# Algorithms Laboratory (CS29203)
# Assignment 7: Heap and Priority Queue
# Department of CSE, IIT Kharagpur

**17th October 2024**

---

## Question-1

**(50 points)**
You are doing scheduling of tasks for your processor. There are $n$ tasks to finish which are represented in a 2D integer array `tasks`, where `tasks[i] = [startTimei, processTimei]` denotes that the $i$-th task will be available to be processed at `startTimei`, and will take `processTimei` to finish. Your processor can process at most one task at a time and works in the following ways:

- If the processor is idle and there are no available tasks to process, the processor remains idle.
- If the processor is idle and there are available tasks, the processor will choose the one with the shortest processing time. If multiple tasks have the same shortest processing time, it will choose the task with the smallest index.
- Once a task is started, the processor will process the entire task without stopping.
- The processor can finish a task then start a new one instantly.

Given the 2D array `tasks`, you have to determine the order in which the processor will process the tasks. Your implementation **must** use the idea of **heap** data structure. The complexity of your solution should not exceed $O(n \log n)$.

Example 1:

```
(Input) tasks = [[1,2],[2,4],[3,2],[4,1]]
(Output) The order is: 0,2,3,1

Explanation:
- At time = 1, task 0 is available to process. Available tasks = {0}.
- Also at time = 1, the idle CPU starts processing task 0. Available tasks = {}.
- At time = 2, task 1 is available to process. Available tasks = {1}.
- At time = 3, task 2 is available to process. Available tasks = {1, 2}.
- Also at time = 3, the CPU finishes task 0 and starts processing task 2 as it is the shortest. Available
tasks = {1}.
- At time = 4, task 3 is available to process. Available tasks = {1, 3}.
- At time = 5, the CPU finishes task 2 and starts processing task 3 as it is the shortest. Available
tasks = {1}.
- At time = 6, the CPU finishes task 3 and starts processing task 1. Available tasks = {}.
- At time = 10, the CPU finishes task 1 and becomes idle.
```

Example 2:

```
(Input) tasks = [[7,10],[7,12],[7,5],[7,4],[7,2]]
(Output) The order is: 4,3,2,0,1

Explanation:
- At time = 7, all the tasks become available. Available tasks = {0,1,2,3,4}.
- Also at time = 7, the idle CPU starts processing task 4. Available tasks = {0,1,2,3}.
- At time = 9, the CPU finishes task 4 and starts processing task 3. Available tasks = {0,1,2}.
- At time = 13, the CPU finishes task 3 and starts processing task 2. Available tasks = {0,1}.
- At time = 18, the CPU finishes task 2 and starts processing task 0. Available tasks = {1}.
- At time = 28, the CPU finishes task 0 and starts processing task 1. Available tasks = {}.
- At time = 40, the CPU finishes task 1 and becomes idle.
```

## Question-2

**(50 points)**
You are working as a scheduling manager in an automation company. There are different machines to process tasks, and each machine needs different amount of electrical power to operate. There are multiple tasks which you have to assign to these machines efficiently. The rules of assigning tasks to machines are described as follows.

Consider two integer arrays `machines` and `tasks` of lengths $n$ and $m$ respectively. `machines[i]` is the amount of power (in kilo Watt) needed by the $i$-th machine, and `tasks[j]` is the time needed to process the $j$-th task in hours. Tasks are assigned to the machines using a machine queue. Initially, all machines are free, and the queue is empty.

At hour $j$, the $j$-th task is inserted into the queue (starting with the 0-th task being inserted at second 0). As long as there are free machines and the queue is not empty, the task in the front of the queue will be assigned to a free machine with the lowest required power, and in case of a tie, it is assigned to a free machine with the smallest index.

If there are no free machines and the queue is not empty, we wait until a machine becomes free and immediately assign the next task. If multiple machines become free at the same time, then multiple tasks from the queue will be assigned in order of insertion following the power and index priorities above. A machine that is assigned task $j$ at second $t$ will be free again at second `t + tasks[j]`.

Given the arrays `machines` and `tasks`, your job is to find an array `assign` of length $m$ where `assign[j]` is the index of the machine the $j$-th task will be assigned to. Your implementation **must** use the idea of **heap** data structure. The complexity of your solution should not exceed $O((n+m)\log n)$.

Example 1:

```
(Input) machines = [3,3,2]
        tasks = [1,2,3,2,1,2]

(Output) assign = [2,2,0,2,1,2]

Explanation:
- At second 0, task 0 is added and processed using machine 2 until second 1.
- At second 1, machine 2 becomes free. Task 1 is added and processed using machine 2 until second 3.
- At second 2, task 2 is added and processed using machine 0 until second 5.
- At second 3, machine 2 becomes free. Task 3 is added and processed using machine 2 until second 5.
- At second 4, task 4 is added and processed using machine 1 until second 5.
- At second 5, all machines become free. Task 5 is added and processed using machine 2 until second 7.
```

Example 2:

```
(Input) machines = [5,1,4,3,2]
        tasks = [2,1,2,4,5,2,1]

(Output) assign = [1,4,1,4,1,3,2]

Explanation:
- At second 0, task 0 is added and processed using machine 1 until second 2.
- At second 1, task 1 is added and processed using machine 4 until second 2.
- At second 2, machines 1 and 4 become free. Task 2 is added and processed using machine 1 until second 4.
- At second 3, task 3 is added and processed using machine 4 until second 7.
- At second 4, machine 1 becomes free. Task 4 is added and processed using machine 1 until second 9.
- At second 5, task 5 is added and processed using machine 3 until second 7.
- At second 6, task 6 is added and processed using machine 2 until second 7.
```