stack $i + 1$. Design an algorithm to find out in $O(n)$ time whether it is possible to make the sizes of stacks strictly increasing. (For example, 1,2,3,4 are strictly increasing but 1,2,2,3 are not). The input for your algorithm is an array A of length n such that $A[i] = h_i$. Note that you are not asked to actually move the blocks, only to determine if such movements exists or not.

Solution Let the total number of blocks be B. Just note that you can make such an increasing sequence just in case you can make sequence $1, 2, 3, n-1$ plus all the remaining $B-(1+2+\ldots+n-1)$ blocks at the last, n^{th} stack. This is because you can start from the left and transfer all additional blocks all the way to the right. But you can obtain the sequence $1, 2, 3, \ldots n-1, B-(1+2+\ldots+n-1)$ just in case $h_1 \geq 1$, $h_1+h_2 \geq 1+2$, $\ldots$ $h_1+\ldots+h_i \geq 1+2+\ldots+i$, $\ldots$ $h_1+\ldots+h_n \geq 1+2+3+\ldots+n$. Note that you are only asked whether such arrangement is possible, not to make such an arrangement. So all you need is to verify that h_i's satisfy these n inequalities. Note also that the i^{th} inequality reduces to $h_1+\ldots+h_i \geq i(i+1)/2$. All the sums $S_i = h_1+\ldots+h_i$ can be computed in linear time recursively via $S_1 = h_1$ and $S_{i+1} = S_i + h_{i+1}$. Thus all the inequalities can be verified in time $O(n)$. (Note: here we assumed that the value of S_{i+1} is obtained in a single step addition of S_i and h_i; if $S_n = O(n)$ then the additions are not $O(1)$ but $O(\log n)$=O(the number of bits) and the algorithm becomes $O(n \log n)$.)

5. You are given n jobs where each job takes one unit of time to finish. Job i will provide a profit of g_i dollars ($g_i > 0$) if finished at or before time t_i, where t_i is an arbitrary integer larger or equal to 1. Only one job can be done at any instant of time and jobs have to be done continuously from start to finish. Note: If a job i is not finished by t_i then there is no benefit in scheduling it at all. All jobs can start as early as time 0. Give the most efficient algorithm you can to find a schedule that maximises the total profit.

   Solution This is a very important problem solved using a strategy that can be called "just in time". We first sort the jobs according to profits g_i. We then schedule the highest profit job so that if finishes just on time t_i. We then proceed down the list of jobs, scheduling each job so that it ends either at its deadline, or, if this slot is already occupied, at the available slot closest to its deadline (if there are any such left, otherwise we do not schedule it at all). There are n jobs to schedule and we might need to backtrack $O(n)$ many slots to find one closest to the deadline of that job, so the algorithm runs in time $O(n^2)$.