Algorithms

Assignment Greedy Algorithms: Solutions
1. **Scheduling n jobs on one machine:** There are \( n \) jobs: \( J_1, \ldots, J_n \) and one machine. For \( 1 \leq i \leq n \), job \( J_i \) requests the service of the machine during the time interval: \([s_i, f_i)\) \((f_i > s_i \geq 0)\). For \( 1 \leq i \leq n \), the profit of serving (scheduling) job \( J_i \) is \( p_i > 0 \). At any point of time \( t \geq 0 \), the machine can serve at most one job (Mutual Exclusion). The goal is maximize the combined profit of all the scheduled jobs.

**A greedy scheme:** Order the jobs according to some greedy criterion and scan them following this order. Accept a job if it can be scheduled without violating the Mutual Exclusion constraint. Otherwise reject it. Decisions are irreversible: a rejected job cannot be scheduled later and an accepted job cannot be preempted.

(a) **The profit of a job is its length:** \( p_i = f_i - s_i \). Intuitively, maximizing the profit is equivalent to finding a set of jobs whose combined time intervals is the “best” cover of the time interval \([\min_{1 \leq i \leq n} s_i, \max_{1 \leq i \leq n} f_i)\).

- Find an instance for which ordering the jobs by their lengths (from the longest to the shortest) is not optimal.
- For any \( n \geq 4 \), find an instance for which the greedy algorithm that follows this order produces a schedule whose profit is very close to only \( \frac{1}{3} \) of the optimal profit.

**Examples:** Consider the following 4 jobs:

\[
J_1 = [0, 2) \quad J_2 = [2, 4) \quad J_3 = [4, 6) \quad J_4 = [1, 5)
\]

The optimal solution schedules \( J_1, J_2, \) and \( J_3 \) for a profit of

\[
P_{opt} = (2 - 0) + (4 - 2) + (6 - 4) = 6
\]

The greedy algorithm schedules \( J_4 \) which is the longest job and then it cannot schedule any other job. Therefore, its profit is

\[
P_{greedy} = 5 - 1 = 4
\]

which is not optimal since \( P_{greedy} < P_{opt} \).
For \( n = 4 \) and an integer \( k \geq 2 \), consider the following jobs:

\[
J_1 = (0, k) \quad J_2 = (k, 2k) \quad J_3 = (2k, 3k) \quad J_4 = (k - 1, 2k + 1)
\]

The optimal solution schedules \( J_1, J_2, \) and \( J_3 \) for a profit of

\[
P_{opt} = (k - 0) + (2k - k) + (3k - 2k) = 3k
\]

The greedy algorithm schedules \( J_4 \) which is the longest job and then it cannot schedule any other job. Therefore, its profit is

\[
P_{greedy} = (2k + 1) - (k - 1) = k + 2
\]

It follows that the ratio between the profits is

\[
\frac{P_{greedy}}{P_{opt}} = \frac{k + 2}{3k} = \frac{1}{3} + \frac{2}{3k}
\]

When \( k \) is very large, this ratio approaches \( 1/3 \).

For \( n > 4 \), assume also that \( k \geq n - 2 \) and add the following \( n - 4 \) jobs to jobs \( J_1, J_2, J_3, J_4 \):

\[
J_5 = [k + 1, k + 2) \quad J_6 = [k + 2, k + 3) \quad \ldots \quad J_n = [k + n - 4, k + n - 3)
\]

Note that the time intervals of all the new jobs intersect the time intervals of both \( J_2 \) and \( J_4 \). The greedy algorithm cannot schedule any of the new jobs after selecting the longest job \( J_4 \) while the optimal solution would prefer \( J_2 \) which is a more profitable then the combined profit of all of the new \( n - 4 \) jobs. As a result the ratio remains \( \frac{1}{3} + \frac{2}{3k} \).
(b) **The activity selection problem**: The profit of each job is 1. In this variant, maximizing the profit is equivalent to finding the largest set of jobs that can be scheduled without violating the Mutual Exclusion constraint. Note, that ordering the jobs by a non-decreasing order of the finish times produces an optimal schedule.

(i) **Ordering by the starting times**:
- Find an instance for which ordering the jobs by a non-decreasing order of their starting times is not optimal.
- For any \( n \geq 3 \), find an instance for which the greedy algorithm that follows this order produces a schedule whose profit is \( 1/(n - 1) \) of the optimal profit.

**Examples**: Consider the following 3 jobs:

\[
J_1 = [0, 4] \quad J_2 = [1, 2] \quad J_3 = [2, 3]
\]

The optimal solution schedules \( J_2 \) and \( J_3 \) while the greedy algorithm schedules only \( J_1 \) whose starting time is the earliest. Therefore, the greedy algorithm is not optimal.

For \( n \geq 3 \), consider the following \( n \) jobs:

\[
J_1 = [0, n + 1] \quad J_2 = [1, 2] \quad J_3 = [2, 3] \quad \ldots \quad J_n = [n - 1, n]
\]

The optimal solution schedules \( J_2, \ldots, J_n \) while the greedy algorithm schedules only \( J_1 \) whose starting time is the earliest. Therefore, its profit is 1 while the optimal profit is \( n - 1 \). Hence, the ratio between the profits is \( 1/(n - 1) \).
(ii) Ordering by lengths:

- Find an instance for which ordering the jobs by their lengths from the shortest to the longest is not optimal.
- For any $n \geq 3$, find an instance for which the greedy algorithm that follows this order produces a schedule whose profit is half of the optimal profit.

Examples: Consider the following 3 jobs:

\[ J_1 = [0, 3] \quad J_2 = [3, 6] \quad J_3 = [2, 4] \]

The optimal solution schedules $J_1$ and $J_2$ while the greedy algorithm schedules only $J_3$ which is the shortest job. Therefore, the greedy algorithm is not optimal.

For $n > 3$, add the following $n - 3$ identical jobs $J_4 = J_5 = \cdots = J_n = [1, 5]$ to jobs $J_1, J_2, J_3$.

The greedy algorithm cannot schedule any other job after scheduling the shortest job $J_3$ and therefore its profit remains 1. The optimal solution schedules $J_1$ and $J_2$ for a profit of 2. Hence, the ratio between the profits remains $1/2$. 
(iii) **Ordering by degrees:** The degree of a job \( J_i \) is the number of jobs whose time intervals intersect with the time interval of \( J_i \). Find an instance for which ordering the jobs by their degrees from the smallest to the largest is not optimal.

**Example:** Consider the following 11 jobs:

\[
\begin{align*}
J_6 &= J_7 = J_8 = [1, 4] & J_9 &= J_{10} = J_{11} &= [9, 12]
\end{align*}
\]

The optimal solution schedules \( J_2, J_3, J_4, J_5 \) for a profit of 4. Since the time interval of \( J_1 \) intersects only the time intervals of \( J_3 \) and \( J_4 \) its degree is 2 while the degree of any other job is at least 3. Therefore, the greedy algorithm schedules \( J_1 \) first. Then, it can schedule at most one job out of \( J_2, J_6, J_7, J_8 \) because their time intervals intersect each other and at most one job out of \( J_5, J_9, J_{10}, J_{11} \) because their time intervals intersect each other. Therefore, the profit of this greedy algorithm is 3 which is less than the profit of the optimal solution.

**Remark:** Observe that the dynamic version of this greedy strategy also fails since after scheduling \( J_1 \), any algorithm can schedule at most two more jobs.
(c) **Arbitrary profits:** The profit of any job is arbitrary. In particular, shorter jobs may be more profitable than longer jobs.

(i) **Ordering by profits:**
- Find an instance for which ordering the jobs by their profits from the most profitable to the least profitable is not optimal.
- For any \( n \geq 4 \), find an instance for which the greedy algorithm that follows this order produces a schedule whose profit is very close to only \( 1/(n-1) \) of the optimal profit.

**Examples:** Consider the following 4 jobs:

\[
J_1 = [0, 5) \quad J_2 = [1, 2) \quad J_3 = [2, 3) \quad J_4 = [3, 4)
\]

in which \( p_1 = 2 \) and \( p_2 = p_3 = p_4 = 1 \).

The optimal solution schedules \( J_2, J_3, J_4 \) for a profit of 3. The greedy algorithm first schedules \( J_1 \) that has the largest profit of 2 and then it cannot schedule any other job. Therefore, the greedy algorithm is not optimal.

For \( n \geq 4 \), consider the following \( n \) jobs:

\[
J_1 = [0, n+1) \quad J_2 = [1, 2) \quad J_3 = [2, 3) \quad \ldots \quad J_n = [n-1, n)
\]

in which \( p_1 = P + 1 \) and \( p_2 = p_3 = \cdots = p_n = P \) for some integer \( P \geq 1 \).

The optimal solution schedules \( J_2, J_3, \ldots, J_n \) for a profit of \( P_{opt} = (n-1)P \). The greedy algorithm first schedules \( J_1 \) that has the largest profit of \( P + 1 \) and then it cannot schedule any other jobs. Therefore, \( P_{greedy} = P + 1 \).

It follows that the ratio between the profits is

\[
\frac{P_{greedy}}{P_{opt}} = \frac{P + 1}{(n-1)P} = \frac{1}{n - 1} + \frac{1}{(n-1)P}
\]

When \( P \) is very large, this ratio approaches \( 1/(n-1) \).
(ii) Ordering by profits and lengths:

- Find an instance for which ordering the jobs by their profit over length ratios from the largest ratio to the smallest ratio is not optimal.
- For any $n \geq 2$, find an instance for which the greedy algorithm that follows this order produces a schedule whose profit could be only a very small fraction of the optimal profit.

**Examples:** For $1 \leq i \leq n$, denote by $\rho_i = \frac{p_i}{f_i - s_i}$ the ratio of profit over length of job $J_i$. For an integer $p \geq 3$, consider the following 2 jobs:

$$J_1 = [0, p) \quad J_2 = [1, 2)$$

in which $p_1 = p$ and $p_2 = 2$ and therefore $\rho_1 = 1$ and $\rho_2 = 2$.

The optimal solution schedules $J_1$ for a profit of $p$. The greedy algorithm schedules $J_2$ that has the largest ratio of 2 and cannot schedule $J_1$. Therefore, its profit is only 2. Since $p > 2$, it follows that the greedy algorithm is not optimal.

For $n > 2$, add the following $n - 2$ jobs

$$J_3 = J_4 = \cdots = J_n = [1, 2)$$

in which $p_3 = p_4 = \cdots = p_n = 1$ and therefore $\rho_3 = \rho_4 = \cdots = \rho_n = 1$.

Both the optimal solution and the greedy algorithm do not schedule any of these $n - 2$ jobs. Therefore, the ratio between the greedy algorithm and the optimal solution remains the same.

It follows that the ratio between the profits is,

$$\frac{P_{\text{greedy}}}{P_{\text{opt}}} = \frac{2}{p}$$

When $p$ is very large, this ratio is very small.
2. **Scheduling all the $n$ jobs on several machine:** There are $n$ jobs: $J_1, \ldots, J_n$ and many machines. For $1 \leq i \leq n$, job $J_i$ must be served on one of the machines during the time interval: $(s_i, f_i) \ (f_i > s_i \geq 0)$. At any point of time $t \geq 0$, a machine can serve at most one job (*Mutual Exclusion*). The goal is to schedule all the jobs on as few as possible machines.

**A greedy scheme:** Order the jobs according to some greedy criterion and scan them following this order. The current considered job is scheduled on the first possible machine (**first-fit-greedy**). When none of the existing machines can serve this job without violating the mutual exclusion constraint, a new machine is added to serve this job.

(i) Show an instance for which ordering the jobs by a non-decreasing order of their finish times does not produce an optimal solution.

**Example:** Consider the following 4 jobs:

$$J_1 = [0, 2), J_2 = [1, 4), J_3 = [5, 6), J_4 = [3, 7) .$$

The optimal solution needs only 2 machines, it schedules $J_1$ and $J_4$ on $M_1$ and schedules $J_2$ and $J_3$ on $M_2$.

The greedy algorithm considers the jobs in the order $J_1, J_2, J_3, J_4$. It first schedules $J_1$ on machine $M_1$ and then schedules $J_2$ on machine $M_2$. It can schedule $J_3$ on either $M_1$ or $M_2$. However, following the first-fit rule, the greedy algorithm schedules $J_3$ on $M_1$. Finally, the greedy algorithm must schedule $J_4$ on a new machine $M_3$ because the time interval of $J_4$ intersects the time intervals of both $J_2$ and $J_3$. The greedy algorithm is not optimal because the optimal solution uses only 2 machines.
(ii) Prove that ordering the jobs by a non-decreasing order of their starting times produces an optimal solution.

Proof: Let G be the greedy algorithm that applies the first-fit rule after ordering the jobs by their non-decreasing order of their starting times. Let the jobs $J_1, J_2, \ldots, J_n$ be such that $s_1 \leq s_2 \leq \cdots \leq s_n$. Note, that in the example of part (i), G produces an optimal solution.

For any time $t \geq 0$, let $B_t$ be the number of jobs for which $s_i \leq t < f_i$ and let

$$B = \max_{t \geq 0} \{B_t\}.$$ 

Observation: Let $t_{max}$ be a time in which $B = B_{t_{max}}$. Then at time $t_{max}$, any optimal schedule must use at least $B$ machines.

Proposition: $G$ schedules all the $n$ jobs on $B$ machines denoted by $M_1, M_2, \ldots, M_B$.

Proof of the proposition: By induction on the number of jobs.

- Trivially, $G$ schedules $J_1$ on $M_1$.
- For $1 \leq k < n$, assume that $G$ schedules the first $k$ jobs $J_1, J_2, \ldots, J_k$ on the machines $M_1, M_2, \ldots, M_B$ and prove that $J_{k+1}$ can be scheduled on at least one of these machines.
- Let the starting time of job $J_{k+1}$ be $t = s_{k+1}$. By definition, $B_t \leq B$. Therefore, after scheduling the first $k$ jobs, at most $B - 1$ machines have scheduled jobs at time $t$. As a result, $G$ can find an available machine among $M_1, M_2, \ldots, M_B$ to schedule job $J_{k+1}$.

QED: The above observation and proposition prove that $G$ is optimal.
You need to pack several items into your shopping bag without squashing anything. The items are to be placed one on top of the other. Each item has a weight and a strength, defined as the maximum weight that can be placed above that item without it being squashed. A packing order is **safe** if no item in the bag is squashed. That is, the strength of each item is at least the combined weight of what is placed in the bag above it.

**Notations:** There are $n$ items $X_1, X_2, \ldots, X_n$. For $1 \leq i \leq n$, the weight of $X_i$ is $w_i$ and the strength of $X_i$ is $s_i$. Denote item $X_i$ by the pair $(w_i, s_i)$.

(a) Consider the following three items:

- A loaf of bread whose weight is 5 and whose strength is 6.
- A carton of eggs whose weight is 4 and whose strength is 4.
- A bottle of milk whose weight is 12 and whose strength is 9.

Which of the six possible orderings of the bread, eggs, and milk are safe and which are unsafe.

**Solution:** The weight of the milk is larger than the strengths of both the bread and the eggs. Therefore, in a safe packing, the milk must be at the bottom. The weight of the bread is larger than the strength of the eggs. Therefore, the eggs must be packed on top of the bread.

As a result, the only possible safe packing is milk at the bottom, bread at the middle, and eggs at the top. Indeed, this packing order is safe because the strength of the milk is enough to carry without being squashed the combine weights of the bread and the eggs and the bread is strong enough to carry the eggs without being squashed.
(b) Consider packing the $n$ items in weight order, with the heaviest at the bottom. Show an example for which this strategy does not produce a safe packing order even if one exists.

**Example:** Consider the following 2 items:

$$X_1 = (3, 1) \quad X_2 = (2, 3)$$

![Diagram](non-safe)

This strategy fails because the strength $s_1 = 1$ of the heavier item $X_1$ is not enough to carry the weight $w_2 = 2$ of the lighter item $X_2$. However, packing $X_1$ above $X_2$ is safe because the strength $s_2 = 3$ of $X_2$ is enough to carry the weight $w_1 = 3$ of $X_1$.

(c) Consider packing the $n$ items in strength order, with the strongest at the bottom. Show an example for which this strategy does not produce a safe packing order even if one exists.

**Example:** Consider the following 2 items:

$$X_1 = (1, 2) \quad X_2 = (3, 1)$$

![Diagram](non-safe)

This strategy fails because the weight $w_2 = 3$ of the weaker item $X_2$ is too much for the strength $s_1 = 2$ of the stronger item $X_1$. However, packing $X_1$ above $X_2$ is safe because the strength $s_2 = 1$ of $X_1$ is enough to carry the weight $w_1 = 1$ of $X_1$. 
(d) Assume that you have a safe packing order in your bag in which $X_j$ is placed directly on top of $X_i$. Suppose also that:

$$s_i - w_j \leq s_j - w_i$$

Show that if you swap $X_i$ and $X_j$ you still have a safe packing order.

**Solution:** Assume that in the original safe packing order the combined weight of the items on top of $X_j$ is $W$. Since $X_i$ is not squashed, it follows that $s_i \geq w_j + W$. Therefore, after swapping $X_i$ with $X_j$, item $X_i$ will not be squashed since the combined weight on top of it is only $W$. Since the swap changed nothing for all the items below $X_i$ and for all the items above $X_j$, it follows that the only item that might be squashed after the swap is $X_j$.

Thus, it remains to show that after the swap $s_j \geq W + w_i$. That is, it remains to show that $X_j$ is strong enough to carry the combined weight of all the items that were above $X_j$ in the original safe packing order and the additional weight of $X_i$.

Since $X_i$ was not squashed in the original safe order, it follows that $s_i \geq W + w_j$. Equivalently

$$W \leq s_i - w_j$$

The assumption that $s_i - w_j \leq s_j - w_i$ implies that

$$W \leq s_j - w_i$$

which is equivalent to what remained to prove that $s_j \geq W + w_i$.

(e) Design a greedy algorithm that produces a safe packing order if one exists.

**Algorithm:** Let the bottom-up packing order be $X_1, X_2, \ldots, X_n$ such that

$$s_1 + w_1 \geq s_2 + w_2 \geq \cdots \geq s_n + w_n.$$ 

That is, greedily pack the items in the bag following the non-increasing order of the sum weight plus strength.

**Example:** In part (a) of this problem, the order of the items by greedy is first milk whose strength plus weight is $21 = 9 + 12$, then bread whose strength plus weight is $11 = 6 + 5$, and then eggs whose strength plus weight is $8 = 4 + 4$. Indeed, following this order would produce a safe packing.

Assume that for some $1 \leq j < i \leq n$ there exists a safe packing order in which $X_j$ is placed directly on top of $X_i$. The above order implies that

$$s_i + w_i \leq s_j + w_j$$

which is equivalent to

$$s_i - w_j \leq s_j - w_i.$$ 

By the previous part of the question, after swapping $X_i$ with $X_j$, the new order is also safe. Keep performing these swaps until there are no such pairs of items. The only order for which there are no such pairs is the sorted packing order adopted by the algorithm. Therefore, this algorithm produces a safe packing order.