Fiche d'identité de l' algorithme de **TRI par SÉLECTION (ou *tri par extraction*)**

Principe : on va chercher le plus petit élément de la liste pour le mettre en premier, puis on repart du second élément et on va chercher le plus petit élément de la liste pour le mettre en second, etc...
Exemple : le photographe, qui place les enfants sur le banc en fonction de leurs tailles,passe en revue les enfants à la recherche du plus petit. Une fois trouvé, il le place sur le banc. Il réitère cette opération sur les enfants restant en plaçant à chaque fois l'enfant sélectionné à la prochaine place disponible sur le banc.

Pseudo Code

On écrit une boucle qui s'applique à une liste t de n éléments : elle parcourt les indices j de la liste t allant de 0 à n-2
*on cherche le minimum de la liste t[j:n-1] **// Il faut une fonction « MINIMUM »**
*on permute ce minimum avec le premier élément de la liste t[j : n-1]

```
n ← nombre d'éléments de t
pour i allant de 1 à n-2
    i_min ← minimum(t,j)   // appel de la fonction minimum : renvoi son indice
    si i_min différent de j alors
        permuter t[i] et t[j]
    fin si
fin pour
```

Python

On parle de procédure car il n'y a pas de renvoi (pas de return) : on trie la liste en place

```
def tri_selection(t):
    n=len(t)
    for j in range (n-1):
        indice = minimum_indice(t,j)
        if indice != j:
            t[indice], t[j] = t[j] , t[indice] # permute le minimum et
                                               # le 1er élément de la liste

            # pas de return, on trie en place et on parle de procédure
```

RAPPEL

```
# fonction issue des exercices sur la recherche séquentielle
def minimum_indice(t,j):
    indice, mini=j,t[j]
    n=len(t)
    for i in range(j+1,n):
        if t[i] < mini:
            indice, mini=i,t[i]
    return indice
```

Algorithme valide (CORRECTION & TERMINAISON)
On choisit comme **invariant de boucle** (procédé itératif) H : « la liste t[0 : i+1] est triée par ordre croissant à l'issue de l'itération i »
Initialisation (**est ce vrai avant d'entrer dans la boucle** ?)
si i=0, au rang 0 on a [t[0]] ou encore t[0:0+1] : constituée d'un seul terme, elle est donc triée : H est vraie
Conservation = hérédité (**reste vraie après une itération, si elle était vraie avant**)
A la fin de l'itération i, t[i : i+1]=[t[0],t[1],...,t[i]] est supposée triée dans l'ordre croissant si H est vraie : on effectue un nouveau tour de boucle donc à l'itération i+1.

```
i_min ← minimum(t,j)
si i_min différent de j alors
    permuter t[i+1] et t[j]
```

Alors la liste t[0 : i+2]=[t[0],t[1],...,t[i], t[i+1]] devient triée dans l'ordre croissant. H reste vraie

Terminaison (**donne le résultat attendu en fin de boucle**)

En sortie de boucle, i a pris sa dernière valeur i= n-2 et la boucle permet d'effectuer le tri au rang n-2

```
i_min ← minimum(t,j)  // indice du minimum entre t[n-2] et t[n-1]
si i_min différent de n-1 alors
    permuter t[n-2] et t[n-1]
```

Malgré l'absence de « renvoi »(return en python) le tri est effectué sur la liste t, modifiée à chaque tour de boucle. La liste t est triée dans l'ordre croissant et la fonction *tri* a rempli son objectif

Complexité en temps : on s'intéresse au nombre de comparaisons effectuées. La liste est triée par ordre croissant. La boucle *pour* de la fonction *tri* effectue n-1 opérations. A chaque itération, la fonction *minimum* est appelée et effectue n-(j+1) itérations. Sa complexité est donc en $\Theta(n^2)$ de l'ordre de $n \times n$

On a le tableau suivant

taille	Temps (sec)
1000	0,06
2000	0,13
4000	0,44
8000	1,78
16000	6,79

Formalisation

1°) Faire fonctionner à la main le tri par selection sur un nouveau jeu de 4 cartes

puis sur une liste, [9,3,1,6] par exemple qui pourra être aussi illustrée si besoin par les 4 cartes de valeur correspondantes → Implémenter alors en pseudo code puis en Python cet algorithme

[*9,3,**1**,6*] → [1,*9,**3**,6*] → [1,3,*9,**6***] → [1,3,6,9] (italique = sous liste sur laquelle on travaille, gras : minimum de cette sous liste)

2°)

Quel est le pire des cas ? Combien y-a-il alors d'itérations dans la boucle *pour*? Combien y-a-il d'appel à la fonction minimum dans chaque tour de boucle? Combien d'itérations effectue cette dernière fonction ?

La liste est triée par ordre décroissant : on effectue n-1 itérations (de l'ordre de n) dans la boucle pour du tri. A chaque itération, la fonction minimum est appelée une fois et effectue n-(j+1) itérations (de l'ordre de n opérations) : la complexité est donc quadratique , en $O(n^2)$.

3°) Définir et trouver un *Invariant de boucle*

Pseudo code + code Python + Correction du code + terminaison du code + complexité en temps → voir CORRECTION fiche ID tri Selection

C Exercices

Exercice 1 – *ANALYSER et CONCEVOIR*

On souhaite (ENCORE!) trier la liste de listes *Pers* selon le numéro situé en 2ème position dans les sous-listes que contient *Pers* :

```
Pers =[['Portillon',4],['Sam',3],['Julie',1],['Tom',2],['Charlie',5]]
```

Proposer une fonction mettant en œuvre un algorithme de tri par sélection permettant de trier la liste *Pers*

Exercice 2 – *ANALYSER et TRADUIRE*

On souhaite maintenant trier la liste de listes *Pers* selon le numéro situé en 2ème position dans les sous-listes que contient *Pers* :

```
Pers =[['Sam',3],['Julie',1],['Tom',2],['Portillon',4],['Charlie',5]]
```

Mais cette fois ci, seul un élément est mal trié. Il s'agit de la liste ['Sam',3] qui correspond à l'indice 0 dans la liste *Pers* . Proposer une fonction mettant en œuvre un algorithme de tri conçu par vous même permettant de trier la liste *Pers.*
On ne remplacera que l'élément mal placé, sans trier toute la liste.
(cette situation peut être illustrée par le rangement d'un livre selon son titre dans une bibliothèque dont les ouvrages sont classés par ordre alphabétique : on ne trie pas toute la bibliothèque mais on insère l'ouvrage à sa place)

Aides : *del Pers[0]* supprime l'élément mal placé
Pers.insert(i,mal_placee) insère la liste *mal_placee* (à déterminer) à l'indice *i* (à déterminer)

Exercice 3 – *ANALYSER et TRADUIRE*

En supposant que le tri par sélection prend un temps directement proportionnel à n^2 et qu'il prend 6,8 secondes pour trier 16000 valeurs (voir tableau des temps de ce tri), calculer le temps qu'il faudrait pour trier un million de valeurs

On utlise la proportionnalite : $\frac{6.8\times 1000000^2}{16000^2}\approx 26650\approx 7\,h\,23\,min$ tout de même.....Ce tri n'est pas celui employé pour des milliers de données...heureusement !

Comparer ces temps avec les tris de Python

On pourra générer aléatoirement des listes de 16000 éléments (liste_1) entre 1 et 10 000, puis de 1 million d'éléments (liste_2) toujours entre 1 et 10 0000 et comparer à l'aide du petit programme de comparaison d'écoulement du temps les temps de tri de *liste*.sort() ; <u>un tri en place</u> et sorted(*liste*) qui crée, elle, une copie triée dans Python.

```
from time import time
debut = time()
# Code dont on mesure le temps
fin = time()
print("Temps passé : ",fin - debut)
```

Les résultats sont surprenants de rapidité

Temps passé L1 sorted : 0.003000020980834961

Temps passé L1 sort : 0.003000020980834961

Temps passé L2 sorted : 0.3500199317932129

Temps passé L2 sort : 0.32901859283447266