Chapitre 10 – ALGORITHMES DE TRI

Trier un tableau c'est ranger ses éléments dans l'ordre croissant ou décroissant.
Il existe plusieurs méthodes de tri qui se différencient au niveau de la simplicité et de l'efficacité.
Dans ce cours on ne fera que des tris dans l'ordre *croissant.*

1. Tri par sélection du minimum

Principe

L'algorithme consiste à déterminer successivement l'élément devant se retrouver en première position, en deuxième position, etc., c'est à dire le plus petit des éléments restants, et ainsi de suite.
Pour cela, on parcourt le tableau de gauche à droite et , à chaque position i, on place le plus petit élément qui se trouve dans le sous tableau droit (entre les positions i et N-1)

Algorithme

```
Procédure triSelection ( E-S tab : Tableau[ ]d'entier , E N : entier )
Variable
      i, j, imin : entier
début
      Pour i = 0 à N-2 Faire
          // on cherche l'indice du plus petit élément entre i et N-1 et on range l'élément à la position i
          imin = i
          Pour j =  i+1 à N-1 Faire
                Si (tab[ j ] < tab[imin]) Alors
                     imin = j
                FinSi
          FinPour
          echanger(tab[ i ], tab[ imin ] )
      FinPour
FinProcédure
```

Exemple

Pour trier <101, 115, 30, 63, 47, 20>, on va avoir les étapes suivantes :

i=0 <**101**, 115, 30, 63, 47, 20>
i=1 <20, **115**, 30, 63, 47, 101>
i=2 <20, 30, **115**, 63, 47, 101>
i=3 <20, 30, 47, **63**, 115, 101>
i=4 <20, 30, 47, 63, **115**, 101>

A la fin, on a : <20, 30, 47, 63, 101, 155>

2. Tri par insertion

Principe

On procède par étape : à chaque étape i > 0, on insère le i ème élément entre les positions 0 et i en sachant que les éléments situés entre 0 et i-1 sont déjà triés.

Algorithme

```
procédure triInsertion ( E-S tab : Tableau[ ] d'Entier , E N : Entier )
Variable
        i, j : Entier
       tmp : Entier
Début
       Pour i = 1 à N-1 Faire
              tmp = tab[ i ]

              // on recherche séquentiellement la position d'insertion
              j =  i^p
              Tant que ( (j > 0) ET ( tab[ j-1 ] > tmp) )Faire
                     tab[ j ] = tab[ j-1 ]      // décalage
                     j = j-1
              FinTQ

              tab[ j ] = tmp   // insertion

       FinPour
FinProcédure
```

Exemple

Pour trier <101, 115, 30, 63, 47, 20>, on va avoir les étapes suivantes :

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

A la fin, on a : <20, 30, 47, 63, 101, 155>

3. Tri bulles

Principe

Le principe consiste à déplacer les petits éléments vers le début du tableau et les grands vers la fin du tableau en effectuant des échanges successifs.
A chaque étape i, on parcourt le tableau à partir de la fin en comparant les éléments consécutifs deux à deux et en les échangeant s'ils ne sont pas dans le bon ordre. Ainsi, à la fin de chaque étape i, on range à la position i le plus petit élément entre i et N.

Algorithme

```
procédure triBulles( E-S tab : Tableau[ ] d'Entier , E N : Entier )
Variable
        i, j : Entier
Début
        Pour i = 0 à N - 2 Faire
                Pour j = N-1 à i+1 par pas de - 1 Faire
                        Si ( tab[ j ]  <  tab[ j - 1 ] ) alors
                            echanger( tab[ j ] , tab[ j-1 ] )
                        FinSi
                FinPour
        FinPour
FinProcédure
```

Exemple

Pour trier <101, 115, 30, 63, 47, 20>, on va avoir les étapes suivantes :

i=0 <**101**, 115, 30, 63, 47, 20>
i=1 <20, **101**, 115, 30, 63, 47>
i=2 <20, 30, **101**, 115, 47, 63>
i=3 <20, 30, 47, **101**, 115, 63>
i=4 <20, 30, 47, 63, **101**, 115>

A la fin, on a : <20, 30, 47, 63, 101, 155>

4.Tri fusion

L'algorithme de tri fusion est construit suivant le paradigme " diviser pour régner " de la manière suivante :

1. **Diviser:** Diviser le tableau à trier en deux sous-tableaux.
2. **Régner:** Trier les deux sous-tableaux récursivement.
3. **Combiner:** Fusionner les deux sous-tableaux triés.

4-1. Algorithme de fusion

Le principe de la fusion est simple : à chaque étape, on compare les minimums des deux sous-tableaux triés, le plus petit des deux étant le minimum de l'ensemble on le copie dans un tableau résultat de la fusion et on recommence.
On conçoit ainsi un algorithme fusionner qui prend comme paramètres le tableau *tab* et trois indices sur le tableau, *deb*, *mil* et *fin*, tels que $deb \leq mil < fin$ et tels que les sous-tableaux [*deb .. mil*] et [*mil+1 .. fin*] soient triés :

```
Procédure fusionner(E-S tab: tableau [ ] d'Entier, E deb: Entier, mil: Entier, fin : Entier)
Variable
   tab2 : tableau [ fin – deb + 1 ] d'Entier  // tableau temporaire dans lequel on met le résultat de la fusion
   i : Entier        //  indice de l'élément courant dans le sous tableau [deb .. mil]
   j : Entier        //  indice de l'élément courant dans le sous tableau [mil+1 .. fin]
   k : Entier        //  indice de la position d'insertion dans tab2
Début
   i = deb
   j = mil+1
   k = deb
   Tant que ( i <= mil ) et  ( j <= fin ) Faire
         Si ( tab[ i ] < tab[ j ] ) Alors
               tab2[ k ] =  tab[ i ]
               i  = i+1
         Sinon
               tab2[ k ] = tab[ j ]
               j =  j+1
         FinSi
         k =  k+1
   FinTQ
   // on copie dans tab2 le reste des éléments du sous tableau [deb .. mil]
   Tant que  ( i <= mil ) Faire
         tab2[ k ] = tab[ i ]
         i =  i+1
         k =  k+1
   FinTQ
   // on copie dans tab2 le reste des éléments du sous tableau [mil+1 .. fin]
   Tant que  ( j <= fin ) Faire
         tab2[ k ] = tab[j]
         j =  j+1
         k =  k+1
   FinTQ
   // on recopie le résultat dans le tableau original
   Pour k =  deb à fin Faire
         tab[ k ] =  tab2[ k ]
   FinPour
FinProcédure
```

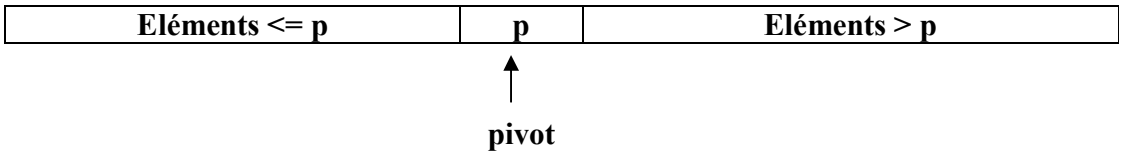
4-2. L'algorithme de tri fusion

```
Procédure triFusion(E-S tab : tableau [ ] d'Entier, E deb: Entier, fin : Entier)
Variable
        mil : Entier
Début
        Si (deb < fin) Alors

                mil = (deb + fin ) div 2

                triFusion(tab,deb,mil)
                triFusion(tab,mil+1,fin)

                fusionner(tab,deb,mil,fin)

          FinSi
FinProcédure
```

5. Tri rapide

Le principe du tri rapide est basé sur le modèle "diviser pour régner". Pour trier un tableau tab d'indice de début deb et d'indice de fin fin, le principe est le suivant :

1. Diviser: On choisit arbitrairement un élément du tableau que l'on appellera pivot. En général, on prend comme pivot le premier élément, c'est à dire tab[deb]. On partitionne (en le réarrangeant) le tableau en deux sous-tableaux, de telle façon que tous les éléments inférieurs ou égaux au pivot soient placés avant lui, et que tous les éléments strictement supérieurs au pivot soient placés après lui.



2. Régner: Le pivot étant à sa place définitive, il reste à trier (de la même façon) les deux sous-tableaux placés avant et après lui.

3. Combiner: Les sous-tableaux étant triés sur place, on peut alors affirmer que le tableau est trié en totalité.

5-1. Algorithme de partitionnement :

```
Procédure partitionner(E-S tab : tableau [ ] d'Entier, E deb: Entier, fin : Entier, E-S placePivot : Entier)
Variable
        pivot: Entier
        i : Entier
Début
    pivot = tab[ deb ]
    placepivot = deb

    Pour i = deb+1 à fin
        si ( tab[ i ]  <=  pivot ) alors
            placepivot = placePivot +1
            échanger( tab[ placePivot ] , tab[ i ] )
        FinSi
    Finpour

    échanger( tab[ deb ], tab[ placePivot ] )

FinProcédure
```

5-2. L'algorithme de tri rapide

```
procédure triRapide(E-S tab : tableau [ ] d'Entier, E deb : Entier, fin : Entier)
Variable
        placePivot : Entier      // indice du pivot après le partitionnement
Début
        Si (deb < fin) Alors
            partitionner( tab, deb, fin, placePivot)
            triRapide( tab, deb, placePivot-1)
            triRapide( tab, placePivot+1, fin)
     FinSi
FinProcédure
```