TP n° 10 : Tris

Le tri est peut-être le problème le plus fondamental en matière d'algorithmique. Donald KNUTH y consacre d'ailleurs le 3e volume de son immense œuvre *The Art of Computer Programming*. Trier est un problème aux applications extrêmement nombreuses et il n'est pas rare de l'utiliser comme sous-routine dans de nombreux algorithmes plus complexes. Dans ce TP, on étudie trois tris de complexité en $\Theta(n^2)$. Vous étudierez en deuxième année ou en option informatique des tris en $\Theta(n \log n)$, ce qui constitue une borne inférieure pour le problème du tri de n objets en utilisant uniquement des comparaisons. Vous pouvez rapidement visualiser la différence entre ces algorithmes de tri.

✔ On triera toutes les données dans l'ordre croissant.

⚠ On annotera toutes les boucles avec un invariant permettant de montrer la correction de la fonction et, pour les boucles `while`, avec un variant de boucle.

1 Comparaisons

En PYTHON, on peut comparer et donc trier des nombres, mais également de très nombreux objets. PYTHON utilise l'ordre lexicographique pour les types composés.

PYTHON

```
In [1]: "Python" < "python"
Out[1]: True

In [2]: False < True
Out[2]: True

In [3]: (1, 2) < (2, 1)
Out[3]: True

In [4]: [1, 5, 3, 7, 2] < [1, 5, 3, 6, 42, 1000]
Out[4]: False
```

2 Le tri en PYTHON

Pour trier une liste en PYTHON, rien de plus simple :

PYTHON
```
In [1]: notes = [10.5, 12, 20, 7.5, 18, 3.5]

In [2]: sorted(notes)
Out[2]: [3.5, 7.5, 10.5, 12, 18, 20]

In [3]: notes.sort()

In [4]: notes
Out[4]: [3.5, 7.5, 10.5, 12, 18, 20]
```

Attention, la fonction `sorted` renvoie une nouvelle liste triée alors que la méthode `.sort()` modifie la liste en la triant (et ne renvoie rien). PYTHON utilise un tri appelé *Tim sort* dont la complexité dans le pire cas est en $\Theta(n \log n)$.

Le but de ce TP est d'implémenter nous-mêmes nos propres tris (qui sont cependant moins efficaces). Il faut savoir le faire. Au concours, il faut toujours bien se poser la question avant d'utiliser directement une de ces deux fonctions : est-ce que leur utilisation ne simplifie pas trop la question ? Est-ce que l'on n'attend pas une solution sans tri ? Est-ce que l'on ne demande pas implicitement de les réimplémenter ? Est-ce que l'on a bien pris en compte la complexité de ces fonctions ?

3 Préliminaires

1. Écrire une fonction `echange(tab, i, j)` effectuant l'échange sur place de `tab[i]` et `tab[j]`, en supposant que le tableau `tab` est une liste PYTHON et que les indices `i` et `j` sont valables, c'est-à-dire positifs et au plus égaux à `len(tab) - 1`. On ne demande pas de le vérifier.

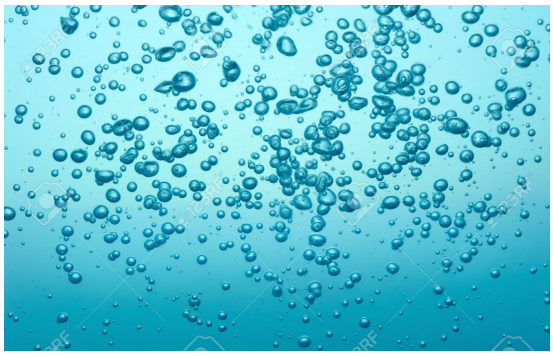
Cette fonction est à réutiliser dans tous les algorithmes de tri.

4 Le tri par sélection

L'idée du tri par sélection est toute simple : on cherche le minimum du tableau, que l'on va mettre à sa place en première position en l'échangeant avec le premier élément ; puis on cherche le minimum du tableau en excluant le premier élément que l'on vient placer à la deuxième position par échange ; et on recommence ainsi de suite avec tous les éléments.

2. Décrire sur feuille les étapes de l'algorithme pour le tri du tableau `[15, 1, 12, 3, 19]`.
3. Écrire une fonction `indice_min(tab ,i)` renvoyant l'indice dans le tableau d'un élément minimal du tableau `tab[i:]`. On suppose que l'indice `i` est valable. Par exemple, pour `tab = [15, 1, 12, 3, 19]` tout comme pour `tab = [3, 1, 12, 3, 19]` , `indice_min(tab, 2)` doit renvoyer `3`.
4. Écrire une fonction `tri_selection(tab)` qui trie un tableau sur place en implémentant l'idée décrite ci-dessus.
5. Estimer le nombre de comparaisons nécessaires dans le meilleur et dans le pire des cas en fonction de la taille du tableau. Quelle est la complexité de la fonction ?

5 Le tri à bulles



L'idée du tri à bulles est, à chaque passage, d'échanger successivement les éléments consécutifs qui ne sont pas dans le bon ordre. On effectue des passages jusqu'à ce qu'il n'y ait plus de changement : lorsque c'est le cas, la liste est bien triée. Vous pouvez visualiser ici ce tri.

6. Décrire sur feuille les étapes de l'algorithme pour le tri de la liste `[15, 1, 12, 3, 19]`. Par exemple, après le premier passage on aura modifié la liste en `[1, 12, 3, 15, 19]`.
7. Justifier qu'après le tout premier passage, le maximum du tableau se trouve en dernière position.
8. Écrire une fonction `passage(tab, fin)` effectuant un passage dans le tableau jusqu'à l'indice `fin` non compris en échangeant chaque élément avec le suivant lorsque c'est nécessaire, et renvoyant `True` s'il y a eu au moins un échange et `False` sinon. Par exemple, pour `[15, 1, 12, 3, 19]` et `fin = 3`, on doit obtenir `[1, 12, 15, 3, 19]` et renvoyer `True`.
9. Écrire une fonction `tri_bulle(tab)` triant un tableau en utilisant un tri à bulles.
10. Déterminer le nombre de comparaisons nécessaires dans le meilleur et le pire des cas en fonction de la taille n du tableau [1] et donner la complexité de la fonction.

6 Le tri par insertion

Le principe du tri par insertion est celui que l'on utilise pour trier un jeu de cartes en main : on insère successivement à la bonne place un élément parmi ceux déjà triés. Vous pouvez visualiser ici ce tri.

11. Décrire les étapes de l'algorithme pour le tri du tableau `[15, 1, 12, 3, 19]`.
12. Écrire une fonction `insere(tab, i)` qui insère l'élément d'indice `i` du tableau `tab` à la bonne place parmi les i premiers éléments qui sont supposés déjà triés, en faisant des échanges successifs.
13. Écrire un algorithme `tri_insertion(tab)` qui implémente ce tri.
14. Déterminer le nombre de comparaisons nécessaires dans le meilleur et le pire des cas en fonction de la taille n du tableau [2] et donner la complexité de la fonction.

Contrairement aux deux tris précédents qui n'ont aucune utilité pratique, ce tri, bien qu'en $\Theta(n^2)$ mais avec une constante cachée assez faible, est l'un des plus efficaces pour des petits tableaux et est souvent utilisé pour de petites entrées ou en combinaison avec d'autres algorithmes de tri plus efficaces pour les grandes instances.

1. On peut montrer qu'il y en a $\frac{n(n-1)}{4}$ en moyenne.
2. On peut montrer qu'il y en a $\frac{n^2}{4}$ en moyenne.