❑ **Exercice 1** : *complexité des opérations*
On considère les structures de données suivantes :

— Un tableau de taille fixe

— Une pile implémentée à l'aide d'une liste simplement chainée

— Une file implémentée à l'aide d'une liste simplement chainée ayant un accès sur son dernier élément

Donner la complexité des opérations suivantes sur chacune de ces structures de données :

1. Accès au n-ième élément
2. Insertion d'un élément au début (au sommet pour la pile)
3. Insertion à la fin (tout en bas pour la pile)
4. Suppression du premier élément
5. Suppression du dernier élément
6. Test d'appartenance d'un élément

❑ **Exercice 2** : *Inversion au sommet*
On suppose qu'on dispose d'une structure de données de type pile dotée de son interface habituelle c'est-à-dire `empiler`, `dépiler` et `est_vide`. Proposer une suite d'opérations permettant d'inverser, lorsqu'ils existent, les deux éléments situés au sommet de cette pile. Si la pile contient moins de deux éléments, elle doit rester en l'état.

❑ **Exercice 3** : *Taille d'une file*
Ecrire une fonction prenant en entrée une file file F et renvoyant sa taille. La file F ne doit pas être détruite mais restituée à son état initial et on ne dispose que de l'interface usuelle d'une file (qu'on rapellera).

❑ **Exercice 4** : *list de OCaml*

1. Rappeler la complexité de l'opérateur `::` en OCaml. Expliquer cette complexité
2. Même question pour `@`
3. On considère deux listes en OCaml : `l1=[2; 4; 8]` et `l2=[2; 3; 5; 7]`. Réprésenter en mémoire la liste `l` obtenue à l'aide de `let l = l1@l2;;`

❑ **Exercice 5** : *Un exemple de complexité amortie*
On prend l'exemple de la structure de données implémentant le type de *list* de Python grâce à un tableau dynamique en C (voir TP). On considère une *list* dont la taille initiale est $t_0 > 0$ et on supposera que la capacité initiale est $c_0 = 2t_0$. Sur cette structure de données, on souhaite déterminer la complexité amortie de l'opération `append` qui consiste à ajouter un élément en fin de liste, deux cas peuvent se produire :

- $c > t$ et donc on peut ajouter un nouvel élément sans redimensionner le tableau.
- $c = t$ et on redimensionne le tableau en *doublant* la capacité du tableau afin d'y insérer le nouvel élément

Montrer que la complexité amortie de `append` est un $\mathcal{O}(1)$.
🛟 Indication : on pourra introduire $\Phi(S) = 2t - c$.