

Exercice 2 (6 points)

Cet exercice porte sur l'algorithmique, les structures de données, et la gestion de processus.

1. Le code ci-dessous permet d'initialiser les tâches `tache1` et `tache2`

```
1 tache1 = Tache(1, "Répondre aux e-mails", 45)
2 tache2 = Tache(2, "Ranger ma chambre", 60)
```

2. On complète le code de la méthode `avancer`.

```
1 def avancer(self, n):
2     self.duree_restante = self.duree_restante - n
```

3. On complète le code de la méthode `est_terminee`.

```
1 def est_terminee(self):
2     return self.duree_restante <= 0
```

4. Avec la représentation de l'énoncé, et en partant de la file proposée, on obtient après insertion :

[debut] (<t3>, 4) (<t7>, 4) (<t1>, 3) (<t2>, 3) (<t6>, 2) (<t4>, 1) (<t5>, 1) [fin]

5. En partant de [début] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin], la valeur de `f.defiler()[0]` est <t3> et la file devient [début] (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin].

6. En repartant de la file [début] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin], la valeur de `f.examiner()[1]` est 4 et la file est inchangée.

7. On complète le code de la fonction `ajouter_file_prio`.

```
1 def ajouter_file_prio(f, t, p):
2     f_aux = File()
3     while not f.est_vide() and f.examiner()[1] >= p:
4         f_aux.enfiler(f.defiler())
5     f_aux.enfiler((t, p))
6     while not f.est_vide():
7         f_aux.enfiler(f.defiler())
8     while not f_aux.est_vide():
9         f.enfiler(f_aux.defiler())
```

8. En comptant en nombre d'opérations sur les files (`examiner`, `est_vide`, `defiler`, `enfiler`), les deux premières boucles `while` nécessitent $4k + 3(m - k) \leq 4m$ opérations, où $k \leq m$ est le nombre de tours de la première boucle, et la dernière boucle `while` nécessite $3(m + 1)$ opérations, donc en tout dans le pire des cas $7m + 4$ opérations avec la ligne 5. C'est donc un coût d'exécution temporel **linéaire**.

9. Les tâches seront exécutées selon les blocs :

3	7	3	3	3	1	2	1	2	2	6	6	6	4	5	4	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

10. On écrit le code d'une fonction `planning`.

```
1 def planning(f):
2     taches=[]
3     while not f.est_vide():
4         tache_a_effectuer, prio = f.defiler()
5         tache_a_effectuer.avancer(25)
6         if not tache_a_effectuer.est_terminee():
7             ajouter_file_prio(f, tache_a_effectuer, prio)
8         taches.append(tache_a_effectuer)
9     return taches
```