Wait No More **Jonathan Wenger**

1. The algorithm used for solving this problem.

I put all of the time/weight jobs into a 2D list (with each index in the list being a time/weight), and then sorted the list in increasing order of the ratio between $\frac{duration}{weight}$. I then iterated through the sorted durations and weights to calculate the minimized unweighted waiting time (i.e. using the formula $\sum_{n=1}^{n}(W_i)$), by storing the time of the n-1 jobs in a variable, and adding the new waiting time to the sum, and then updating the time variable. Intuitively, this algorithm minimizes the waiting time, as it completes the "best" jobs first ("best" meaning smallest $\frac{duration}{weight}$ ratio), and uses those while the total time elapsed is as small as it can be. This algorithm is in fact greedy, because it does not take any previous steps into account for deciding which next value to take from the durations/weights; it just takes the next smallest $\frac{duration}{weight}$ ratio that has not been used yet in the summation.

2. Prove the correctness of your algorithm: i.e, that it performs the desired minimization.

Let's call the schedule produced by this greedy algorithm S, and the schedule of the most optimal solution S*. I can "transform" S* into S, without making the summation any higher than what it previously was. S* has no "idle time" (i.e. time when no one is doing a job), because if S* had idle time, the total summation would only increase (and not be optimal). Similarly, S has no idle time, because S is always doing a job until all the jobs are complete.

Let's say there are two jobs (job i with ratio $r_i = \frac{t_i}{w_i}$, and job j with ratio $r_j = \frac{t_j}{w_j}$, where $r_j < r_i$). Solution S' has an inversion if job i is scheduled before job j. By definition, S has no inversions (as S is sorted in increasing ratio order).

Lemma 1 - All schedules with no inversions and no idle time have the same weighted waiting time.

- Proof - If two different schedules have neither inversions nor idle time, they may not have the same order of jobs, However, the only possible difference between these two schedules is the order of jobs with identical ratios are selected.
- **Lemma 1A -** Let's say there are x number of jobs that have the same ratio r. No matter what order you put these x number of jobs in, that will not affect the weighted waiting time.
  - Proof - Based on S, all jobs with a ratio less than r have already been completed, and all jobs with a ratio greater than r have not yet been completed. Let's say there are y number of jobs completed before any of the x number of jobs of ratio r are completed. To calculate the weighing time of y+x jobs, we take (summation of the y jobs)+ $\sum_{y}^{y+x}(W_i \times w_i)$. Because the ratio of all of the x jobs are equal, $\sum_{y}^{y+x}(W_i \times w_i)$ will be always equal regardless of the orderings of the x jobs. This is because $\sum_{y}^{y+x}(W_i \times w_i) = \sum_{y}^{y+x}(W_i) \times \sum_{y}^{y+x}(w_i)$. $\sum_{y}^{y+x}(w_i)$ is always equal regardless of the ordering. $W_i = \sum_{0}^{i-1}(t_i)$, so regardless of the ordering, $\sum_{y}^{y+x}(W_i)$ will also be equal regardless of ordering. Therefore, the order of these x jobs with the same ratio r will not affect the waiting time. (end of proof for Lemma 1A) ▫
- Therefore, all schedules with no inversions and no idle time have the same weighted waiting time. (end of proof for Lemma 1) ▫

Lemma 2 - There is an optimal schedule that has no inversions and no idle time.

- Proof - As stated earlier, S* has no idle time. If S* has an inversion, then there are a pair of jobs i and j such that j is scheduled immediately after i and $r_j<r_i$, as that is the definition of an inversion. After swapping i and j, we get a schedule with one less inversion, as i and j were directly next to each other, and also no new inversions will be created (let's call that solution after the swap $\overline{S}$). Then, r_j will come before r_i, and $\overline{S}$ will have one less inversion than S*.
- **Lemma 2A -** $\overline{S}$ has a minimum summation no greater than S*.
  - Proof - Within S*, assume that each of the r jobs have a weight w_r, waiting time W_r, duration t_r, and product P_r= ($W_r \times w_r$). $\overline{S}$ (as defined above) also has r jobs, and those r jobs have the values $\overline{w}_r$, $\overline{W}_r$, $\overline{t}_r$, and $\overline{P}_r$, which relate to w_r, W_r, t_r, and P_r, respectively. Going back to the two swapped jobs i and j, the starting time of job i in S is the exact same starting time as job j in $\overline{S}$. All other jobs besides j and i start and finish at the exact same time in S* and $\overline{S}$, so the product = ($W_i \times w_i$) of all other jobs besides i and j will be the exactly equal in S* and $\overline{S}$. Additionally, the product of job j will be smaller in $\overline{S}$ than in S* (as j now has an earlier starting time), so we only need to worry about job i and its increased product in $\overline{S}$.
  - While job i's product certainly did increase (as its starting time increased), its weight did not change at all. Job i's waiting time changed from W_i to W_i+d_j, and Job j's waiting time changed from W_j to W_j-d_i. However, i cannot increase the summation in $\overline{S}$ more than j increased it in S. This is because $r_j<r_i$, so $\overline{W}_j$ decreased a larger amount than $\overline{W}_i$ increased (as the smaller the ratio, the greater the decrease). Therefore, the swap of jobs j and i did not increase the minimum summation. (end of proof for Lemma 2A) ▫
- Therefore, there is an optimal schedule that has no inversions and no idle time. (end of proof for Lemma 2) ▫

Because there is an optimal schedule with no inversions and no idle time (Lemma 2), and all schedules with no inversions and no idle time have the same minimum summation (Lemma 1), the greedy algorithm (S) is optimal. ▫

3. State (and prove) the running time of your algorithm.

The running time of this algorithm is O(n log n), where n is the size of the duration and weight arrays. This is correct because my algorithm does:

- Iterates through both arrays to put them in a 2D list, which is O(n)
- Sorts the 2D list in increasing order of the ratio between $\frac{duration}{weight}$, which is O (n log n)
- Iterates through the 2D list to find the minimized waiting time, which is O (n)

Because my algorithm takes n+n log n+n, n log n is the dominating term of this expression, so this algorithm takes O(n log n) time.