❑ **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 un tableau donc la taille initiale est 1 et sur lequel on effectue n opérations `append`. Le but de l'exercice est de montrer qu'on obtiendra alors un nombre d'opérations en $O(n)$. On dira alors que le `append` a une **complexité amortie** en $O(1)$.

1. Montrer que le coût du redimensionnement d'un tableau de taille n est un $O(n)$.
2. Montrer que n opérations `append` vont nécessiter $\lfloor \log_2(n) \rfloor$ redimensionnement de tableau.
3. Donner la taille des tableaux lors de chaque redimensionnement.
4. En déduire le coût des redimensionnements.
5. Montrer que le coût total est un $O(n)$.

❑ **Exercice 6** : *Implémentation d'une file avec deux piles*

On reprend l'implémentation d'une file avec deux piles (voir TP), le but de l'exercice est d'établir la complexité de `défiler`

1. Donner les complexités dans le pire et le meilleur des cas
2. On suppose à présent qu'en tout on a enfilé et défilé n éléments. Montrer que le nombre total d'opérations nécessaire est un $O(n)$.