Le tri par sélection (Algo 10-11)

I – Algorithme

But : trier sur place un tableau initialement non trié. Le tri se fera ici dans l'ordre croissant.

Description des entrées / sortie

- ENTREES :
  - **tableau** : un tableau
  - (option) **longueur** : le nombre d'éléments dans le tableau
- SORTIE : aucune. Le tableau est trié sur place. On modifie directement tableau.

Exemple : `[13, 18, 89, 1024, 45, -12, 18]` deviendrait

En Français

- Pour chaque index **debut** du tableau
  - On cherche l'**index_min** contenant le plus petit élément dans le sous-tableau [index debut ; fin].
  - On intervertit les contenus des **index debut** et **index_min**.

Algorithme du `tri_selection(tableau)`

```
POUR debut variant de 0 à (longueur – 1)
    index_min ← minimum(tableau, debut)
    intervertir(tableau, debut, index_min)
Fin POUR
Renvoyer VIDE (⌀)
```

Tant que équivalent :

Algorithme de `minimum(tableau, debut)`

```
index_min ← debut
POUR index variant de (debut + 1) à (longueur - 1)
    SI tableau[index] < tableau[index_min]
        index_min ← index
    Fin SI
Fin POUR
Renvoyer index_min
```

Tant que équivalent :

Algorithme de `intervertir(tableau, debut, index_min)`

```
temp ← tableau[debut]
tableau[debut] ← tableau[index_min]
tableau[index_min] ← temp
Renvoyer VIDE (⌀)
```

Implémentation Python

→ *sur la page suivante*
(sans commentaire, ni documentation)

Allez voir la question 13 de l'activité Algorithme Tri par sélection pour la version complète.

Le prototype décrit les paramètres et le retour en utilisant le typing pour gagner de la place.

```python
import random as rd

serie1 = [rd.randint(0, 100) for x in range(20)]

def intervertir(tableau:list, a:int, b:int) -> None :
    temp = tableau[a]
    tableau[a] = tableau[b]
    tableau[b] = temp

def minimum(tableau:list, debut:int) -> int :
    index_minimum = debut   !
    for index in range(debut+1, len(tableau)) :
        if tableau[index] < tableau[index_minimum] :
            index_minimum = index
    return index_minimum

def tri_selection(tableau:list) -> None:
    for debut in range(len(tableau)) :
        index_min = minimum(tableau, debut)
        intervertir(tableau, debut, index_min)
```

Remarque : VOCABULAIRE

Création par _____ `serie1 = [rd.randint(0, 100) for x in range(20)]`

Création par _____

```
serie2 = []
    for x in range(20) :
        serie2.append(rd.randint(0,100))
```

II – Preuve de terminaison (variant)

Définition :

Pour prouver qu'une boucle non bornée (Tant que) s'arrête, il faut montrer

- que la boucle peut s'écrire _____ et
- que la suite $\mathbf{u_n}$ est une suite à valeurs
  - **entières**
  - **strictement décroissante** (on est certain que la valeur diminue à chaque boucle)
  - positives (on a bien une valeur supérieure à 0 au début)

Cette suite u_n est ce qu'on nomme le _____.

Question 01 : Montrer la terminaison de cet algorithme

Remarque : BOUCLE FOR

```
i ← 45
TANT QUE i < 200
    Faire un truc (qui ne modifie pas la valeur de i !)
    i ← i + 10
Fin TANT QUE
```

Question 02 : Montrer la terminaison de **minimum**

Question 03 : Montrer la terminaison de **tri_selection**

III – Coût quadratique du tri par sélection

Question 04 : On considère un tableau de **n** éléments. On considère qu'on effectue le premier tour de boucle. La variable **debut** contient donc **0**. Combien de comparaison va faire la fonction **minimum** avant de répondre ?

Question 05 : On considère le deuxième tour de boucle. La variable debut contient donc 1. Combien de comparaison va devoir faire la fonction minimum avant de renvoyer sa réponse sur ce tour de boucle ? Combien a-t-il fallu de comparaisons au total pour l'instant ?

Question 06 : A votre avis, combien de comparaisons au total lors du prochain tour de boucle ?

Question 07 : Combien l'algorithme va-t-il faire de comparaisons pour déterminer le contenu de l'avant-dernière cas, lorsque debut vaut longueur-2 ?

Question 08 : Combien l'algorithme va-t-il faire de comparaisons pour déterminer le contenu du dernière cas, lorsque debut vaut longueur-1 ? Combien de comparaisons aura-t-on effectué au total sur tout le tableau ?

On obtient :

$$C_n = 1 + 2 + 3 + ...n - 3 + n - 2 + n - 1 = \frac{(n-1)n}{2} \quad \text{ou encore} \quad C_n = \sum_{x=1}^{n-1} x = \frac{(n-1)n}{2}$$

Question 09 : Démontrer cette formule (sans utiliser les formules vues éventuellement en spécialité mathématiques de première).

Question 10 : Retrouver la formule en utilisant cette fois la formule : $1 + 2 + 3... + n = \frac{n(n+1)}{2}$

Question 11 : que peut-on écrire de la complexité du tri par sélection ?

- A : Elle est **logarithmique** (**Θ(lg n)**), c'est à dire que la complexité évolue moins vite que le nombre n de données (par exemple : si on multiplie le nombres de données n par 100, le temps d'exécution n'est multiplié que par 8)
- B : Elle est **linéaire** (**Θ(n)**), c'est à dire que la complexité évolue comme le nombre n de données (par exemple : si on multiplie le nombres de données n par 100, le temps d'exécution est multiplié par 100)
- C : Elle est **quadratique** (**$\Theta(n^2)$**), c'est à dire que la complexité évolue comme le carré du nombre n de données (par exemple : si on multiplie le nombres de données n par 100, le temps d'exécution est multiplié par 10000, 100*100)
- D : Elle est **exponentielle** (**$\Theta(x^n)$**), c'est à dire que la complexité évolue à terme beaucoup plus vite que n'importe quelle fonction polynomiale du nombre n de données (par exemple : si on multiplie le nombres de données n par 100, le temps d'exécution est multiplié par 2^{100}, soit 1267650600228229401496703205376)

IV – Preuve de correction (invariant)

Pour prouver que l'algorithme est correct, nous allons devoir trouver **un invariant** : une propriété vérifiable et qui permettra de prouver que le résultat final après exécution est bien le résultat attendu. La démonstration se fait en trois temps.

1 - L'INITIALISATION

On montre que l'**invariant** est vrai **avant le passage** dans une boucle. Si on note la propriété invariante P, on montre que P_0 == VRAI .

2 - LA CONSERVATION

On montre que l'**invariant** reste vrai **après chaque passage** dans une boucle. Cela revient à montrer que si l'invariant est vrai à la boucle k, il reste vrai à la fin de la boucle k+1 : $\mathbf{P_k} \Rightarrow \mathbf{P_{k+1}}$

3 - LA TERMINAISON

On exploite les deux phases précédentes pour montrer qu'**une fois toutes les boucles effectuées**, l'invariant reste vrai et que l'algorithme a bien répondu à son objectif en s'appliquant à toutes les données.

Invariant sur cet algorithme :

1 - L'INITIALISATION

Question 12 : Si on n'est pas encore rentré dans la boucle, combien a-t-on d'éléments dans notre sous-tableau trié ? Peut-on considérer que l'invariant est vérifié ?

2 - LA CONSERVATION

Question 13 : Quelle va être la première valeur prise par **`debut`** ? Que va contenir **`tableau[debut]`** après la première boucle ? Combien d'éléments k triés ? Et après le deuxième tour de boucle ? En déduire la relation entre k et debut

Question 14 : Montrer qu'avec un tour de boucle en plus, P_k vrai implique que P_{k+1} est vrai.

3 - LA TERMINAISON

Après le tour de boucle k, le sous-tableau trié contient _____

A la fin du tour pour lequel debut vaut 0, on a un élément trié : k = 1

A la fin du tour pour lequel debut vaut 1, on a deux éléments triés : k = 2

A la fin du tour pour lequel debut vaut 2, on a trois éléments triés : k = 3

On remarque qu'à la fin de la boucle, on peut écrire la relation suivante **k =** _____

Lors du dernier tour de boucle, la variable de boucle debut vaut _____.

On peut donc en déduire que k vaut _____.

_____

V – Vérification expérimentale de l'invariant avec une assertion

```
 1  def tri_selection(tableau) :
 2      for debut in range(len(tableau)) :
 3          index_min = minimum(tableau, debut)
 4          intervertir(tableau, debut, index_min)
 5          assert est_trie(tableau, debut)
 6
 7  def est_trie(tableau,k) :
 8      '''Renvoie True si le sous-tableau [0;k] est trié dans l'ordre croissant'''
 9      if len(tableau) < 2 :
10          return True
11      for index in range(k) :
12          if tableau[index] > tableau[index+1] :
13              return False
14      return True
```

Question 15 : Ligne 5 : que va-t-il se passer si le sous-tableau [0;k] est bien trié ? est mal trié ?