**Documentation Project 3**

**Advanced Algorithm and Data Structures**

Group XX (non so che numero siamo)

Exercise 1:

Exercise 2:

*Import:*

* **time**
* **random**
* **AdaptableHeapPriorityQueue** from **priority\_queue.adaptable\_heap\_priority\_queue**

The exercise requires the implementation of a non pre-empitive scheduler. It’s has been developed the class Scheduler that inherits from **AdaptableHeapPriorityQueue**. We have chosen this kind of implementation among all the possible priority queue’s implementation because it has logarithmic time complexity for the basic operation:

* Insert;
* Update;
* delate.

The **AdaptableHeapPriorityQueue** allows us also to change the priority of an element. The element stored contains a key, the associated priority, and a name, the associated task. It also stores the number of time slices after which the priority of a task in the scheduler itself must be incremented. The scheduler class has the following public methods:

* \_\_init\_\_:

The constructor of the class. It requires the parameter slice\_to\_increment.

In this method the super() constructor is called, the value slice\_to\_increment is saved and it’s defined another attribute of the class: number\_time\_slice that indicates how many time slices have been spent.

* add\_job:

This method allows the insertion of a new job in the scheduler. It requires as input parameters, the priority of the task *k* and a tuple *v* that contains the name of the job and how many time slices it requires. This method verifies that the priority of the job is a valid number (otherwise an Exception is raised), re-arrange the information present in the tuple and insert the job into the Scheduler via the *add*() method present in **AdaptableHeapPriorityQueue.**

* job\_execution:

This method prints on the output information about the current job in execution, if the scheduler is empty the message “*The scheduler has no tasks.”* is printed. This method is responsible for the update of the priority of the job, the method *update*() in the class **AdaptableHeapPriorityQueue** is used.

In the same script is also present a function, **random\_add(scheduler)**, that randomly adds a job in the scheduler. The job could be chosen among a list: *CPU*, *Memory* and *I/O.* The priority of each task is randomly generated and even the number of time slices for each task is randomly generated. When the script is run the first time, it requires as input the number of time slices after which the priority must be incremented. A check is present to verify that the user has inserted a valid number. After this “setup” phase, five jobs are added to the scheduler. In the infinite loop, it is generated a random value and if it is greater than a threshold, a new task is added to the scheduler.