❄ Chapitre 8 ❄

Tri de listes

I. Pourquoi trier un tableau ?

.....

.....

.....

.....

Exemple 1:

Remarque :

Dans la leçon nous utiliserons en permanence des algorithmes sur des tableaux de nombres entiers. En les modifiant légèrement, ces algorithmes de tri pourront être utiliser pour trier d'autre objets que des nombres entiers. Pour cela il faudra qu'il existe une relation d'ordre totale sur ces objets, c'est a dire qu'on doit être capable de les comparer (dire qui est le plus grand, qui est le plus petit).

II. Le tri par sélection

1. Principe général

Méthode 1 : *Tri par sélection*

1. On cherche le minimum dans la liste.
2. On échange ce minimum avec le premier élément de la liste.
3. On recommence avec le reste de la liste, jusqu'au dernier élément.

Exemple 2:

En suivant le protocole détailler ci contre, ordonnons la suite de nombre suivante :

6	4	8	3	→					→					→				

2. Algorithme de tri par sélection

Entrée : .....

Pré-condition : .....

.....

.....

Sortie : .....

Post-condition : .....

.....

.....

.....

Algorithme en langage naturel :

```
fonction tri_ selection(liste L) :
    n ← taille de L
    pour i allant de 0 à n-1
        min ← i
        pour j allant de i+1 à n
            si L[j] < L[min] alors
                min ← j
            fin si
        fin pour
        si min ≠ i alors
            échanger L[i] et L[min]
        fin si
    fin pour
    Afficher L
```

Définition 1: *Invariant*

Un invariant de boucle est un propriété qui est vraie avant et après chaque répétition.

Exemple 3:

Ici, après le premier passage dans la boucle, le plus petit élément se retrouve devant.
Après deux passages les deux plus petits éléments se retrouvent devant dans l'ordre croissant.
Après p passage, les p plus petits éléments se retrouvent triés dans les p premières cases.

Définition 2: *Cout d'un algorithme*

La complexité d'un algorithme est le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuées par un algorithme. Ce nombre s'exprime en fonction de la taille n des données.

Exemple 4:

Essayons de voir ce qui se passe pour $n = 4$.

1 affectation.
La première boucle For va donc s'exécuter 4 fois ($n - 1 = 4 - 1 = 3$)
Cette boucle comporte au maximum 3 opérations (Affectation, test et échange) + une deuxième boucle.
La deuxième boucle sera elle effectué 2 fois ($n - (i + 1) = 4 - (1 + 1) = 2$), puis 1 fois.
Cette boucle comporte au maximum 2 opérations (Test et affectation)

Donc si on fait le calcul, cet algorithme effectue au maximum $1 + 4 \times (3 + (2 \times 2 + 1 \times 2)) = 29$ opérations pour une liste de longueur 4.

III. Tri par insertion

1. Principe général

Méthode 2 : *Tri par insertion*

1. On prend le deuxième élément dans la liste.
2. On insert cet élément à la bonne position dans la sous-liste précédente. Pour ce faire, on décale l'élément vers la gauche tant qu'il est inférieur à l'élément précédent et qu'il n'est pas en première position.
3. On recommence avec le reste de la liste, jusqu'au dernier élément.

Exemple 5: *Vidéo expliquant le tri par insertion*

2. Algorithme de tri par insertion

Entrée : .....

Pré-condition : .....

.....

.....

Sortie : .....

Post-condition : .....

.....

.....

.....

Algorithme en langage naturel :

```
fonction tri_ insertion(liste L) :
    pour PosElt allant de 1 à n-1
        PosAct ← PosElt
        element ← L[PosElt]
        tant que element < L[PosAct - 1] et PosAct > 0
            L[PosAct] ← L[PosAct - 1]
            PosAct ← PosAct - 1
        fin tant que
        liste[PosAct] ← element
    fin pour
    Afficher L
```