TP - Algorithmes de tri

M. Tellene

1 Création de l'environnement de travail

Avant de commencer le TP, vous allez créer votre environnement de travail. Pour ce faire vous allez dans votre dossier personnel. Une fois arrivé, créer un dossier « TP_tris ». C'est dans ce dossier que vous sauvegarderez les exercices de ce TP.

Vous allez commencer créer un fichier `tris.py` et écrire une fonction `est_trie()`. Cette fonction prend en argument un tableau et renvoie `True` si le tableau est trié, `False` sinon.

En plus de cette fonction `est_trie()`, vous écrirez une fonction `nb_occ()` qui prend en argument un tableau et renvoie le dictionnaire des occurrences correspondant.

Ces deux fonctions **devront** être utilisées afin de tester vos algorithmes de tri.

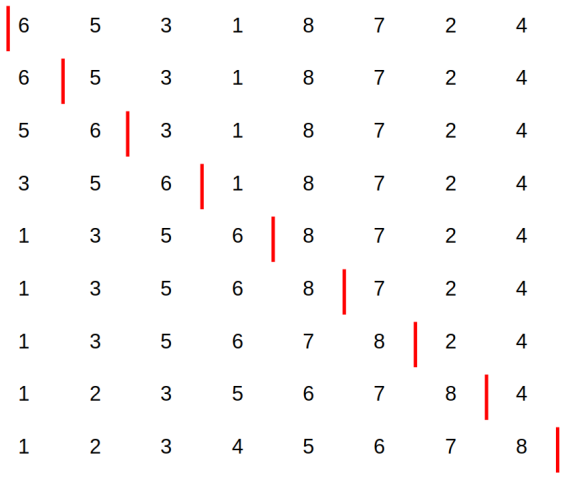
2 Programmation des méthodes

2.1 Tri insertion

Le tri par insertion considère chaque élément de la liste et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.

Pour trouver la place où insérer un élément parmi les précédents, il faut le comparer à ces derniers, et les décaler afin de libérer une place où effectuer l'insertion. Le décalage occupe la place laissée libre par l'élément considéré. En pratique, ces deux actions s'effectuent en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Exemple du tri insertion :



Pour vous aider, il vous est conseillé d'écrire une fonction `permutation()` qui prend en argument un tableau et deux indices (`x` et `y` par exemple). La fonction renvoie le tableau où les éléments aux indices `x` et `y` ont été permutés.

```
>>> L = [9, 7, 2, 5, 10, 1, 6, 4, 3, 8]
>>> permutation(L, 0, 5)
[1, 7, 2, 5, 10, 9, 6, 4, 3, 8]
```

Une fois fait, écrire une fonction `insertion()` qui prend en argument un tableau et un indice (`x` par exemple). La fonction doit renvoyer le tableau où l'élément à l'indice `x` est insérer au bon endroit. Dans l'exemple ci-dessous, les étapes intermédiaires ont été affichées.

```
>>> L = [9, 7, 2, 5, 10, 1, 6, 4, 3, 8]
>>> insertion(L, 5)
[9, 7, 2, 5, 1, 10, 6, 4, 3, 8]
[9, 7, 2, 1, 5, 10, 6, 4, 3, 8]
[9, 7, 1, 2, 5, 10, 6, 4, 3, 8]
[9, 1, 7, 2, 5, 10, 6, 4, 3, 8]
[1, 9, 7, 2, 5, 10, 6, 4, 3, 8]
[1, 9, 7, 2, 5, 10, 6, 4, 3, 8] #c'est le renvoi
```

Enfin, écrire une fonction `tri_insertion()` qui prend en argument un tableau. La fonction doit renvoyer le tableau trié en suivant la méthode du tri insertion.

```
>>> L = [9, 7, 2, 5, 10, 1, 6, 4, 3, 8]
>>> tri_insertion(L)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2.2 Pour tester vos fonctions

Vous trouvez un fichier `generation_tableau.py`. Ce fichier contient une fonction `generateur(n)` qui génère 3 tableaux de n éléments :

- un tableau trié par ordre croissant
- un tableau trié par ordre décroissant
- un tableau trié aléatoirement

2.3 Tri sélection

Sur une liste `L` de n éléments (numérotés de 0 à $n-1$), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément parmi `L[0;n-1]`, et l'échanger avec l'élément `L[0]` ;
- rechercher le plus petit élément parmi `L[1;n-1]`, et l'échanger avec l'élément `L[1]` ;
- continuer de cette façon jusqu'à ce que la liste soit entièrement triée.

Exemple du tri sélection :

6	5	3	1	8	7	2	4
1	5	3	6	8	7	2	4
1	2	3	6	8	7	5	4
1	2	3	6	8	7	5	4
1	2	3	4	8	7	5	6
1	2	3	4	5	7	8	6
1	2	3	4	5	6	8	7
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Pour vous aider, il vous est conseillé d'écrire une fonction `minimum()` qui prend en argument un tableau et un indice `x`. La fonction renvoie l'indice de l'élément de valeur minimale du tableau à partir de l'indice `x`.

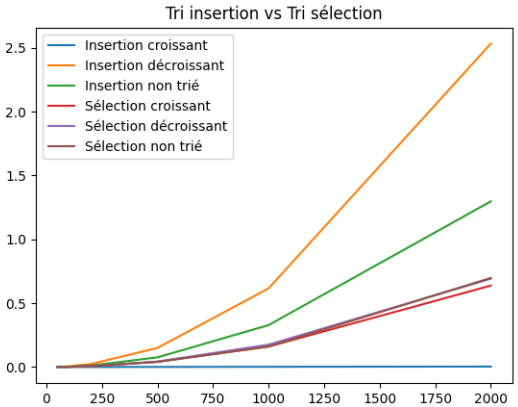
```
>>> L = [9, 7, 2, 5, 10, 1, 6, 4, 3, 8]
>>> minimum(L, 6)
8
```

En vous servant de `minimum` et possiblement de `permutation`, écrire une fonction `tri_selection()` qui prend en argument un tableau. La fonction doit renvoyer le tableau trié en suivant la méthode du tri insertion.

```
>>> L = [9, 7, 2, 5, 10, 1, 6, 4, 3, 8]
>>> tri_selection(L)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3 Comparaison de performances

Le but de cette partie est de comparer les deux méthodes via un graphique. Ajouter le contenu du fichier `performances.py`. Compléter les « ... » afin de rendre le code fonctionnel. Le résultat devrait être (approximativement) le graphique ci-contre.



Dans un second temps, nous allons afficher seulement les courbes des deux méthodes l'une après l'autre. Commencer par afficher les trois courbes du tri insertion, puis les trois courbes du tri sélection.

Enfin, nous allons comparer les méthodes en fonction du tableau donné. Afficher d'abord les courbes des tableaux croissants, puis décroissants et enfin aléatoires.