Les tris

Définition d'un algorithme de Tri

Les tableaux permettent de stocker plusieurs éléments de même type au sein d'une seule entité. Lorsque le type de ces éléments possède un ordre total, on peut donc les ranger en ordre croissant ou décroissant. Trier un tableau c'est donc ranger les éléments d'un tableau en ordre croissant ou décroissant.
Dans ce cours on ne fera que des tris en ordre croissant.
Il existe plusieurs méthodes de tri qui se différencient par leur complexité d'exécution et leur complexité de compréhension pour le programmeur.

La procédure échanger

Tous les algorithmes de tri utilisent une procédure qui permet d'échanger (de permuter) la valeur de deux variables.
Dans le cas où les variables sont entières, la procédure échanger est la suivante :

```
procédure inverser(E/S val1, E/S val2 : entiers)

    début
        val1 ← val1 + val2
        val2 ← val1 - val2
        val1 ← val1 - val2
    fin
ou
    début
        tmp ← val1
        val1 ← val2
        val2 ← tmp
    fin
```

Tri par minimum successif

Principe

Le tri par minimum successif est un tri par sélection : pour une place donnée, on sélectionne l'élément qui doit y être positionné.
De ce fait, si on parcourt la tableau de gauche à droite, on positionne à chaque fois le plus petit élément qui se trouve dans le sous tableau droit. Ou plus généralement :
pour trier le sous-tableau t[i … nbElements], il suffit de positionner au rang i le plus petit élément de ce sous-tableau et de trier le sous-tableau t[i+1..nbElements].

Par exemple, pour trier *<101, 115, 30, 63, 47, 20>*, on va avoir les boucles suivantes :

i=1 <101, 115, 30, 63, 47, 20>
i=2 <20, 115, 30, 63, 47, 101>
i=3 <20, 30, 115, 63, 47, 101>
i=4 <20, 30, 47, 63, 115, 101>
i=5 <20,30, 47, 63, 115, 101>

Donc en sortie : *<20, 30, 47, 63, 101, 155>*

Il nous faut donc une fonction qui pour soit capable de déterminer le plus petit élément (en fait l'indice du plus petit élément) d'un tableau à partir d'un certain rang.

Fonction indiceDuMinimum

```
fonction minimum(tab : tableau[ 1 … MAX] d'entiers; rang, nbElements : naturels) :
naturel
    variables    i, indice : naturels
    début
        indice ← rang
        pour i ← rang+1 à nbElements faire
            si t[i] < t[indice] alors
                indice ← i
            fsi
        fpour
        retourner indice
    fin
```

L'algorithme de tri est donc :

```
procédure triParMinimumSuccessif(E/S t : tableau[1 … MAX] d'entiers, E
nbElements : naturel)
    variables    i,indice : naturels
    début
        pour i ← 1 à nbElements-1 faire
            indice ← minimum(t, i, nbElements)
            si i ≠ indice alors
                inverser(t[i],t[indice])
            fsi
        fpour
    fin
```

Complexité

Recherche du minimum sur un tableau de taille n → Parcours du tableau.
Complexité en $\Theta(n^2)$.

Le tri à bulles

Principe de la méthode : sélectionner le minimum du tableau en parcourant le tableau de la fin au début et en échangeant tout couple d'éléments consécutifs non ordonnés.

Par exemple, pour trier *<101, 115, 30, 63, 47, 20>*, on va avoir les boucles suivantes :

i=1 *<101, 115, 30, 63, 47, 20>*
<101, 115, 30, 63, 20, 47>
<101, 115, 30, 20, 63, 47>
<101, 115, 20, 30, 63, 47>
<101, 20, 115, 30, 63, 47>
i=2 *<20, 101, 115, 30, 63, 47>*
...
i=3 *<20, 30,101, 115, 47, 63>*
...
i=4 *<20, 30,47,101, 115, 63>*
...
i=5 *<20, 30, 47, 63, 101, 115>*

Donc en sortie : *<20, 30, 47, 63, 101, 115>*

```
procédure triBulles(E/S t : tableau[1 … MAX] d'entiers, E nbElements : naturel)
    variables    i,k : naturels
    début
        pour i ← 1 à nbElements faire
            pour k ← nbElements à i+1 pas -1 faire
                si t[k] < t[k-1] alors
                    inverser(t[k],t[k-1])
                fsi
            fpour
        fpour
    fin
```

Complexité

Nombre de tests (moyenne et pire des cas) : Compléxité en $\Theta(n^2)$.
Nombre d'échanges (pire des cas) :
$E(n) = n - 1 + n - 2 + \cdots + 1 \rightarrow \Theta(n^2)$
Nombre d'échange (en moyenne) $\Theta(n^2)$ (calcul plus compliqué)
En résumé : complexité en $\Theta(n^2)$.

Le tri rapide

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.
Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, pour partitionner un sous-tableau :

1. on place le pivot à la fin (arbitrairement), en l'échangeant avec le dernier élément du sous-tableau ;
2. on place tous les éléments inférieurs au pivot en début du sous-tableau ;
3. on place le pivot à la fin des éléments déplacés.

```
procédure triRapide(E/S t : tableau[1 … MAX] d'entiers, E/S premier, dernier :
naturels, pivot : naturel)
        variables       gauche, droite: naturels
        début
                gauche ← premier - 1
                droite ← dernier + 1
                inverser(t[pivot], t[dernier])
                tant que gauche < droite faire
                        répéter
                                gauche ← gauche + 1
                        tant que t[gauche] < t[dernier]
                        si gauche < droite
                                répéter
                                        droite ← droite - 1
                                tant que t[droite] > t[dernier]
                                si t[gauche] ≥ t[droite]
                                        inverser(t[gauche], t[droite]
                                fin si
                        sinon
                                inverser(t[gauche], t[dernier])
                        fin si
                fin tant que
                triRapide(t, premier, gauche-1, t[gauche-1])
                triRapide(t, droite+1, dernier, t[dernier])
        fin
```

Algorithme alternatif

```
fonction partition(t : tableau[1 … MAX] d'entiers, premier, dernier, pivot : entiers)
    variables    i, j : naturels
    début
        inverser(t[pivot],t[dernier])
        j ← premier
        pour i ← premier à dernier-1 faire
            si t[i] ≤ t[dernier] alors
                inverser(t[i],t[j])
                j ← j +1
            fin si
        fin pour
        inverser(t[dernier],t[j])
        retourner j
    fin
procédure triRapide(E/S t : tableau[1 … MAX] d'entiers, E/S premier, dernier :
naturels)
    variables    pivot : naturel
    début
        si premier < dernier alors
            pivot ← choixPivot(t, premier, dernier) {arbitraire/aléatoire/optimal}
            pivot ← partition(t, premier, dernier, pivot)
            triRapide(t, premier, pivot-1)
            triRapide(t, pivot+1, dernier)
        fin si
    fin
```

Choix du pivot et complexité

Pivot arbitraire

Une manière simple de choisir le pivot est de prendre toujours le premier élément du sous-tableau courant (ou le dernier). Lorsque toutes les permutations possibles des entrées sont équiprobables, la complexité moyenne du tri rapide en sélectionnant le pivot de cette façon est $\Theta(n \log n)$. Cependant, la complexité dans le pire cas est $\Theta(n^2)$, et celle-ci est atteinte lorsque l'entrée est déjà triée ou presque triée.

Si on prend comme pivot le milieu du tableau, le résultat est identique, bien que les entrées problématiques soient différentes.

Il est possible d'appliquer une permutation aléatoire au tableau pour éviter que l'algorithme soit systématiquement lent sur certaines entrées. Cependant, cette technique est généralement moins efficace que de choisir le pivot aléatoirement.

Pivot aléatoire

Si on utilise la méthode donnée dans la description de l'algorithme, c'est-à-dire choisir le pivot aléatoirement de manière uniforme parmi tous les éléments, alors la complexité moyenne du tri rapide est $\Theta(n \log n)$ sur n'importe quelle entrée. Dans le pire cas, la complexité est $\Theta(n^2)$. Néanmoins, l'écart type de la complexité est seulement $\Theta(n)$, ce qui signifie que l'algorithme s'écarte peu du temps d'exécution moyen.

Dans le meilleur cas, l'algorithme est en $\Theta(n \log n)$.

Pivot optimal

En théorie, on pourrait faire en sorte que la complexité de l'algorithme soit $\Theta(n \log n)$ dans le pire cas en prenant comme pivot la valeur médiane du sous-tableau. L'algorithme BFPRT ou médiane des médianes permet en effet de calculer cette médiane de façon déterministe en temps linéaire. Mais l'algorithme n'est pas suffisamment efficace dans la pratique, et cette variante est peu utilisée.