Exercice 1

Etude de la fonction `interclassement`

1.1 Etudier le script de cette fonction

1. Quel est le type de chaque paramètre de la fonction ?
2. Quel est le type de chacune des variables internes ?
3. Tester cette fonction à partir de 2 listes passées en argument : [1,4,3,7] et [2,5,8,6]. Que retourne t-elle ?
4. Tester maintenant cette fonction à partir de [1,3,4,7] et [2,5,6,8]. Conclure.
5. Ecrire le prototype de cette fonction à partir des réponses precedentes. (params, variables, returns, examples)

```
def interclassement(L1,L2):
    lN = []
    n1, n2 = len(L1),len(L2)
    i1, i2 = 0,0
    while i1<n1 and i2<n2:
        if L1[i1] <= L2[i2]:
            lN.append(L1[i1])
            i1 += 1
        else:
            lN.append(L2[i2])
            i2 += 1
    return lN + L1[i1:] + L2[i2:]
```

Exercice 2

Algorithmes de tri

On donne la description de 3 algorithmes

2.1 Tri par selection

Sur un tableau de n éléments (numérotés de 0 à n-1), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié (jusqu'au rang n-2).

2.2 Tri insertion

Trier, c'est déplacer des éléments, et y insérer l'élément rangé, depuis le debut déjà trié de la liste, jusqu'à la fin :

- Hyp : l'élément non rangé est le j. Tous les autres éléments sont rangés jusqu'à j.
- Il faut d'abord conserver sa valeur à l'aide d'une variable temp
- On décale tous les éléments i, depuis le rang j jusqu'à l'élément dont la valeur est inférieure à celle de j (et donc de temp), en redescendant.

2.3 tri fusion

L'algorithme est naturellement décrit de façon récursive.

- Si le tableau n'a qu'un élément, il est déjà trié.
- Sinon, séparer le tableau en deux parties à peu près égales.
- Trier récursivement le sous-tableau de gauche avec ce même algorithme du tri
- Trier récursivement le sous-tableau de droite avec ce même algorithme du tri
- Fusionner les deux tableaux triés en un seul tableau trié.

2.4 Questions

1. A l'aide des scripts python suivants : Quelle fonction correspond à :
   - tri par insertion : ...
   - tri par selection : ...
   - tri fusion : ...
2. La complexité de l'algorithme du tri fusion est $O(N \times log(N))$. Pour les tris simples et par selection, c'est $O(N^2)$. Lequel de ces tris est le plus efficace ? Pourquoi ?

```python
def tri1(L):
    for j in range(len(L)):
        temp = L[j]
        i = j
        while i>0 and L[i-1]>temp:
            L[i]=L[i-1]
            i-=1
        L[i]=temp
    return L

def select(T,debut) :
    indiceDuMin=debut
    for k in range(debut+1,len(T)) :
        if T[k]< T[indiceDuMin] :
            indiceDuMin=k
    if indiceDuMin !=debut :
        T[debut],T[indiceDuMin]=T[indiceDuMin],T[debut]


def tri2(T):
    for j in range(0,len(T)-1) :
        select(T,j)
```

```
23      return T
24
25  def tri3(L):
26      if len(L) <=1:
27          return L
28      m = len(L)//2
29      gauche = tri3(L[:m])
30      droite = tri3(L[m:])
31      return interclassement(gauche,droite)
```

Exercice 3

Analyse de l'algorithme du tri par insertion

Utiliser le script du `tri1` (exercice précédent)

3.1 Montrer que pour `j = 1`, à la fin de la boucle principale `for`, la liste est triée jusqu'au rang 1 inclus.

3.2 Supposons qu'à la fin du tour `j-1`, les valeurs sont triées jusqu'au rang `j-1` inclus. Montrer alors que, lors du tour `j`, la case `temp = L[j]` sera insérée correctement et que la liste sera alors triée jusqu'au rang j.

On considèrera 2 cas :

- **Cas 1** : `L[j] >= L[0]` : la situation suivante pourra fournir les explications necessaires :

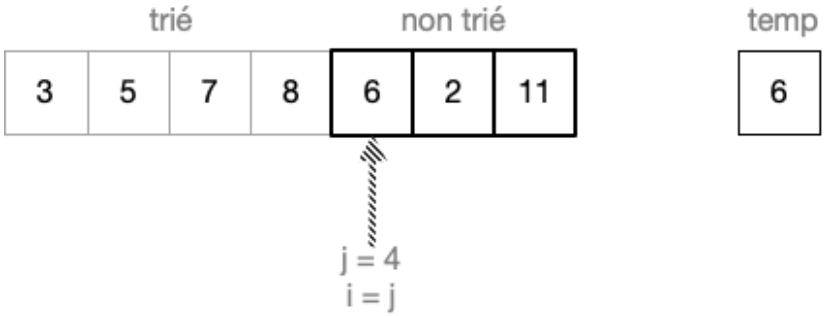

FIGURE 1 – liste triée jusqu'au rang j = 4

- **Cas 2** : `L[j] < L[0]` : prendre l'élement au rang j = 5 de la liste précédente.
- Conclure que la liste sera entièrement triée avec cet algorithme.

3.3 Complexité de l'algorithme de tri par insertion

On va dénombrer le nombre d'*affectations* réalisées pour trier la liste. Puis établir une loi recursive sur `n`.

3.3.1 Avec la liste triée jusqu'au rang `j-1 = 3`, combien d'affectations sont réalisées pour placer correctement la valeur `6` ?

3.3.2 Combien faudra-t-il d'affectations pour placer correctement la valeur `2` ?

3.3.3 On dispose maintenant d'une liste L de dimension `n`, qui est triée en sens inverse. On lui applique l'algorithme de tri par insertion. Il s'agit du *pire des cas* pour cet algorithme : Combien d'affectations sont réalisées pour trier toute la liste ?

3.3.4 Donner la complexité O(g(n)).

Exercice 4

Tri par selection du plus grand élément

La fonction `tri2` est celle du tri par sélection du plus petit élément. Il est possible aussi de faire un tri par *sélection du plus grand élément*. On place alors systématiquement l'élément le plus en debut de liste. Celle-ci apparait alors triée à l'envers, du plus grand au plus petit.

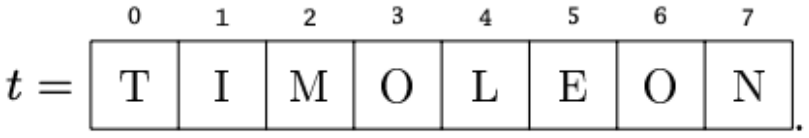

FIGURE 2 – tableau à trier

4.1 Donner les états successifs du tableau à la fin de chaque étape du tri par *sélection du plus grand élément*.

4.2 *Programmation* : Ecrire le script de la fonction de tri par *sélection du plus grand élément*.

Exercice 5

Amélioration du tri fusion

La fonction `tri3` est celle du tri par fusion. Le tri par insertion est plus rapide que le tri fusion lorsque la portion de liste à trier est petite. Modifiez le tri fusion pour qu'il utilise le tri par insertion en dessous d'une certaine taille de liste (moins de 10 éléments, par exemple).