Les tris

Les algorithmes de tris sont fondamentaux. Il en existe plusieurs, ayant tous la même finalité : Trier un tableau de nombre. Nous allons étudier deux de ces tris cette année. Le tri sélection et le tri par insertion. L'idée est de comparer leur efficacité On étudiera leur complexité et la preuve de correction.

1. Le tri sélection

Le principe de tri sélection est le suivant :

A la première étape :

A partir d'un tableau non trié, il faut trouver le plus petit élément du tableau et le placer en première position. On compare donc le deuxième élément au premier. S'il est plus petit, on les permute, sinon, on passe au troisième élément, etc…

A la deuxième étape :

On parcourt les n-1 derniers éléments du tableau, on compare les éléments au premier élément de ces n-1 et on les permute si on trouve un élément plus petit que celui du début du tableau…

Etc…

Exemple

Voici un tableau non trié : [5,2,6,3,8,1]

Etape 1

[5,2,6,3,8,1]	On compare le 5 et le 2 : on les permute
[2,5,6,3,8,1]	On compare le 6 et le 2.Rien
[2,5,6,3,8,1]	On compare le 3 et le 2.Rien
[2,5,6,3,8,1]	On compare le 8 et le 2.Rien
[1,5,6,3,8,2]	On compare le 1 et le 2. On les permute

A la fin de l'étape 1, on a bien le plus petit élément en première position.

Ecriture de l'algorithme du tri sélection

Algorithme TriSelection (T)

Entrée :

Sortie :

Pour ……

Retourner T

1. Combien y aura-t-il de comparaisons à la deuxième étape ? A la troisième ?
2. Supposons que la liste a n éléments. Combien y a-t-il de comparaisons à la première étape ? A la deuxième ? A la p^{ème} ?

Remarque : Une autre façon de considérer le tri selection : On cherche le plus petit élément de la partie de la liste restant à trier et on le permute, ou non, avec le premier. Cela réduit le nombre de permutation. L'ordre du coût ne changera pas.

On rappelle 😊 la formule de la somme des n premiers entiers naturels non nuls $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

3. Pouvez-vous donner une idée du coût d'un tel tri en vous appuyant sur le cours de la complexité ?

Preuve de correction

Recherche de l'invariant de boucle :

La question à se poser est : Qu'est ce qui est constant à chaque étape de cet algorithme ? Il y a deux éléments à noter :

- Au début de la $i^{ème}$ étape , les i-1 premiers éléments du tableau sont triés par ordre croissant.
- Au début de la $i^{ème}$ étape , les éléments de rang supérieurs ou égal à i sont tous supérieurs au i-1 ^{ème} élément.

Cet invariant de boucle nous prouve la correction de cet algorithme.

Terminaison

On a deux boucles for qui sont imbriquées qui forcément se termine , l'une ayant n-1 tours et l'autre au maximum n-1, puis n-2…jusqu'à 1.

La terminaison est clairement assurée.

2. Le tri insertion

Le principe de ce tri est le suivant :

- On compare les deux premiers éléments, on les met dans l'ordre.
- On prend le troisième élément, on l'insère au bon endroit par rapport aux deux premiers pour avoir les trois nombres classés par ordre croissant.
- On prend le quatrième élément, on l'insère au bon endroit par rapport aux trois premiers

Exemple

Voici un tableau non trié : [5,2,6,3,8,1]

Ecrire la trace permettant de trier ce tableau avec la méthode donnée.

Etat initial	Etat final
[5,2,6,3,8,1]	
	[1,2,3,5,6,8]

Ecriture de l'algorithme du tri insertion

Algorithme TriInsertion (T)

Entrée :

Sortie :

Pour ……

Retourner T

<u>Complexité</u>

Dans le meilleur des cas, il n'y a qu'une comparaison à chaque étape (il y en a n-1) , on sera en O(n).

Dans le pire des cas, il y a pour le $k^{ème}$ élément k-1 comparaisons à faire, ce qui nous donne

$$\sum_{k=1}^{n-1} k = \frac{n*(n-1)}{2}$$

Soit $O(n^2)$.

Il est évident que l'énorme majorité des cas , on sera en $O(n^2)$.

<u>Preuve de correction :</u>

Terminaison

La boucle *for* ne pose pas de problème. Pour la boucle *while*, on part de i qui est égal à un entier naturel et l'on va au maximum jusqu'à 0, en décrémentant i à chaque étape, ce qui assure la terminaison.

Correction

On a deux boucles, il nous faut donc deux invariants.

- **Boucle for** : A la fin de la i^{ème} itération, les i premiers éléments du tableau sont classés ou encore

$$\forall\, \boldsymbol{k} \in [\boldsymbol{1}, \boldsymbol{i}], \boldsymbol{t}[\boldsymbol{k}-\boldsymbol{1}] \leq \boldsymbol{t}[\boldsymbol{k}]$$

- **Boucle while :** A la fin de la j^{ème} étape, la valeur que l'on cherche à insérer est plus petite que toutes les valeurs de l'indice i+1-j à l'indice i ou

$$\forall\, \boldsymbol{k} \in [\boldsymbol{i}+\boldsymbol{1}-\boldsymbol{j}, \boldsymbol{i}], \boldsymbol{valeur} \leq \boldsymbol{t}[\boldsymbol{k}]$$, **k≥1**, le i étant celui utilisé dans la boucle for.

Annexe : Programmes utilisés

Pour le tri insertion

```
def inserebis(t,n):
    i,ins =n-1, t[n]
    while i>=0 and t[i]>ins :
        t[i+1]=t[i]
        i=i-1
    t[i+1]=ins
    return t
```

```
def TriInsertion(t)
    for i in range (len(t)-1):
          inserebis(t,i+1)
    return t
```

Pour le tri selection

```
def TriSelection(t):
     for i in range(len(t)-1):

         for j in range(i+1,len(t)):
             if t[j]<t[i]:

                 t[i],t[j]=t[j],t[i]

    return t
```

```
def TriSelectionbis(t):
    for i in range(len(t)-1):
        mini=i
        for j in range(i+1,len(t)):
            if t[j]<t[mini]:
                mini=j
        t[i],t[mini]=t[mini],t[i]

    return t
```