Learning objective: find classical optimal scheduling rules and extreme instances when no information about tasks is known until they arrive.

Before starting, we introduce some definitions for scheduling using Graham’s notation:

- `pmtn` in the `β` part corresponds to preemption, which allows task executions to be paused;
- `r_j` in the `β` part represents release times;
- `online` in the `β` part (which implies `r_j`) means that no information is known at execution before tasks are released;
- `L_{max}` in the `γ` part (which implies `d_j`) is the lateness objective: `max(c_j - d_j, 0)` where `c_j` is the completion time of task `j` and `d_j` is its deadline;
- ∑`f_j` in the `γ` part where `f_j = c_j - r_j` is the flowtime, the time spent by task `j` in the system.

For instance, the following instance for the problem `1|\ r_j, pmtn, online| L_{max}` contains release times, processing costs and deadlines: `{r_j} = \{0, 2, 3\}`, `{p_j} = \{3, 2, 1\}` and `{d_j} = \{6, 5, 4\}`. A possible solution consists in executing task 1 from 0 to 2, then task 2 until 4, then 1 again until 5 and finally 3 until 6. The maximum lateness in this case is `L_{max} = 2`.

**Exercise 1:** `1|r_j, pmtn, online| L_{max}`

Find the optimal schedule for the following instance: `{r_j} = \{0, 2, 3\}`, `{p_j} = \{3, 2, 1\}` and `{d_j} = \{6, 5, 4\}`.

Propose different algorithms to solve the problem and compare their solutions.

**Hint:** at each step, there is a set of available tasks with different total processing times, release times, due dates and remaining processing times.

**Exercise 2:** `1|r_j, pmtn, online| \sum f_j`

Compare the previous algorithms on the previous instance but for this problem.

Determine which strategy is the best with this objective.

**Exercise 3:** `1|r_j, pmtn, online| \sum f_j/p_j`

The flowtime is normalized by the task processing cost (also the stretch) to give more importance to small tasks. For this problem, SRPT is 2-competitive. This means that the objective value is at most twice the optimal value that can be achieved a posteriori. Find an instance for which SRPT is not optimal.

**Hint:** since SRPT is optimal for the flowtime but no the stretch (a measure that gives more importance to short tasks), it probably favors long tasks. It should be possible to find an instance where SRPT continues a long task delaying an upcoming short task whereas it would be better to do the opposite.

**Exercise 4:** `P|r_j, pmtn, online| \sum f_j`

For this problem, SRPT is `max(p_{max}/p_{min}, n/m)`-competitive. Find an instance for which SRPT is not optimal.