Table des matières

Préliminaires : listes

Exercice 1 *Listes*

1. Vrai ou faux : `len(L)` renvoie le nombre d'éléments d'une liste `L` ?
2. Vrai ou faux : `range(n)` correspond aux entiers de $[\![0, n]\!]$?
3. Vrai ou faux : `range(n)` correspond aux entiers de $[\![0, n-1]\!]$?
4. Vrai ou faux : `range(1,n)` correspond aux entiers de $[\![1, n-1]\!]$?
5. Vrai ou faux : `L[0]` correspond au premier élément de la liste `L` ?
6. Vrai ou faux : `L[n-1]` correspond au dernier élément de la liste `L` de longueur n ?

I. Tri par sélection, via une recherche de minimum

I.1 Principe algorithmique

Il s'agit de placer successivement, de gauche à droite, les minimums successifs, en parcourant la liste.

Partant d'une liste $L = [\ell_0, \ell_1, \ldots, \ell_{n-1}]$, on recherche successivement, pour i allant de 0 à $n-1$, l'élément le plus petit parmi les $n-i$ restant à trier, et on l'échange si besoin avec l'élément en i-ème position.

Pour cela, on utilise les variables locales :

- `n` qui représente la taille de la liste L à trier
- `i` qui représente un entier entre 0 et $n-1$ permettant d'obtenir successivement les i premiers éléments triés
- `j` qui représente un entier entre $i+1$ et $n-1$ permettant de comparer $L[i]$ aux éléments suivants
- `selec` qui représente l'élément selectionné pour être le minimum des $n-i$ éléments à trier lors de la i-ème étape.
- `imin` qui représente l'indice entier de `selec`

I.2 Code Python

Complétez le code Python ci-dessous pour modifier la liste `L` en une liste triée

```
n=len(L)
for i in range(...) :
    # on cherche le minimum selec et son indice imin parmi les non-triés
    imin=i
    selec=L[i]
    for j in range(i+1,n) :
        # on teste si l'élément L[j] en jème est plus petit que selec
        if L[j]<..... :
            # si oui, on met à jour la valeur de imin et selec
imin=j
            selec=.....
    if imin !=i :
        # Si L[i] n'était pas le min parmi les non-triés on l'échange avec le min
        L[i],L[imin]=..... , .....
```

I.3 Exemple pratique du tri par sélection

Le **tri sélection** est l'un des tris les plus simple à comprendre. Lorsque l'on organise les photos de classe, le photographe demande à avoir les plus petits devant, et les plus grand au fond. La maîtresse envoie donc les élève s'installer un par un, dans l'ordre croissant. Pour ce faire, elle cherche d'abord l'élève le plus petit, et l'envoie. Puis le second élève plus petit et ainsi de suite. Le tri sélection utilise cette méthode pour trier des données dans l'ordre croissant. Le principe est de sélectionner le plus petit élément parmi ceux qu'il reste à trier.

7	2	1	5	8	3	3

$i = 0$: pour j allant de 1 à 7, on parcourt la liste et on trouve `imin=2` pour | 1 | que l'on échange avec `L[0]` | 7 |

1	2	7	5	8	3	3

$i = 1$: pour j allant de 2 à 7, on parcourt la liste et on trouve `imin=1` pour | 2 |, pas d'échange pas avec `L[1]` | 2 |

1	2	7	5	8	3	3

$i = 2$: pour j allant de 3 à 7, on parcourt la liste et on trouve `imin=5` pour | 3 |, pas d'échange pas avec `L[1]` | 7 |

1	2	3	5	8	7	3

...

L'algorithme du tri par sélection est assez simple puisqu'il est constitué de $n-1$ recherches de maximum. On considère dans le pseudo-code que les listes sont numérotées de 0 à $n-1$. La boucle située entre les lignes 6 et 12 effectue la recherche de l'indice correspondant au maximum dans le sous-tableau L[i + 1 . . .] La boucle commençant en ligne 3 a pour borne supérieure n - 2 car lorsque tous les n - 1 premiers éléments sont en place, le dernier l'est automatiquement.

I.4 Algorithme

Algorithm 1: Tri sélection

Data: L liste non triée
Result: liste triée croissante correspondante

```
1  begin
2      n ← longueur(L);
3      for i de 0 à n − 2:
4          imin ← i
5          t ← L[i]
6          for j de i + 1 à n − 1:
7              if L[j] < t:
8                  imin ← j
9                  t ← L[j]
10         if imin ≠ i:
11             échanger L[i] et L[imin]
12     return L
13 end
```

I.5 Programmation Python

```
def tri_select(L) :
    n=len(L)
    for i in range(n-1) :
        imin=i
        selec=L[i]
        for j in range(i+1,n) :
            if L[j]<selec :
                imin=j
                selec=L[j]
        if imin !=i :
            L[i],L[imin]=L[imin],L[i]
    return L
```

I.6 Complexité

Le nombre de comparaisons de données est
$$\sum_{i=0}^{n-2}\left(\sum_{k=i+1}^{n-1}\right) = \frac{n(n-1)}{2} = O(n^2).$$
On admet que le nombre d'échanges moyen est $\Theta(n - \ln n)$, le nombre maximal d'échanges est $n-1$, le nombre de passages dans la boucle principale.

II. Tri par insertion

II.1 Principe algorithmique

Partant d'une liste non triée, il s'agit de déplacer ses éléments vers la gauche un à un, en les ordonnant par ordre croissant.

Partant d'une liste $L = [\ell_0, \ell_1, \ldots, \ell_{n-1}]$, on part de la liste $[\ell_0]$, puis pour i allant de 1 à $n-1$, on insère à la bonne place l'élément ℓ_i dans la liste trié contenant $\{\ell_0, \ldots, \ell_{i-1}\}$.

Pour cela, on utilise les variables locales :

- `j` qui représente le nombre d'éléments contenus dans la liste déjà triée.
- `i` qui représente un entier allant de $j-1$ à 0 permettant d'obtenir la position d'insertion de ℓ_j
- `cle` qui représente l'élément ℓ_j à comparer pour être inséré en bonne position

II.2 Code Python

Complétez le code Python ci-dessous pour modifier et trier la liste `L`.

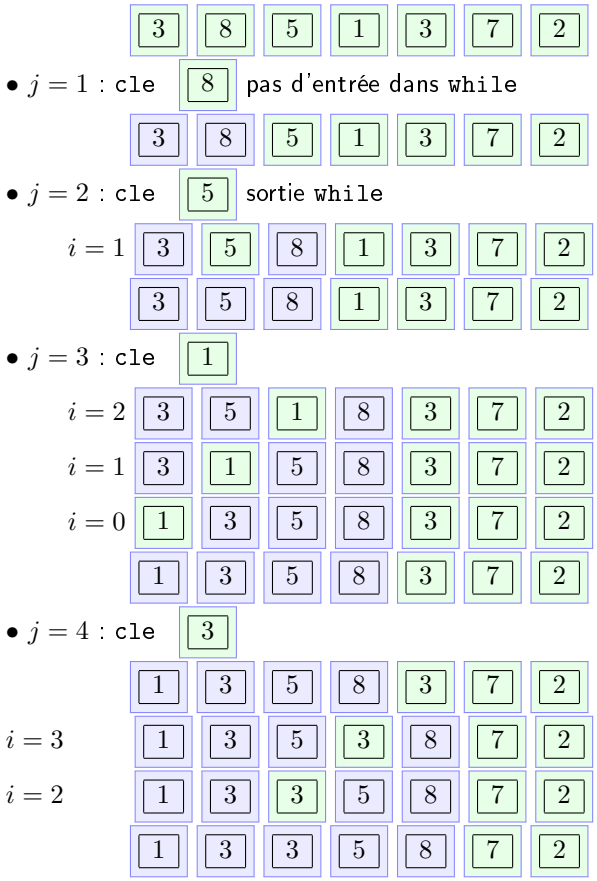
```
n=len(L)
for j in range(1,n) :
    # on stocke dans cle la valeur de ℓ_j
    cle=...
    # on initialise i à la valeur j-1 du dernier indice à droite de la liste déjà triée
    i=j-1
    # on va diminuer i tant que cle < ℓ_i et déplacer ℓ_i vers la droite
    while i>=... and cle<A[i] :
        L[i+1]=L[i]
        i=i-1
    # on place la cle à la bonne position
    L[i+1]=...
    # en sortie de boucle while les éléments de 0 à j sont alors triés
# en sortie de boucle for les éléments de 0 à n-1 sont alors triés
```

II.3 Exemple pratique du tri par insertion

Le tri insertion est un algorithme efficace de tri d'un petit nombre d'éléments.

L'algorithme de tri par insertion est l'un des plus simples. Le principe est de parcourir le tableau T en procédant de manière à maintenir la partie allant de l'indice 0 à l'indice $j-1$ triée.

Supposons que l'on ait la disposition ci-contre où la partie bleutée est triée. Pour progresser dans l'algorithme, il faut placer l'élément d'indice j à la bonne place. Pour cela, on le permute avec l'élément précédent jusqu'à ce que ce soit le cas.



Son principe de fonctionnement est celui qui est souvent utilisé pour trier un jeu de cartes.

On commence avec la main gauche vide et le paquet de carte retourné sur la table. On retourne ensuite la première carte du paquet et on l'amène à la bonne position dans la main gauche. Pour trouver cette bonne position, on compare la carte piochée aux cartes de la main gauche une par une, de droite à gauche. A chaque instant, les cartes dans la main gauche sont triées et la main est composée des carte qui étaient au dessus du paquet.

II.4 Algorithme

Algorithm 2: Tri insertion

Data: $A[0, n-1]$, Variable GLOBALE
Result: on trie par insertion la variable A

```
1  begin
2      for j de 1 à longueur(A) − 1:
3          cle ← A[j]
4          i ← j − 1
5          while i ≥ 0 et A[i] > cle:
6              A[i + 1] ← A[i]
7              i ← i − 1
8          fintq
9          A[i + 1] ← cle
10     return L
11 end
```

II.5 Programmation Python

```
def tri_insertion(L) :
    n=len(L)
    for i in range(1,n) :
        cle=L[i]
        j=i-1
        while j>=0 and L[j]>cle :
            L[j+1]=L[j]
            j=j-1
        L[j+1]=cle
    return L
```

II.6 Complexité

Le cas le meilleur est lorsque les données sont déjà triées, le cas le pire est lorsque les données sont triées en sens inverse. Le nombre de comparaisons de données varie entre n et $\frac{n(n-1)}{2}$, et est en moyenne égal à $\frac{(n+8)(n-1)}{4} = \Theta(n^2)$.

On admet que le nombre d'échanges moyen est $\Theta\left(\frac{n^2+3n}{4} - \log n\right)$, le nombre maximal d'échanges est $n-1$ le nombre de passages dans la boucle principale.