Corrigé

Tri par piles

1. Tri avec une pile

Question 1.

a) La séquence $(2,4,1,3)$ n'est pas triable car $(2,4,1)$ ne peut être trié.

La séquence $(3,1,2,5,4)$ est triable par la suite EEDEDDEEDD.

La séquence $(4,5,3,7,2,1,6)$ n'est pas triable car $(4,5,3)$ ne peut être trié.

b) Supposons qu'à une certaine étape d'un tri valide il existe i et j dans la pile tel que $i < j$ et que i soit situé en dessous de j. Alors i ne peut sortir de la pile avant j, ce qui est absurde. La pile est donc à tout moment triée par ordre croissant à partir du sommet de la pile.

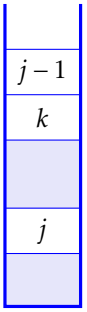
c) Plaçons-nous en cours de tri, et considérons le prochain élément à devoir sortir de la pile. S'il se trouve au sommet, il doit obligatoirement sortir à l'étape suivante sous peine d'être recouvert par un autre élément. Et s'il ne se trouve pas au sommet de la pile, la seule opération possible consiste à empiler (s'il en reste) le prochain élément de la séquence à devoir entrer. La démarche est donc déterministe à chaque étape.

```
def triPile(a):
    n = len(a)
    p = Pile()
    k = 0
    s = 0
    for _ in range(2*n):
        if not p.empty() and p.peek() == s + 1:
            s = p.pop()
        elif k < n:
            p.push(a[k])
            k += 1
        else:
            return False
    return True
```

L'entier k désigne l'indice du premier élément de la séquence non encore entré dans la pile ; l'entier s désigne le dernier des éléments à en être sorti.

Question 2. Si $(2,3,1)$ est un motif de a, il existe $i < j < k$ tel que ces éléments soient dans l'ordre $\cdots j \cdots k \cdots i \cdots$ dans a. Puisque i est inférieur à j et k, ces deux éléments doivent encore être dans la pile lorsque i y entre. Mais alors j est situé sous k et les éléments de la pile ne sont pas ordonnés par ordre croissant à partir du sommet. a n'est donc pas triable.

Réciproquement, si a n'est pas triable c'est qu'il existe une étape bloquante pour l'algorithme ci-dessus. Notons j le premier élément à ne pas pouvoir sortir de la pile. On a nécessairement $j \geqslant 2$ (car 1 peut toujours sortir) donc on peut considérer le moment où $j-1$ est sorti de la pile. À ce moment là j est déjà dans la pile car sinon il pourrait lui aussi sortir. De même, il ne peut être situé immédiatement en dessous de $j-1$. Notons donc k l'élément situé juste en dessous de $j-1$.
Puisque k n'est pas encore sorti, c'est que $k > j-1$. Et comme on vient de le dire, ce ne peut être j, donc $k > j$.
Posons alors $i = j-1$. On a $i < j < k$ et compte tenu de leurs positions relatives dans la pile, on a $a = \cdots j \cdots k \cdots i \cdots$ donc $(2,3,1)$ est bien un motif de a.



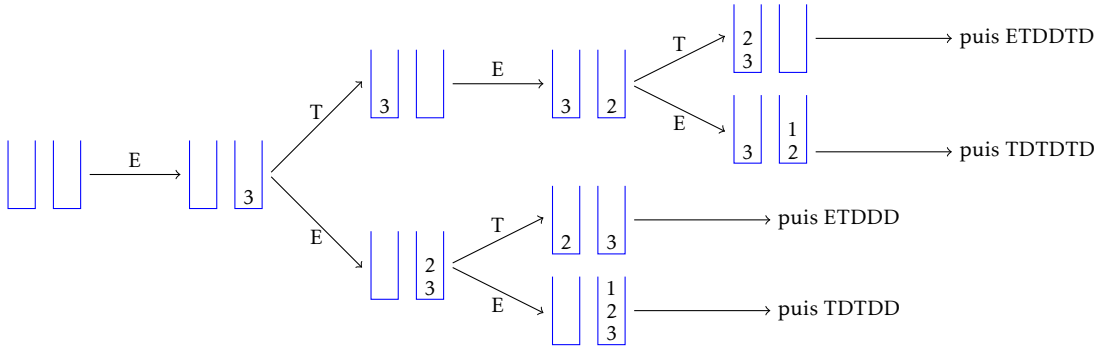
Question 3. Considérons une séquence triable a de $[\![1,n]\!]$, et notons k le rang où est situé l'entier n dans cette séquence. Notons b la séquence des $k-1$ entiers qui le précède, et c la séquence des $n-k$ entiers qui le suivent.
Si x appartient à b et y appartient à y alors (x,n,y) est un motif de a donc $x<y<n$ (faute de quoi a ne serait pas triable). Ainsi, b est une séquence triable de $[\![1,k-1]\!]$ et c une séquence triable de $[\![k,n-1]\!]$ (triables car ne comportant pas le motif $(2,3,1)$).
Il y a c_{k-1} possibilités pour b et c_{n-k} pour c donc $c_n = \sum_{k=1}^{n} c_{k-1}c_{n-k}$.

2. Tri avec deux piles en série

Question 4.

a) On peut trier $(2,4,3,1)$ par exemple par les séquences EETEETDTTDDD ou ETEEETDDTDTD.

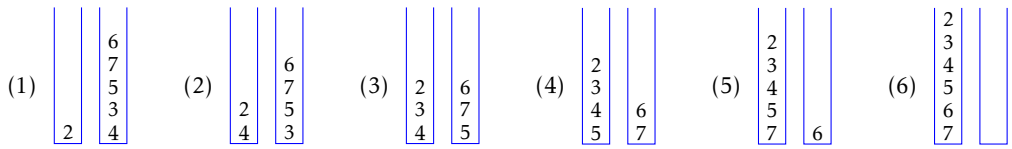
b) Une étude exhaustive montre qu'il y a quatre façons de trier $(3,2,1)$:



c) Notons u_n le nombre de tris possible de la séquence $(n,n-1,\dots,2,1)$, et intéressons-nous au mouvement de n : le premier empilement et le dernier dépilement doivent concerner cet entier, et il ne peut être transféré de P_1 vers P_2 qu'à la condition qu'il n'y ait aucun autre élément dans chacune des deux piles. Cet événement ne peut donc se produire qu'à la seconde ou à l'avant dernière étape ; il n'y a donc que deux possibilités : ET$\cdots$D ou E$\cdots$TD. Les autres mouvements concernent le tri de la séquence $(n-1,\dots,2,1)$; il y a par définition u_{n-1} tris possibles.
On en déduit que $u_n = 2u_{n-1}$, ce qui donne compte tenu de la question précédente : $u_n = 2^{n-1}$.

d) Supposons cette séquence triable et considérons l'instant où 1 sort. Puisqu'il est entré en dernier dans le système, tous les autres entiers doivent être dans l'une des deux piles.
Cherchons maintenant où se trouve 2.

– S'il est dans P_2, le contenu de cette pile dépend des éléments qui ont été transférés avant lui, ce qui laisse 6 configurations possibles, sachant que P_2 doit être trié par ordre croissant :



Les trois premières configurations conduisent à des impasses car la séquence $(6,7,5)$, isomorphe à $(2,3,1)$, n'est pas triable à l'aide de la seule pile P_2.
Or les trois dernières ne peuvent être atteintes car cela signifierait qu'à l'aide de la seule pile P_1 on pourrait trier $(4,3,5)$ en $(5,4,3)$, ce qui n'est pas possible (la séquence $(4,3,5)$, isomorphe à $(2,1,3)$, est équivalent au motif $(2,1,3)$ lorsqu'il s'agit de trier par ordre décroissant).

– S'il est encore dans P_1, il se trouve tout au fond et la suite de la séquence va consister à transférer les éléments situés au dessus de lui avant de pouvoir faire sortir 2. À ce moment, la configuration atteinte doit être celle représentée en (6), et nous avons vu qu'elle est hors d'atteinte.

Un algorithme glouton

Question 5.

a) si i est j sont présents simultanément dans P_2 à un moment donné et si j est situé sous i, alors i en sortira avant j et donc $i<j$. P_2 est donc toujours rangé par ordre croissant à partir de son sommet.

b) En revanche, ce n'est pas toujours le cas de la pile P_1, comme par exemple lors du tri de la séquence $(2,1,3)$ par la succession d'opérations EEETTDTDD.

Question 6.

a) L'algorithme trie la séquence $(2,4,3,1)$ en appliquant les opérations ETEEETDDTDTD.

b) Soit a une séquence triable par une pile, et s le mot sur l'alphabet $\{E,D\}$ qui la trie. Alors il suffit d'appliquer la transformation $(E \mapsto ET)$ sur s pour obtenir la séquence d'opération qui trie a à l'aide de deux piles en série en suivant l'algorithme décrit. En effet, tout empilement dans P_1 est immédiatement suivi d'un transfert licite vers P_2 d'après la question 1b.

c) En suivant cet algorithme toute séquence de longueur inférieure ou égale à 4 peut être triée, et seules deux séquences de longueur 5 ne le peuvent pas : $(2,3,5,4,1)$ et $(4,2,3,5,1)$. Ces deux séquences peuvent néanmoins être triées, la première en appliquant les opérations EETTEEETDDDTDTD et la seconde les opérations ETEETTEETDDDDTD.

d) On peut rédiger cet algorithme de la façon suivante :

```
def triPileLeftToRight(a):
    n = len(a)
    p1 = Pile()
    p2 = Pile()
    k = 0
    s = 0
    for _ in range(3*n):
        if not p2.empty() and p2.peek() == s+1:
            s = p2.pop()
        elif not p1.empty() and (p2.empty() or p1.peek() < p2.peek()):
            p2.push(p1.pop())
        elif k < n:
            p1.push(a[k])
            k += 1
        else:
            return False
    return True
```

Comme dans l'algorithme précédent, k désigne l'indice du premier élément de la séquence non encore entré dans la pile et s le dernier des éléments à en être sorti.

Tri par sas

Question 7.

```
def valide(a, c):
    n = len(a)
    p1 = Pile()
    p2 = Pile()
    k = 0
    s = 0
    i = 0
    for _ in range(n+len(c)):
        if i < len(c) and not p1.empty() and p1.peek() == c[i]:
            p2.push(p1.pop())
            i += 1
        elif k < n:
            p1.push(a[k])
            k += 1
        else:
            return False
    for _ in range(2 * n - len(c)):
        if not p2.empty() and p2.peek() == s+1:
            s = p2.pop()
        elif not p1.empty():
            p2.push(p1.pop())
        else:
            return False
    return True
```

Ici, l'entier i désigne l'indice du prochain élément de la liste c à devoir être transféré de P_1 vers P_2.

Question 8.

a) La démarche est simple : c doit être une suite extraite de $(n, n-1, \ldots, 1)$; il suffit de les passer toutes en revue jusqu'à en trouver une qui convienne.

```
def triSas(a):
    n = len(a)
    for j in range(n+1):
        for c in combinations(range(n, 0, -1), j):
            if valide(a, c):
                return True
    return False
```

b) Le coût de la fonction `valide` est un $O(n)$ (chaque élément de a doit effectuer chacune des trois opérations E, T, D une et une seule fois). Sachant qu'il y a 2^n suites extraites c possibles, le coût de la fonction `triSas` est un $O(n2^n)$.