# 4TPM205U: Algorithmique des tableaux: feuille 7

## Algorithmes de tri (1)

### À traiter avant la feuille 6 en 2019-2020

## Travaux dirigés sur machine

*Rappel : le fichier* `bibTableau.py` *doit se trouver dans le même répertoire que vos programmes* Python.

Depuis la page `http://dept-info.labri.fr/ENSEIGNEMENT/algotab/src/` téléchargez le fichier `tp07.py`. Il contient des fonctions utilisées pour les tests. Vous y ajouterez les fonctions demandées.

**Exercice 1 – Tri par sélection : expérimentation**
Lancer l'application `http://inriamecsci.github.io/#!/grains/methodes-tri`

1. Testez le **tri visuel** du tri par sélection (sur cette application l'algorithme cherche le maximun au lieu du minimum).
2. À l'aide de l'onglet **tri temporel** faites des tests sur le temps de calcul pour trier par exemple sur :
   - 100 tableaux de taille 100
   - 100 tableaux de taille 200
   - 100 tableaux de taille 500
   - 500 tableaux de taille 500
   - 1000 tableaux de taille 500
   - 1 tableau de taille 3000
3. Comparez les résultats stockés dans le journal avec les résultats théoriques vus en cours.

**Exercice 2 –**

1. Écrire une fonction `indiceMin`, qui renvoie l'indice d'un élément minimal d'un tableau, entre les indices `i` et `n-1` inclus. Quels seront les paramètres de cette fonction ?
2. Écrire une fonction `echanger` qui prend comme paramètres un tableau d'entiers et les indices (valides) de deux cases dont on veut échanger le contenu.
3. Écrire une fonction `triSelection(t, n)` qui appelle `indiceMin` et `echanger` pour effectuer le tri du tableau `t` contenant `n` nombres.
4. Tester votre fonction en utilisant la fonction de test `testsTriSelection` fournie dans `tp07.py`.

### Exercice 3 – Tri par insertion : expérimentation

Lancer l'application `http://inriamecsci.github.io/#!/grains/methodes-tri`

1. Testez le **tri visuel** du tri par insertion.
2. À l'aide de l'onglet **tri temporel** faites des tests sur le temps de calcul pour trier par exemple sur :
   - 100 tableaux de taille 100
   - 100 tableaux de taille 200
   - 100 tableaux de taille 500
   - 500 tableaux de taille 500
   - 1000 tableaux de taille 500
   - 1 tableau de taille 3000
3. Comparez les résultats stockés dans le journal avec les résultats théoriques vus en cours et en TD.

### Exercice 4 – Tri par insertion

1. Écrire une fonction `triInsertion(t, n)` qui met en œuvre l'algorithme de tri par insertion sur un tableau `t` contenant `n` nombres.
2. Tester votre fonction sur un tableau trié dans l'ordre décroissant. Vous vous rappelez du nombre de décalages nécessaires pour le trier dans l'ordre croissant ?

**Exercice 5** – En appelant la fonction `testTris` fournie dans le fichier `tp07.py`, remplir le tableau suivant qui recense le temps d'exécution (en millisecondes) sur le même tableau de vos fonctions de *tri par sélection* et *tri par insertion*.

Tableau aléatoire

| nombre d'élements | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| tri par sélection | | | | |
| tri par insertion | | | | |

Tableau trié croissant

| nombre d'élements | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| tri par sélection | | | | |
| tri par insertion | | | | |

Tableau trié décroissant

| nombre d'élements | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| tri par sélection | | | | |
| tri par insertion | | | | |

### Exercice complémentaire 1

1. Écrire la fonction `triBulle` étudiée en cours.
2. Pour améliorer le tri à bulle, on peut faire en sorte de limiter les parcours sans échanges. Si pendant un parcours du tableau aucun échange ne se produit, que peut-on en déduire ? Écrire une fonction `triBulleAmeliore(t, n)` utilisant cette remarque (on pourra utiliser un booléen `continuer`).

3. Pour améliorer le tri à bulle, on peut faire en sorte de limiter les parcours. On peut mémoriser le plus grand indice à partir duquel les échanges ne se font plus, c'est à dire l'indice après lequel la liste est triée. Écrire une fonction `triBulleLimite(t,n)` qui améliore la version précédente en utilisant cette remarque.
4. Pour améliorer le tri à bulle, on peut encore faire la modification suivante : alterner les sens des parcours consécutifs afin de traiter le cas où les éléments mal placés se trouvent en début de tableau. Par exemple, dans un tri à bulle, le tableau qui contient $5, 7, 9, 3$ est trié en trois passes, alors que le tableau contenant $9, 3, 5, 7$ est trié en une passe. Utiliser les trois améliorations précédentes pour écrire une fonction `triShacker(t,n)` améliorant le tri à bulle.

**Exercice complémentaire 2**
Dans le fichier `tp07.py` modifier la fonction `affichageAppelsTris` en ajoutant les instructions permettant d'afficher les résultats des appels des différentes versions du tri à bulles.