Algorithmes de tri

On voit dans ce cours 2 algorithmes permettant de trier une liste.
Le tri est une opération indispensable dans beaucoup de domaines tels les statistiques, la gestion des bases de données ...
Pour une liste en Python, la méthode *sort()* existe et est très performante. Cependant comprendre comment s'effectue un tri est un exercice pertinent car les contraintes de résolution se retrouvent dans beaucoup de problèmes.
De plus, les méthodes de résolution de tri sont souvent très différentes et peuvent avoir des coûts tout aussi différents.
Les deux méthodes au programme sont le **tri par sélection** et le **tri par insertion**.
On prendra dans la suite les tris par ordre croissant.
Pour avoir une représentation concrète des tris les plus courants ainsi que les implémentations dans différents langages :
http ://lwh.free.fr/pages/algo/tri/tri.htm

1 Tri par sélection

a Principe

Ce tri est facile à comprendre : on prend le plus petit élément de la liste et on l'échange avec le premier élément de la liste, on prend ensuite l'élément le plus petit dans le reste de la liste et on l'échange avec le deuxième élément et ainsi de suite jusqu'à arriver au dernier élément de la liste qui sera le plus grand.
On doit se servir d'une fonction permettant d'échanger deux éléments d'une liste listA :

```
def echange(listA,i,j):
    temp=listA[i]
    listA[i]=listA[j]
    listA[j]=temp
```

b Algorithme

On va ainsi réaliser une boucle en prenant tous les éléments de la liste consécutivement, en effectuant à chaque itération le remplacement de l'élément correspondant à cette itération par le plus petit élément présent dans le reste de la liste, donc pour cela aussi une boucle.

Langage naturel
Pour i allant de 0 à longueur de la liste listA -1 faire
min=i
Pour j allant de i+1 à longueur de la liste listA -1 faire
Si listA[j]<listA[m] alors m=j
Fin Pour
Echanger les éléments d'indice i et m
Fin Pour

Langage Python

```
for i in range(len(listA)):
    min=i
    for j in range(i+1,len(listA)):
        if listA[j]<listA[min]:
            min=j
    echange(listA,i,min)
```

c Invariant de boucle

Il s'agit de déterminer une propriété vraie avant et après chaque itération dans la boucle. Il y a deux boucles imbriquées : il s'agit donc de trouver cette propriété suivant la boucle principale, la deuxième permettant de déterminer l'indice du plus petit élément d'une partie de la liste (déjà fait dans le précédent chapitre sur la recherche d'un extremum).
Pour une itération donnée, au rang i, on a la situation suivante avant passage dans la boucle :

0	i
Partie triée	Partie non triée

Après passage, on se retrouve avec cette situation :

0	i+1
Partie triée	Partie non triée

On peut décrire cette situation par : ***Les i premiers éléments de la liste sont triés, les suivants étant constitués d'éléments plus grands.***
Il s'agit de l'invariant de boucle pour cette algorithme, justifiant la terminaison de l'algorithme : quand i prend la dernière valeur len(listA)-1, on se retrouve avec une partie gauche triée contenant tous les éléments de la liste (la partie droite devenue vide), ce qui est le résultat attendu.

d Coût de l'algorithme

Dans cette algorithme, les manipulations de données se font essentiellement par des comparaisons (on a à effectuer aussi des échanges de valeurs mais on verra que leur nombre est négligeable) : on retiendra donc ce nombre de comparaisons pour évaluer le coût d'un algorithme de tri.
Notons n le nombre d'éléments de la liste.
Pour le tri par selection,
à l'indice 0, on effectue $n-1$ comparaisons
à l'indice 1, on effectue $n-2$ comparaisons
à l'indice 2, on effectue $n-3$ comparaisons
...
à l'indice $n-2$, on effectue 1 comparaison
à l'indice $n-1$, on effectue 0 comparaison
Au total, on a donc $n-1+n-2+n-3+...+1+0=\dfrac{n(n-1)}{2}=\dfrac{n^2}{2}-\dfrac{n}{2}$
On a obtenu une expression du second degré : on dit que le coût est **quadratique**.
On aura par exemple besoin de faire 49995000 comparaisons pour trier une liste de 10000 éléments.

2 Tri par insertion

a Principe

Ce tri est assez similaire au précédent au sens où l'invariant de boucle est quasi identique et le principe repose aussi sur des échanges d'éléments consécutifs à des comparaison.
Le tri par insertion est celui dit des joueurs de cartes :
on prend les deux premiers éléments, on place correctement le deuxième c'est-à-dire on l'échange avec le premier si celui-ci est plus grand ;
on prend ensuite le troisième élément et on le place de manière que les trois premiers éléments soient ordonnés ;
on continue le processus jusqu'au dernier : on l'insère au bon endroit dans la partie ordonnée (en décalant les éléments qui suiventpour garder l'ordre).

b Algorithme

On effectue une boucle pour prendre en considération chaque élément de la liste et ensuite une boucle pour comparer cet élément à chaque élément de la partie déjà triée (avant cet élément) : on pourra donc utiliser une boucle non bornée *Tant Que* afin de diminuer éventuellement le nombre de comparaisons.
Langage naturel
Pour i allant de 1 à longueur de la liste listA -1 faire
element=listA[i]
j=i
Tant que j>0 et element<listA[j-1] Faire
listA[j]=listA[j-1]
j=j-1
Fin Tant Que
listA[j]=element
Fin Pour
Langage Python

```
for i in range(1,len(listA)):
    element=listA[i]
    j=i
    while j>0 and element<listA[j-1]:
        listA[j]=listA[j-1]
        j=j-1
    listA[j]=element
```

c Invariant de boucle

Il y a aussi deux boucles imbriquées : il s'agit donc de trouver cette propriété suivant la boucle principale, la deuxième permettant d'insérer un élément à sa place dans une sous-liste triée.
Pour une itération donnée, au rang i, on a la situation suivante avant passage dans la boucle :

listA[0 :i] est composé des i premiers éléments de listA triés

Après passage, on se retrouve avec i+1 éléments puisque l'on a inséré le iéme élément de manière ordonné parmi les i autres déjà présent :

listA[0 :i+1] est composé des i+1 premiers éléments de listA triés

On peut décrire cette situation par : ***Les i premiers éléments de la liste sont triés***
Il s'agit de l'invariant de boucle pour cette algorithme, justifiant la terminaison de l'algorithme : quand i prend la dernière valeur len(listA)-1, on se retrouve avec une liste complètement triée, ce qui est le résultat attendu.

d Coût de l'algorithme

Dans cette algorithme, les manipulations de données se font essentiellement par des comparaisons présents dans la deuxième boucle (où il y a deux comparaisons) : on retiendra donc ce nombre de comparaisons pour évaluer le coût de l'algorithme.
Dans la boucle *Tant Que*, on a plusieurs possibilités : au mieux on effectue que deux comparaisons (l'élément au rang i est plus grand que les autres), au pire il est inférieur à tous les autres et on effectue $2i$ comparaisons, et on a toutes les autres cas intermédiaires possibles suivant la composition de la liste à trier : le coût n'est pas fixe.
Notons n le nombre d'éléments de la liste.
La boucle principale commence à l'indice 1.
Pour le tri par insertion,
à l'indice 1, on effectue 2 comparaisons
à l'indice 2, on effectue 2 ou 4 comparaisons
à l'indice 3, on effectue 2, 4 ou 6 comparaisons
...
à l'indice $n-2$, on effectue 2 à $2(n-2)$ comparaisons
à l'indice $n-1$, on effectue 2 à $2(n-1)$ comparaisons
Au total, on a donc :
au mieux $2(n-1) = 2n-2$ comparaisons
au pire $2\times 1+2\times 2+2\times 3+...+2(n-2)+2(n-1) = 2(1+2+3+...+(n-2)+(n-1)) = n(n-1) = n^2-n$ comparaisons.
Le coût n'est pas fixe.
Dans le meilleur des cas, il est **linéaire**, mais cela signifie que la liste à trier est ... déjà trié.
Dans le pire des cas, il est **quadratique**, et ce pire des cas a lieu lorsque la liste est ... triée par ordre décroissant.
Pour tous les cas, on pourrait estimer le coût moyen en faisant une moyenne des deux formules obtenues, on obtient :
$\frac{n^2}{2}+\frac{n}{2}-1$
soit un coût **quadratique**.

Le coût de ces deux algorithmes sont globalement similaires : d'autres algorithmes comme le tri rapide ont un coût moindre que quadratique (et plus que linéaire). Pour de grandes listes (par exemple dans les bases de données), avoir le meilleur (donc le moindre) coût est indispensable pour la réalisation des processus.