# H6.SOL

## PROBLEM 1 *Ski racing*

You run the Giant slalom course at Loon on weekends. There must always be a ski patrol on duty in case there are injuries during training. There are $n$ patrol certified instructors who can work, but they are busy college students with complex social obligations; instructor $i$ can work starting at time $a_i$ and ending at time $b_i$ on Saturdays. Your goal is to cover the entire day from 8a–8p *using the fewest instructors as possible*. You can assume that there are enough instructors to cover the entire day; the only problem is to find the smallest number of them to do it.

1. The first idea that comes to mind is to start with an empty schedule, and then add the instructor who covers *the most amount of time that is not already covered*. Show that this algorithm fails by providing a counter-example.

   **Solution:** Consider 3 available guards, $(8, 14), (9, 19), (14, 20)$. The strategy above would pick guard 2 first, and then also need guards 1 and 3, requiring a total of 3 guards. However, guards 1 and 3 suffice for an optimal solution.

2. State and prove an exchange style lemma that you can use to construct a greedy algorithm.

   **Solution:** Exchange property: the optimal solution for a lifeguard problem between the hours of $(a, b)$ includes the guard who begins at or before time $a$ and guards for the longest period of time.

   To see this exchange, consider any other optimal solution. Let $g$ be first guard in that solution, and let $(g_s, g_e)$ be that guard's starting and ending times respectively. Let $g_a^*$ be the guard that begins at or before time $a$ and covers the maximum time after $a$. Note that replacing $g$ by $g_a^*$ yields another valid solution and has the same number of guards as the optimal, and must therefore also be optimal.

3. Present pseudo-code and running time analysis for an algorithm that uses your lemma from above to output the smallest number of instructors needed.

   **Solution:** Set $t$ =8a. Pick the guard who begins at or before $t$ and covers the maximum time after $t$. Let that end time be $t_{new}$. Set $t = t_{new}$ and repeat until the $t \geq 8p$.

## PROBLEM 2 *Latte love*

Every morning, customers $1, \ldots, n$ show up to get their drink. Suppose the omniscient barista knows all of the customers, and knows their orders, $o_1, \ldots, o_n$. Some customers are nicer people; let $T_i$ be the barista's expected tip from customer

$i$. Some drinks like a simple double-shot, are easy and fast, while some other orders, like a soy-milk latte take longer. Let $t_i$ be the time it takes to make drink $o_i$. Assume the barista can make the drinks in any order that she wishes, and that she makes drinks back-to-back (without any breaks) until all orders are done. Based on the order that she chooses to complete all drinks, let $D_i$ be the time that the barista finishes order $i$. Devise an algorithm that helps the barista pick a schedule that minimizes the quantity

$$\sum_{i}^{n} T_i D_i$$

In other words, she wants to serve the high-tippers the fastest but she also wants to take into consideration the time it takes to make each drink. (Hint: think about a property that is true about an optimal solution.)

1. State and prove an exchange lemma that is true about this problem and can be used for a greedy algorithm.

   **Solution:**

   **Lemma 1** *Let $R_i = \frac{T_i}{t_i}$ be the ratio of tip to time it takes to make the drink for each order $o_i$. For any set of orders $O$, there exists an optimal solution in which the barista makes the drinks in descending order according to $R_i$ (i.e., largest $R_i$ first).*

   *Proof.* Let $j, i$ be consecutive orders in a schedule in which drink $j$ is made first. If $R_i > R_j$, then swapping the schedule and making order $i$ first results in lower cost schedule. This follows because

   $$\frac{T_i}{t_i} > \frac{T_j}{t_j}$$

   implies that $T_i t_j > T_j t_i$. Adding $T_i t_i + T_j t_j$ to both sides and factoring implies that

   $$T_i(t_i + t_j) + T_j t_j > T_i t_i + T_j(t_i + t_j)$$

   This shows that the total cost in any schedule in which order $j$ is made first will be higher than if order $i$ is made first.

   Applying this fact repeatedly implies that the minimum cost schedule begins with the order with largest ratio $R_i$. It also implies that the second drink made should have the second highest $R_j$, etc. □

2. Provide pseudo-code and run time analysis for your algorithm.

   **Solution:** This lemma suggests a simple greedy strategy. Pick the order with the highest ratio first, remove from the set, and solve the smaller problem. This can be quickly implemented by computing $R_i$ and sorting.

PROBLEM 3 *5800 is tough*

This class is hard and so are all of your others. You are given $n$ assignments $a_1, \ldots, a_n$ in your courses. Each assignment $a_i = (d_i, t_i)$ has a deadline $d_i$ when it is due and an estimated amount of time it will take to complete, $t_i$. You would like to get the most out of your education, and so you plan to finish all of your assignments. Let us assume that when you work on one assignment, you give it your full attention and do not work on any other assignment.

In some cases, your outrageous professors demand too much of you. It may not be possible to finish all of your assignments on time given the deadlines $d_1, \ldots, d_n$; indeed, some assignments may have to be turned in late. Your goal as a sincere student is to minimize the lateness of any assignment. If you start assignment $a_i$ at time $s_i$, you will finish at time $f_i = s_i + t_i$. The lateness value—denoted $\ell_i$—for $a_i$ is the value

$$\ell_i = \begin{cases} f_i - d_i & \text{if } f_i > d_i \\ 0 & \text{otherwise} \end{cases}$$

Your goal is to computes a schedule $O$ that specifies the order in which you complete assignments which minimizes the maximum $\ell_i$ for all assignments, i.e.

$$\min_O \max_i \ell_i$$

In other words, you do not want to turn in *any* assignment *too* late, so you minimize the lateness of the latest assignment you turn in.

1. State an exchange lemma that captures the right choice to make in this problem for a greedy algorithm.

   **Solution:** The solution is the same as the scheduling problem discussed in class: solve the earliest-deadline-first. The reason that this rule is optimal is similar but slightly different from the one presented in class. We show that for a schedule $O$, swapping any two consecutive activities that are out-of-order with respect to deadlines cannot increase the maximum lateness.

   To make this argument, we must first rule out any "idle time" in an optimal schedule. We say that $O$ has *idle* time if $O$ contains a time period during which no work is scheduled, and yet there remains assignments to be done. If $O$ is a schedule, we use the notation $O(i)$ to refer to the $i^{\text{th}}$ job scheduled by $O$.

   **Claim 1** *There exists an optimal schedule with no idle time.*

   (Remove the idle time by shifting each assignment so that they are completed consecutively. This operation can only decrease the maximum penalty, since the order of operations remains the same.)

   **Claim 2** *Let $O$ be an idle-free optimal schedule in which there is some $i$ for which $d_{O(i+1)} \leq d_{O(i)}$. In other words, in schedule $O$, job $i+1$'s deadline is before job $i$'s deadline. Then swapping job $i$ and $i+1$ results in another optimal schedule $O'$.*

*Proof.* Let $s$ be the time when job $i$ begins in $O$. Let $O'$ be identical to $O$ with the exception that jobs $O(i)$ and $O(i+1)$ are swapped. First, notice that

$$\begin{aligned}\ell_{O(i+1)} &= s + t_{O(i)} + t_{O(i+1)} - d_{O(i+1)} \\ &\geq s + t_{O(i)} - d_{O(i)} \\ &= \ell_{O(i)}\end{aligned}$$

This follows because task $t_{O(i+1)} > 0$ and deadline $d_{O(i+1)}$ is earlier (smaller) than $d_{O(i)}$. Similarly, since $t_{O(i)} > 0$, it holds that

$$\ell_{O(i+1)} \geq s + t_{O(i+1)} - d_{O(i+1)} = \ell'_{O'(i)}$$

and moreover

$$\ell_{O(i+1)} \geq s + t_{O(i+1)} + t_{O(i)} - d_{O(i)} = \ell'_{O'(i+1)}$$

which again follows because $d_{O(i+1)} < d_{O(i)}$. The lateness of all other jobs in $O'$ and $O$ are equal. Thus, because the lateness of jobs in $O'$ are no larger than those in $O$, $O'$ must also be optimal. □

By repeatedly applying claim 2 to the optimal schedule $O$, one can arrive at an optimal schedule that is identical to the earliest-deadline-first schedule.

2. Describe with pseudo-code and analyze and algorithm that uses your lemma to solve the problem.

**Solution:**

As in class, we can implement the earliest-deadline-first scheduling by sorting the tasks by deadline in time $\Theta(n \log n)$ and completing them in that order, in a back-to-back fashion.