TP6 TNSI	Algorithmique - 1

L'objectif de ce TP est de réviser les méthodes de base en algorithmique, ainsi que les algorithmes déjà vus en première.

I - Recherche d'un minimum

Cette partie a pour objectif, sur un exemple, de réinvestir les méthodes de base du domaine.

A - Écrire un algorithme

La spécification d'un algorithme est la description rigoureuse de ses entrées et sorties. C'est donc l'ensemble de ses préconditions et de ses postconditions.
Dans ce I, on s'intéresse à l'algorithme Minimum(L) dont la spécification est :

- précondition : l'entrée est une liste L de nombres de longueur n
- postcondition : la sortie est le plus petit de ces nombres

1 - Écrire l'algorithme Minimum(L)

B - Prouver la correction d'un algorithme

Prouver la correction d'un algorithme, c'est montrer qu'il est correct, c'est à dire qu'il respecte sa spécification ou encore, montrer qu'il fait bien ce pour quoi il est conçu.

1 - Trouver une propriété qui est vraie à chaque boucle de l'algorithme Minimum.

2 - Montrer que cette propriété est vraie pour i = 0

3 - Montrer que si cette propriété est vraie pour la ième boucle, elle est vraie pour la i+1 ième.

4 - En déduire la correction de l'algorithme : il fait bien ce pour quoi il est conçu.

C - Complexité

1 - Complexité temporelle

a - Définition

La complexité temporelle est liée au temps d'exécution d'un algorithme. On le caractérise par le nombre d'opérations élémentaires (calculs, affectations, comparaisons) nécessaires en fonction de la longueur de l'entrée (en général notée n).
Cette expression est évaluée dans le pire des cas (complexité temporelle dans le pire des cas) ou en moyenne (complexité temporelle moyenne).

b - Application : Combien d'opérations sont nécessaires dans le pire des cas pour trouver le minimum d'une liste de longueur n ?

2 - Complexité spatiale

a - Définition

C'est l'espace occupé en mémoire par l'algorithme.

b - Application

Comme ici, l'espace occupé en mémoire est constant (n entiers pour la liste + 1 entier pour la variable mini), on parle d'algorithme en espace constant. On note la complexité spatiale : O(1) que l'on lit « grand O de 1 ».

3 - Simulation

a - Compléter le fichier ImplementationComplexite.py afin que la fonction `alea` respecte sa docstring :

```
def alea(i):
        '''In : i un entier, la taille de la liste à générer
        Out : une liste de i entiers aléatoires compris entre 0 et
100'''
```

b - Compléter le même fichier afin que la fonction Minimum soit l'implémentation en python de l'algorithme du minimum proposé au I-1.

c - Utiliser la fonction trace afin d'afficher la durée du calcul en fonction de la longueur de la liste. Retrouvez-vous le résultat du C-1-b ? Justifier votre réponse en utilisant une règle.

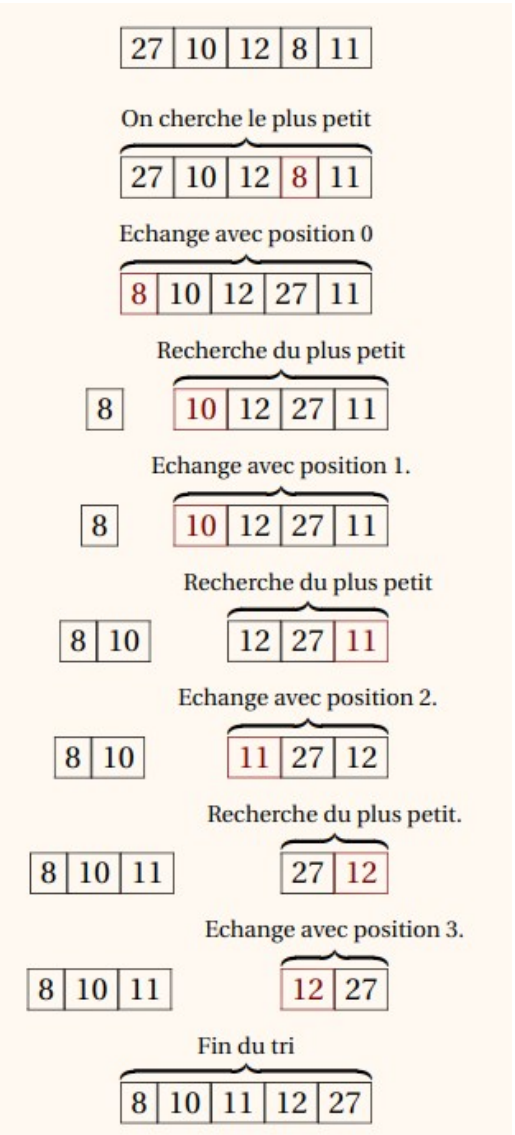
II - Algorithmes de tri

A - Tri par sélection

1 - Description

Dans le tri par sélection, à chaque étape, on sélectionne le plus petit élément de la partie non triée de la liste, puis on le place à la suite de la partie triée.

2 - Exemple (ci-contre)



3 - Caractéristiques

Le tri par sélection est un tri :

- en place : les éléments sont échangés au sein de la liste initiale.
- instable : l'ordre de deux éléments égaux ne sera pas nécessairement respecté après le tri.

4 - Algorithme

ENTRÉE : A est une liste avec une relation d'ordre comme les entiers

tères.

Pour $i = 0$ à $longueur(A) - 1$ **faire**
  $m \leftarrow i$
  Pour $j = i + 1$ à $longueur(A) - 1$ **faire**
    Si $A[j] < A[m]$ **alors**
      $m \leftarrow j$
    Fin Si
  Fin Pour
  Echanger $A[i]$ et $A[m]$
Fin Pour

SORTIE : Une liste triée A

5 - Réalisation

a - Programmer cet algorithme en python dans le fichier ImplementationComplexite (fonction TriSelection).

b - Tests

On vous propose le jeu de test suivant :

- Entrée : [27,10,12,8,11] ; sortie : [8,10,11,12,27]
- Entrée:[-27,-10,-12,-8,-11] ; sortie : [-27,-12,-11,-10,-8]
- Entrée : [5,4,3,3,1,0] ; sortie : [0,1,3,3,4,5]
- Entrée : [-1,-2,0,1,2] ; sortie : [-2,-1,0,1,2]
- Entrée:[2,1.1,1.2] ; sortie:[1.1,1.2,2]

Tester votre algorithme.

Le jeu de test proposé est-il couvrant ?

6 - Complexité

a - Complexité temporelle

Combien d'opérations sont nécessaires dans le pire des cas pour trier une liste de longueur n avec le tri par sélection ?

b - Grâce à la fonction trace, vérifier votre calcul du a avec python.

7 - Complexité spatiale

Quel est l'espace occupé par ce tri ?
En déduire sa complexité spatiale.

8 - Correction

a - Donner un invariant de la boucle sur i de l'algorithme.

b - Prouver cet invariant.

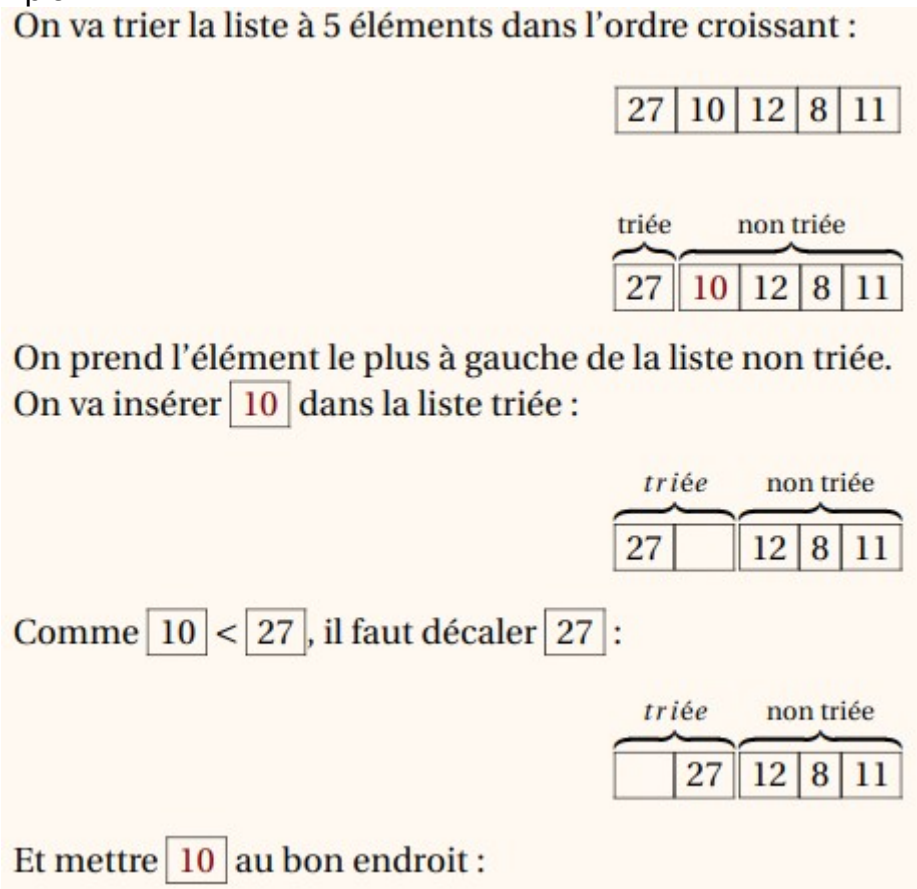
c - En déduire la correction de l'algorithme.

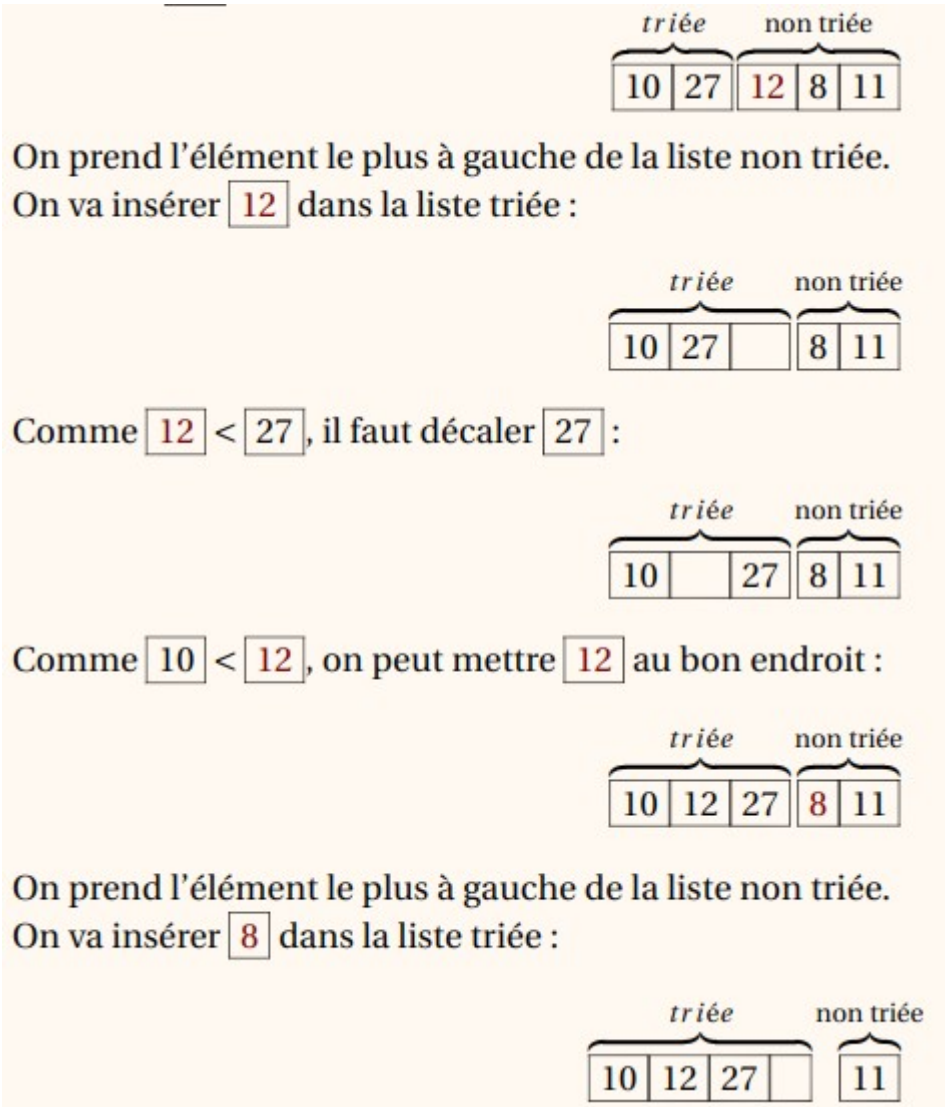
B - Tri par insertion

1 - Description

Dans le tri par insertion, à chaque étape, on insère l'élément nouveau pris en compte à la bonne place de la partie de liste triée précédemment.

2 - Exemple





3 - Caractéristiques

Le tri par sélection est un tri :

- en place : les éléments sont échangés au sein de la liste initiale.
- stable : l'ordre de deux éléments égaux sera respecté après le tri.

4 - Algorithme : Tri par insertion

ENTRÉE : A est une liste avec une relation d'ordre comme les entiers, les flottants ou les chaînes de caractères.

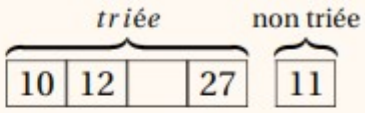
Pour $i = 1$ à $longueur(A) - 1$ **faire**

Inserer $A[i]$ dans la liste $A[0...i[$

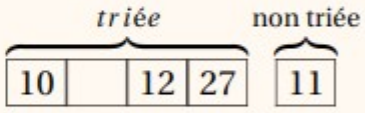
Fin Pour

SORTIE : Une liste triée A

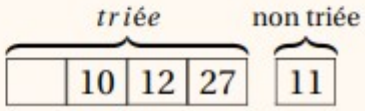
Comme 8 < 27, il faut décaler 27 :



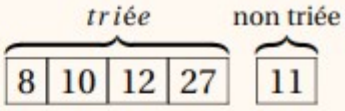
Comme 8 < 12, il faut décaler 12 :



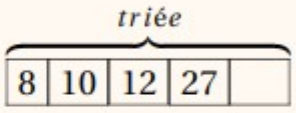
Comme 8 < 10, il faut décaler 10 :

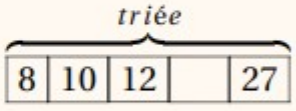


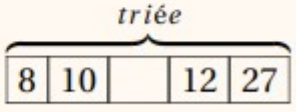
Et mettre 8 au bon endroit :

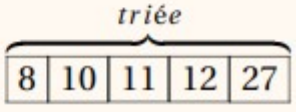


Il reste 11 à insérer :









ATTENTION : toute la finesse du raisonnement est dans l'algorithme d'insertion :

ENTRÉE : A est une liste avec une relation d'ordre comme les entiers, les flottants ou les chaînes de caractères, i est entier qui est l'indice de l'élément à insérer dans A[0...i[

$m = A[i]$

Tant que $i > 0$ et $m < A[i-1]$ **faire**

$A[i] = A[i-1]$

$i \leftarrow i - 1$

Fin Tant que

$A[i] = m$

SORTIE : Une liste triée A

5 - Réalisation

a - Programmer cet algorithme en python dans le fichier ImplementationComplexite (fonction TriInsertion). Ne pas oublier d'écrire la fonction Inserer !

b - Tests

A l’aide du jeu de test précédent (voir la partie Tri par Sélection), tester votre algorithme.

6 - Complexité

a - Complexité temporelle

Combien d’opérations sont nécessaires dans le pire des cas pour trier une liste de longueur n avec le tri par sélection ?

b - Grâce à la fonction trace, vérifier votre calcul du a avec python.

7 - Complexité spatiale

Quel est l’espace occupé par ce tri ?
En déduire sa complexité spatiale.

8 - Correction

a - Donner un invariant de la boucle sur i de l’algorithme.

b - Prouver cet invariant.

c - En déduire la correction de l’algorithme.

9 - Terminaison

La présence d’une boucle non bornée dans la fonction insérer, doit nous amener à prouver que la fonction se termine bien.

a - Donner un variant de la boucle while de la fonction inserer. (Un variant de boucle est une propriété qui va permettre de montrer que l’on se rapproche à chaque boucle de la condition de fin du while).

b - Montrer que l’on a bien un variant.

c - En déduire que l’algorithme se termine bien.

Pour un algorithme dans lequel le nombre des solutions possibles diminue d’un facteur 2 à chaque étape, la complexité est de l’ordre de $\log_2 n$: $O(\log_2 n)$.