# CS 577: Introduction to Algorithms — Homework 4 Solutions

## Problem 1

We present a greedy algorithm that returns a feasible set of deliveries $S$ that maximizes the profit. By the provided lemma, completing the deliveries in $S$ in order of their due date gives us a schedule for deliveries in $S$ that meets all deadlines. The algorithm builds a feasible set $S$ by including deliveries in decreasing order of points while maintaining the invariant that the set $S$ is feasible.

**Algorithm 1:** Pseudocode of greedy

**Input:** number of deliveries $n$; each delivery's deadline $t_i$ and payment amounts $p_i$.
Sort the deliveries by decreasing order of payment amount;
Let $L$ represent this sorted list;
Initialize set $S = \emptyset$;
**foreach** $d \in L$ **do**
    **if** $S \cup \{d\}$ *is feasible* **then**
        add $d$ to $S$;
**return** $S$.

To test the "If" condition, we maintain an array $A$ that keeps track of number of deliveries in $S$ due on or before day $t$ for all $t$. That is, $A[t]$ keeps track of number of deliveries in $S$ due on or before day $t$. When a delivery with due date $f$ is added to $S$, add 1 to $A[t]$ for all $t \geq f$.

### Proof of Correctness:

By construction the set $S$ that the algorithm outputs is feasible. We will show that $S$ also attains maximum possible profit or in other words that $S$ is optimal, via an exchange argument.

Let the deliveries in $S$ listed in decreasing order of payment amount be $\{d_1, d_2 \dots d_k\}$. Suppose that there is an optimal solution $M$ that is different from $S$ and that $j$ is the smallest index such that $d_j$ is not in $M$. That is, assume there is an optimal solution $M = \{d_1, d_2, \dots d_{j-1}, g_j, \dots g_{k'}\}$, where $g_j, \dots, g_{k'}$ are different from $d_j$. Note that since $M$ is feasible, so is its subset $\{d_1, \dots, d_{j-1}, g_j\}$. Now we observe that $g_j$ cannot have larger value than $d_j$, otherwise our algorithm would have considered it before $d_j$ and added it to our solution. Thus $g_j$ has value less than $d_j$ and in fact all the other $g$'s have smaller payment amount than $d_j$.

We will now carry out an exchange argument. Specifically, we will exchange $d_j$ with some other delivery $g$ in $M$ such that the resulting set $M \cup \{d\} \setminus \{g\}$ is both feasible and has more profit than $M$. We first consider the set $M' = M \cup \{d_j\}$. This set may not be feasible, however since $M$ was feasible, it must hold that for any day $t$, the number of deliveries in $M'$ due on or before day $t$ is $\leq t+1$. Let $t_0$ be the smallest $t$ for which the number of deliveries due on or before day $t$ is exactly $t+1$. Note that if we delete a delivery from $M'$ due on or before $t_0$, then the resultant set is feasible. The key observation here is that there must exist a delivery $g$, of value smaller than $d_j$, that is due before $t_0$. This is because, if the only deliveries due on or before $t_0$ in $M'$ are the $d_i$'s, then the number of deliveries due on or before $t_0$ must be no more than $t_0$ (because $\{d_1, \dots d_j\}$ form a feasible set). Therefore deleting the delivery $g$ from $M'$ results in a feasible set of total value greater than $M$. This is a contradiction, and thus $S$ is an optimal set.

The argument in the prior case shows that we can start with an optimal set $M$ that contains the $j-1$ deliveries $d_1, d_2, \dots d_{j-1}$, but not $d_j$, and transform it into a feasible set of value equal to or greater than $M$. Repeating this gives us an optimal solution $T$ that contains $S$. Since $T$ must be feasible, and adding any delivery to $S$ gives a set that is not feasible, we have $T = S$. This completes the proof.

### Running time analysis:

Sorting the list with respect to payment amount takes $O(n \log n)$ time. To check whether the set $S$ is feasible at any given stage, we keep track of an array of length $n$ in which the $i$-th entry stores the number of deliveries in $S$ due on or before day $i$. Updating this array every time a delivery is added to $S$ and checking whether $S$ remains feasible requires $O(n)$ time. Since we need to do this $n$ times, the entire algorithm requires $O(n^2)$ time. Finally sorting $S$ with respect to due date requires $O(n \log n)$ time. Therefore the algorithm requires $O(n^2)$ time.

## Problem 2

(a) The valid orderings that work for the following matrix is 1, 3, 4, 2 or 1, 4, 3, 2 or 3, 1, 4, 2.

(b) The greedy rule for determining which of the $n$ switches should be pressed last: pick any switch that contains only 0s or 1s and press it last.

(c) Since all doors can be opened by pressing switches in some order, there has to be a switch that only contains 0 and 1. If not, then no matter which switch we press at last, there will be at least one door that is closed. This is a contradiction because we know that there exists an ordering of switches that can open all doors.

Now to prove the correctness of the algorithm, we do an exchange argument. Let $G$ be the switched pressed last by the greedy algorithm, and let $O = (O_1, O_2, \cdots, O_n)$ be an optimal ordering of the switches with $O_n \neq G$. Since $G$ appears somewhere in the optimal ordering, suppose that $O_i = G$. We will now show that the ordering $O' = (O_1, \cdots, O_{i-1}, O_{i+1}, \cdots, O_n, G)$ will also open all doors. That is, it is safe to remove switch $G$ from the optimal ordering and move it to the end. Note that $G$'s row in the matrix only contains 0s and 1s. For a door $j$ that is undistured by $G$ (i.e.the $j$th entry in $G$'s row is 0), $G$'s position in the ordering does not effect its final state. Since $O$ opened the door $j$, so does $O'$. For a door $j$ that is opened by $G$ (i.e.the $j$th entry in $G$'s row is 1), it is opened in $O'$ no matter what its state was at the end of the previous $n-1$ switches. Therefore, $O'$ opens all doors and our proof is complete.

(d) Here $S$ denotes the set of switches left to consider when some partial suffix of the ordering has been determined.

**Algorithm 2:** Order($M$): return a reverse ordering over $1, \cdots, n$ that opens all doors given matrix $M$.

Initialize $S = \{1, \cdots, n\}$.
Initialize $D = \{1, \cdots, m\}$
**while** $S \neq \emptyset$ **do**
    For every $i \in S$, check if $(i.j) = -1$ for $j \in D$.
    Let $i \in S$ be any switch that only contains 0 or 1 at position $j$ where $j \in D$.
    Append $i$ to the end of the ordering.
    Set $S := S \setminus \{i\}$.
    Let $J$ be the set of doors s.t. $(i, j) = 1$.
    Set $D := D \setminus J$.
Return ordering.

(e) The asymptotic runtime of your algorithm is $O(n^2 m)$, because we go through the while loop $n$ times, and every time we do $O(nm)$ work.