Scheduling to Minimize Lateness: (Section 4.2 of book)

We have a set $\{1, 2, \ldots, n\}$ of requests. Each request i has a deadline $d_i > 0$ and requires a contiguous time interval of length $t_i > 0$.

Resource is available at time $t = 0$.

We want to schedule all requests. A valid schedule is one that assigns to each request i an interval $[s(i), f(i)]$ so that

(a) $s(i) \geq 0$ and $f(i) = s(i) + t_i$ for each i.

(b) The intervals assigned to any two requests don't overlap.

In a valid schedule, some jobs/requests may finish after their deadline. ~~For~~ In a given schedule, we define the lateness of i, l_i, as:

$$l_i = 0 \quad \text{if} \quad f(i) \leq d_i$$
$$= d_i - f(i) \quad \text{if} \quad f(i) > d_i.$$

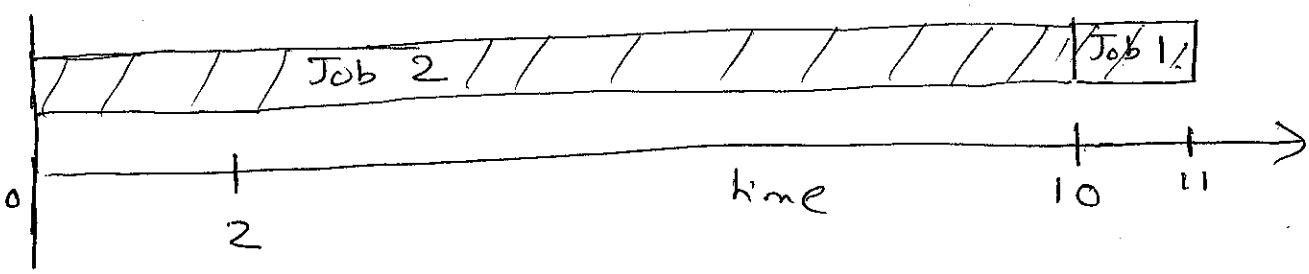
The lateness of ~~a~~ the schedule, denoted L, is $\max_i l_i$.

The algorithmic problem is to find the schedule that minimizes lateness over all valid schedules.

Example 1.

$t_1 = 1, \quad d_1 = 2$

$t_2 = 10, \quad d_2 = 10$

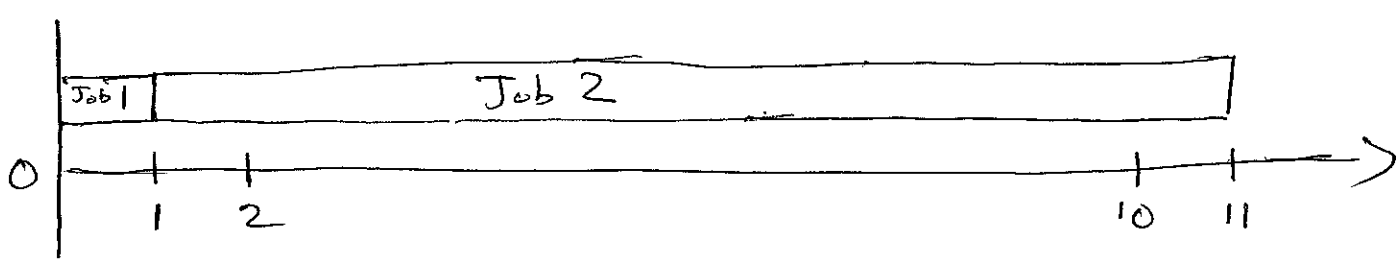


In above schedule,

$l_1 = 11 - 2 = 9$

$l_2 = 0$

So $L = 9$.



In this schedule,

$l_1 = 0$

$l_2 = 11 - 10 = 1$

So $L = 1$. This is the optimal schedule

Let us consider some greedy heuristics:

- Schedule in order of increasing length t_i.

  This works in Example 1, but not in the following

  Example 2:

  $t_1 = 1 \quad d_1 = 100$

  $t_2 = 10 \quad d_2 = 10$

- Schedule in order of increasing <u>slack</u> $d_i - t_i$:

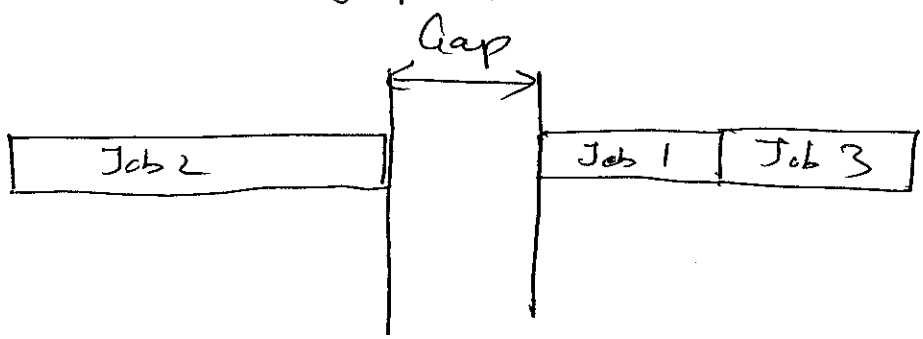
  This works in Example 2, but fails in Example 1

There is a third greedy strategy which always works. Schedule jobs in order of increasing deadlines, breaking ties arbitrarily.

We'll prove that this strategy always produces an optimal schedule.

First we need some notions.

A schedule is said to have <u>idle time</u> if it has gaps, for eg,



~~A schedule is said to have~~

An inversion in a schedule is a pair of jobs i and j so that

$d_j < d_i$ and i is scheduled before j.

Clearly, our greedy strategy produces a schedule with no inversions and no idle time.

As the first and most important step towards showing that our ~~strated~~ strategy produces an optimal schedule, we show:

<u>Claim</u>: There is an optimal schedule with no inversions and no idle time.

<u>Proof</u>: Our proof will start with an optimal schedule and transform it via a sequence of steps. At the end of these steps, we'll have a schedule with no inversions and no idle time. We'll ensure that each

step <u>does</u> <u>not</u> <u>increase</u> the lateness. This will establish the claim.

Our first step is to "collapse" the schedule so that it has no idle time. Clearly, this step cannot increase the lateness. Let O denote resulting schedule.

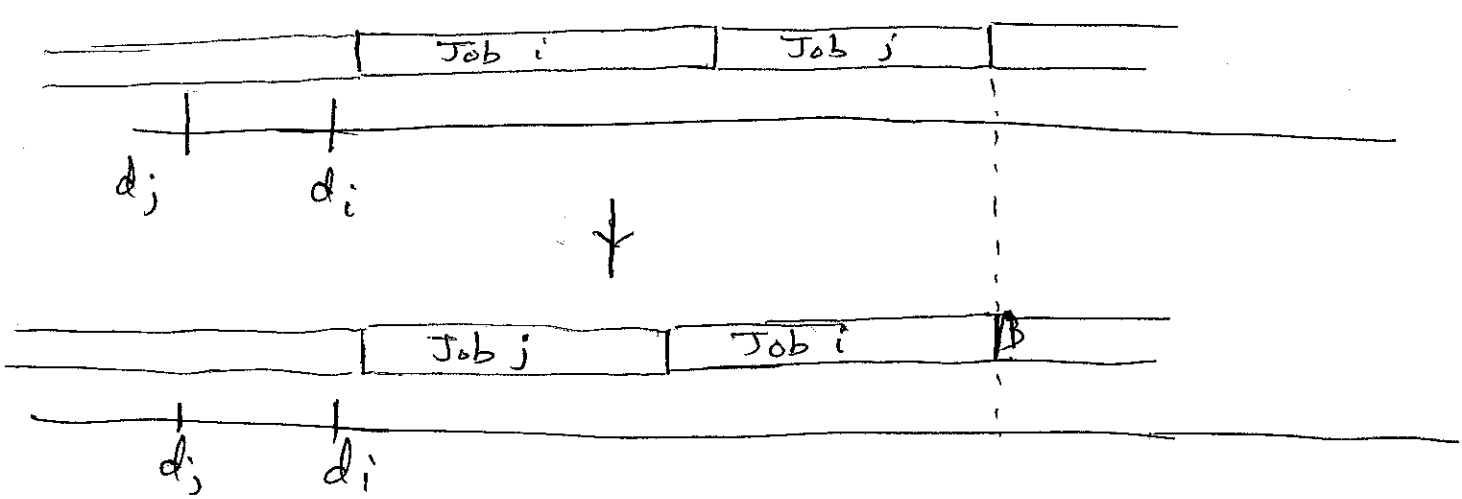
While O has an inversion do

- Find a pair of jobs i and j such that $d_j < d_i$ and j is scheduled <u>immediately</u> after i (in O). (Such an inversion exists - seeing this requires some thought.)
- Swap i and j, and call the resulting schedule O. (This swap reduces the number of inversions by 1. This requires some thought.)

endwhile.

The while loop terminates because each iteration decreases the number of inversions by 1. Thus we have specified a finite sequence of steps as promised.

All that remains is to show that the swap in each iteration of the while loop cannot increase the lateness. To show this, ~~Be~~ suppose i and j are swapped, ~~For consider~~ in a particular iteration of the while loop with $d_j < d_i$.



The lateness of jobs other than i and j do not change ~~of~~ ~~a~~ as a result of the swap.

The lateness of j does not increase as a result of the swap.

The lateness of i can increase as a result of the swap, but it is upper bounded by the lateness of j before the swap.

We conclude that each swap does not increase lateness of the schedule. This completes the proof. $\boxtimes$

<u>Claim 2</u>: All schedules with no inversions and no idle time have same lateness.

This Claim addresses the situation

where two or more jobs have the same deadline. Its point is that it does not matter how we break such ties.

From Claim 1 and Claim 2, it follows that

Claim 3: Our greedy strategy produces an optimal schedule.

Observe that our greedy strategy can be implemented in $O(n \log n)$ time.

Compare this with the naive algo, which goes through all $n!$ schedules with no idle time.