CSE 101 Summer 2022
Programming Assignment 2

Instructions: Solutions to this assignment must be submitted through Gradescope by Sunday, Aug 14th 2022 at 10:00 PM— no late submissions will be accepted.

Note: All necessary pseudocode may be found on the lecture slides.

1. **($1||\sum U_j$)** Given n jobs with processing times $p_1, \ldots, p_n$ and due dates $d_1, \ldots, d_n$, we will decide how to schedule the jobs on a single processor with the objective of minimizing the number of late jobs, $\sum U_j$. Moore-Hodgson's algorithm yields the optimal solution to this objective.

   Implement Moore-Hodgson's algorithm: Write two separate functions

   (a) `myMoore(P,D)` that takes as input

   - `P`: an array of n positive integers (processing times $p_1, \ldots, p_n$ of jobs $J_1, \ldots, J_n$)
   - `D`: an array of n positive integers (due dates $d_1, \ldots, d_n$ of jobs $J_1, \ldots, J_n$),

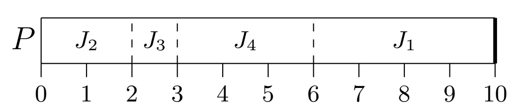
   and *returns* the schedule that results from Moore-Hodgson's algorithm, along with their corresponding end times.

   (b) `myMooreLate(P,D)` that takes in the same input, and *returns* the list of late jobs.

   For example, suppose that for a set of jobs $J_1, \ldots, J_4$

   the deadlines are: (5,2,3,4)

   and the processing times are (4,2,1,3). The schedule resulting from Moore-Hodgson's algorithm is



   where jobs J_4 and J_1 are late. Note that J_4 is the first job that has been removed from the on-time jobs and J_1 is the second job that has been removed from the on-time jobs.

   Then, calling `myMoore(P,D)` should return the schedule of the jobs as a **vector of pairs**, where the first item in each pair is the job number and the second item is the end time:

```
[(2,2), (3,3), (4,6), (1,10)]
```

   NOTE: Assume that the first job scheduled on the processor always starts at time 0. i.e. J_2 starts at time 0 and ends at 2, thus it is represented as $(2, 2)$.

   Then, calling `myMooreLate(P,D)` should return the late jobs in order:

```
[4, 1]
```

2. **(P || $\mathbf{C}_{\max}$)** Given n jobs with processing times $p_1, \ldots, p_n$, we will decide how to schedule the jobs on m parallel and identical processors (without preemption) with the objective of minimizing the makespan.

Implement List Scheduling, LPT, and SPT: Write three separate functions

(a) `myListScheduling(P,m)`

(b) `myLPT(P,m)`

(c) `mySPT(P,m)`

that take as input

- `P`: an array of n positive integers (processing times $p_1, \ldots, p_n$ of jobs $J_1, \ldots, J_n$)
- `m`: a positive integer (number of parallel and identical processors)

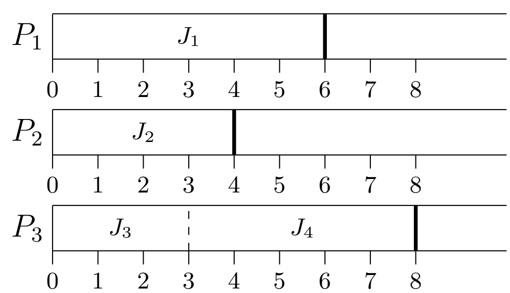
and *returns* the schedule that results from the above algorithms in the following form:

- `X`: The schedules on each processor - An array of arrays containing pairs.
  The pairs are formatted as:

```
[index of job J_j, completion time of job J_j]
```

  Note: the start time of job J_j is the completion time of job J_{j-1} on processor P_i. The start time for the first job on each processor is always 0.

For example, suppose that you receive input processing $P = (p_1, p_2, p_3, p_4) = (6, 4, 3, 5)$, $m = 3$ (as an example, job J_1 has processing time $p_1 = 6$ and we have $m = 3$ processors.) The schedule resulting from the `myListScheduling` algorithm is



Then calling `myListScheduling(P,3)` should print the following:

```
[ [ (1, 6) ],
  [ (2, 4) ],
  [ (3, 3), (4, 8) ] ]
```

First row: The first row represents the schedule on the first processor P_1 in which J_1 starts at timepoint 0 (by default) and ends at timepoint 6.

Second row: The second row represents the schedule on processor P_2 in which J_2 begins at timepoint 0 and ends at timepoint 4.

Third row: The third row shows the schedule on processor P_3 in which J_3 begins at timepoint 0 and ends at timepoint 3, and J_4 starts at timepoint 3 and finishes at timepoint 8.