**Q1.** Define a network as follows. Put a source node $s$ and a sink node $t$; put a node $u_\ell$ and add a directed edge with capacity 1 from $u_\ell$ to $t$, for every time slot $\ell \in [L]$; for every $i \in [n]$ and $j \in [\frac{L}{p_i}]$, put a node $v_{ij}$ for the $j^{th}$ instance of task $i$, add a directed edge of capacity 1 from $s$ to $v_{ij}$, and add directed edges from $v_{ij}$ to each $u_\ell$ for each $\ell$ in $j \cdot p_i + 1, \ldots, j \cdot p_i + p_i - 1$. Find a maximum flow from $s$ to $t$. If there is flow on the edge $(v_{ij}, u_\ell)$, schedule the $j^{th}$ instance of task $i$ in time slot $\ell$.

First, we prove that if $\sum_i \frac{1}{p_i} \leq 1$, a feasible schedule exists. We construct a fractional flow as follows. Set the flow of every edge $(v_{ij}, u_\ell)$ to $\frac{1}{p_i}$; set the flow of every edge $(s, v_{ij})$ equal to 1; and set the flow of every edge $(u_\ell, t)$ equal to $\sum_i \frac{L}{p_i}$. This is clearly a feasible flow of size $\sum_i \frac{L}{p_i}$. Since an integral maximum flow of the same size must exist, every task is feasibly scheduled.

To prove the necessity of $\sum_i \frac{L}{p_i} \leq 1$, observe that each task must be scheduled $\frac{L}{p_i}$ times and there are a total of $L$ time slots, so $\sum_i \frac{L}{p_i} \leq L$, dividing both sides by $L$ we get the condition.

Note that a greedy algorithm which considers tasks in increasing order of $p_i$ does not work. Consider the following example. There are 4 tasks with $p_1 = 3$, $p_2 = 4$, $p_3 = 5$, $p_4 = 6$. Such a greedy algorithm computes a schedule as follows.

| *task* | 1 | | | 1 | | | ... |
|---|---|---|---|---|---|---|---|
| *timeslot* | 1 | 2 | 3 | 4 | 5 | 6 | ... |

then

| *task* | 1 | 2 | | 1 | 2 | | ... |
|---|---|---|---|---|---|---|---|
| *timeslot* | 1 | 2 | 3 | 4 | 5 | 6 | ... |

then

| *task* | 1 | 2 | 3 | 1 | 2 | 3 | ... |
|---|---|---|---|---|---|---|---|
| *timeslot* | 1 | 2 | 3 | 4 | 5 | 6 | ... |

But then it fails to schedule the $4^{th}$ job because all the first 6 time slots are already full!

**Q2.** Label the jobs in the order they are considered by the greedy algorithm. Let $j$ be the job that finishes last in the schedule computed by the greedy algorithm. Let $s_j$ be the starting time of job $j$. All the machines must have been busy at time $s_j$ so $OPT > s_j$. On the other hand, $OPT \geq p_j$. There are two possible scenarios:

- either $p_j \leq \frac{OPT}{3}$, which implies that

$$GREEDY = s_j + p_j \leq OPT + \frac{OPT}{3} = \frac{4}{3}OPT,$$

- or $p_j > \frac{OPT}{3}$, which implies that $p_1, \ldots, p_j > \frac{OPT}{3}$. Observe that it must be $j \leq 2m$, otherwise the optimal schedule must have at least 3 jobs of size more that $\frac{OPT}{3}$ on the same machine which is a contradiction. It is easy to see that the optimal schedule for the first $j$ jobs is exactly the schedule computed by the greedy algorithm which looks like

| | | | | |
|---|---|---|---|---|
| | | ... | $m+2$ | $m+1$ |
| 1 | 2 | ... | $m-1$ | $m$ |

In other words, job $m-1$ and $m+2$ should be on the same machine, $m-2$ and $m+3$ on the same machine, and so on. Therefor the greedy can be no worse than optimal for this first $j$ jobs in this case, so $GREEDY = OPT$.

To prove that the bound is tight, we need a tight example for every $m$. Consider an instance with $2m+1$ jobs with $p_j = 2m+j-1$. The schedule computed by the greedy algorithm looks like the following.

| | | | | | |
|---|---|---|---|---|---|
| $2m$ | | ... | | | |
| $2m+1$ | $2m+2$ | ... | $3m-2$ | $3m-1$ | $3m$ |
| $4m$ | $4m-1$ | ... | $3m+3$ | $3m+2$ | $3m+1$ |

On the other hand, the optimal looks like the following.

| | | | | | |
|---|---|---|---|---|---|
| | | ... | | | $2m$ |
| $2m+3$ | $2m+4$ | ... | $3m$ | $3m+1$ | $2m+1$ |
| $4m$ | $4m-1$ | ... | $3m+3$ | $3m+2$ | $2m+2$ |

In other words, compared to the optimal, the jobs on the seconds row has been shifted to the right by two positions and the 3 smallest jobs have been moved to the right most machine. Observe that $GREEDY = 8m+1$ while $OPT = 6m+3$, therefore $\frac{GREEDY}{OPT} \approx \frac{4}{3}$ as $m$ grows large.

**Q3.** Observe that the contribution of job $j$ to the total completion time is exactly $k \cdot p_{ij}$ if it is schedules as the $k^{th}$ to the last job on machine $i$. Create a bipartite graph in which there is a vertex $v_j$ for each job $j \in [n]$ and $n$ vertices $u_i, \ldots, u_{i,n}$ for each machine $i \in [m]$ (i.e., a total of $m \cdot n$ vertices for the machines). Add an edge between each $v_j$ and each $u_{ik}$ with weight $k \cdot p_{ij}$. Compute a minimum weigh maximum matching in this graph[1]; assign job $j$ as the $k^{th}$ to the last job on machine $i$ iff $v_i$ is matched to $u_{ik}$.

[1] equivalently, we could set the weight of each edge $(v_j, u_{ik})$ to $C - k \cdot p_{ij}$ for some large constant $C$ and then compute a maximum weight matching.

**Q4.** WLOG, assume that the jobs are numbered in the order in which they appear in the optimal schedule. The proof is by contradiction. Suppose the jobs are not scheduled in the non-decreasing order of their rank in the optimal schedule; so there must be some index $j$ such that $\frac{p_j}{w_j} > \frac{p_{j+1}}{w_{j+1}}$. We show that by swapping the jobs $j$ and $j+1$, we can reduce the total weighted completion time which would contradict the optimality of the schedule. Observe that this swap does not affect the completion times of any jobs other than $j$ and $j+1$. Furthermore, the total change in the weighted completion times of jobs $j$ and $j+1$ due to this change is exactly

$$\Delta = p_{j+1} \cdot w_j - p_j \cdot w_{j+1} < 0$$

which implies the swapping $j$ and $j+1$ strictly reduces the total weighted completion time.