CHAPITRE 3

LES ALGORITHMES DE TRI

I/ Introduction

Selon le dictionnaire
"**trier**" signifie «répartir des objets suivant certains critères».

En informatique le "tri" un processus de classement d'une suite d'éléments dans un ordre donné.

Il existe deux catégories de tris :
- **Les tris internes** : Méthodes destinées à des masses de données limitées, stockées dans une structure de données se trouvant dans la mémoire centrale (Exemple : tableaux).

- **Les tris externes** : Méthodes destinées à de grandes masses de données, stockées dans des structures de données comme par exemple les fichiers.

I/ Tri par sélection

Activité

Ecrire un programme nommé Tri_Sélection, qui remplit de façon aléatoire un tableau T par N Réels.
Affiche l'état actuel du tableau puis celui du tableau trié par ordre croissant en utilisant la méthode du tri par sélection.
N est entier compris entre 4 et 25.

* - * - * - * - * - * - * - * - * - * - *

Analyse du programme Tri_Sélection

Résultat = Tableau trié
Traitement :
- Une procédure Affiche_Tab, permet d'afficher le contenu du tableau. Elle va servir pour l'affichage du tableau non trié puis pour le tableau trié.
- Une procédure Tri_Sélect, permet le tri par sélection du tableau.
- Une procédure Saisie, permet la saisie et le test de N.
- Une procédure Remplir_Hasard, permet de remplir de façon aléatoire (au hasard) le tableau.

Fin Analyse

Algorithme

0) **Début** Tri_Sélection
1) Proc Saisie (N)
2) Proc Remplir_Hasard (T, N)
3) Ecrire ("Tableau non trié ")
4) Proc Affiche_Tab (T, N)

5) Proc Tri_Select (T, N)
6) Ecrire ("Tableau trié ")

7) Proc Affiche_Tab (T, N)
8) **Fin Tri_Sélection**

Tableau de déclaration des objets globaux

Objet	Type / Nature	Rôle
N	Entier	Nombre d'éléments du tableau
T	Tab	Tableau des réels
Saisie	Procédure	Saisie et test de N
Remplir_Hasard	Procédure	Remplit au hasard le tableau T
Affiche_Tab	Procédure	Affiche le contenu du tableau
Tri_Sélect	Procédure	Tri le tableau par sélection

Tableau de déclaration des nouveaux types

Types
TAB = Tableau de 25 réels

Analyse de la procédure Saisie

Résultat = Saisie et test de N
Traitement
Répéter
N = Donnée ("Entrer la taille du tableau : ")
Jusqu'à $(4 \leq N \leq 25)$
Fin Saisie

Algorithme

0) **Procédure** Saisie (**VAR** N : Entier)
1) Répéter
Ecrire ("Entrer la taille du tableau : ")
Lire (N)
Jusqu'à $(4 \leq N \leq 25)$
2) **Fin Saisie**

Analyse de la procédure Remplir_Hasard

Résultat = Remplir le tableau T par des réel pris au hasard
Traitement :
- La fonction prédéfinie Hasard (N) en Pascal Random (N), renvoie un entier aléatoire compris entre 0 et N-1.
Si N est omis, la fonction renvoie un réel compris entre 0 et 9.999….
Donc pour obtenir un réel, par exemple entre 0 et 100, on multiplie le résultat de Hasard par 100.

```
Pour i de 1 à N Faire
        T[i] ← Hasard  * 100
Fin Pour
```

En Pascal, pour que cette fonction génère à chaque appel des nombres différents, elle doit être initialisée avec la procédure prédéfinie Randomize.
Fin Analyse

Algorithme

```
0) Procédure Remplir_Hasard (VAR T : Vect ; N : Entier )
1) Randomize
2) Pour i de 1 à N Faire
                T[i] ← Hasard  * 100
   Fin Pour
3) Fin Remplir_Hasard
```

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur

Analyse de la procédure Tri_Select

Résultat = Tableau T trié
Traitement :
Lorsque le tri est par ordre croissant, la méthode est parfois appelée tri par recherche de minima, car on commence par le plus petit élément.

Lorsque le tri est par ordre décroissant, la méthode appelée tri par recherche de maxima, car on commence par le plus grand.

Dans notre cas c'est un tri décroissant, donc c'est une sélection par par recherche de minima.

La méthode de tri par sélection est donc la suivante :
1- On cherche le plus petit élément en parcourant tout le tableau et on le permute avec celui occupant la première case.
2- La plus petite valeur occupe définitivement sa place qui est la première case. On cherche maintenant la plus petite valeur dans le tableau T mais on commençant à partir du deuxième élément. Si elle existe elle sera permuter avec c'elle de la deuxième case.
3- On répète le même traitement à partie de la troisième case et ainsi de suite jusqu'à la case N-1.

- La recherche de la position de la plus petite valeur à partir d'un point de départ donné est confiée à une fonction nommé Cherche_Min.
- La permutation du contenu de deux cases du tableau est faite par une procédure nommé Permute.
Fin Analyse

Algorithme

0) **Procédure** Tri_Select (**VAR** T : Vect ; N : Entier)
1) Pour i de 1 à N-1 Faire
    Pos_Min ← Fn Cherche_Min (T, i, N)
    Si Pos_Min ≠ i Alors Proc Permute (T[i] , T[Pos_Min])
    Fin Si
  Fin Pour
2) **Fin tri_Select**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur
Pos_Min	Entier	Position de la valeur minimale
Cherche_Min	Fonction	Retourne la position de la valeur minimale
Permute	Procédure	Permute le contenu de deux variables

Analyse de la fonction Cherche_Min

Résultat = Cherche_Min
Traitement :
- On suppose que l'élément de départ est le plus petit donc sa position est son indice.
    Min ← T[départ]
    Indice ← Départ
- On parcourt le tableau à partir de la position départ + 1 jusqu'à N à la recherche d'un élément plus petit, s'il existe, on mémorise sa valeur et sa position.
    Pour j de départ + 1 à N Faire
        Si T[j] < Min Alors Min ← T[j]
                             indice ← j
        Fin Si
    Fin Pour
    - Le résultat est la valeur indice
Cherche_Min ← indice
Fin Analyse

Algorithme

0) **Fonction** Cherche_Min (T : Vect ; départ, N : Entier) : Entier
1) Min ← T[départ]
2) Indice ← Départ
3) Pour j de départ + 1 à N Faire
        Si T[j] < Min Alors Min ← T[j]
                             indice ← j
        Fin Si
  Fin Pour
4) Cherche_Min ← indice
5) **Fin Cherche_Min**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
j	Entier	Compteur
Min	Réel	Valeur minimale
indice	Entier	Position de la valeur minimale

Analyse de la procédure Permute

Résultat = Permuter le contenu de deux variables

Traitement :
- Pour permuter le contenu de deux variables, nous allons utiliser une troisième pour la sauvegarde temporaire d'une des deux valeurs.

temp ← a
a ← b
b ← temp

Fin Anlyse

Algorithme

0) **Procédure** Permute (**VAR** a, b : Réel)
1) temp ← a
   a ← b
   b ← temp
2) **Fin Permute**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
temp	Réel	Variable temporaire pour la permutation

Analyse de la procédure Affiche_Tab

Résultat = Afficher le contenu du tableau T.
Traitement :
- Parcourir le tableau et afficher tous ses éléments à l'aide d'une itération complète.

Pour i de 1 à N Faire
    Ecrite (T[i])
Fin Pour

Fin Analyse

Algorithme

0) **Procédure** Affiche_Tab (T : Vect ; N : Entier)

1) Pour i de 1 à N Faire
Ecrite (T[i])
Fin Pour
2) Fin Affiche_Tab

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur

Programme en Pascal

```
PROGRAM Tri_Selection ;
USES Crt ;
TYPE
        Vect = ARRAY [1 .. 25 ] Of Real ;
VAR
        N : Integer ;
        T : Vect ;

{-------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
        Repeat
                Write ('Entrer la taille du tableau : ') ;
                ReadLn (N) ;
        Until N In [4 .. 25 ] ;
END ;


{-------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR T : Vect ; N : Integer ) ;
VAR   i : Integer ;
BEGIN
        Randomize ;
        For i := 1 To N Do
                T[i] := Random  * 100 ;
END ;


{-------------------------------------------------------------------------------}
FUNCTION Cherche_Min (T : Vect ; depart, N : Integer) : Integer ;
VAR
        j, indice : Integer ;
        Min : Real ;
BEGIN
```

```
        Min := T[depart] ;
        Indice := depart ;
        For  j :=  depart + 1 To N  Do
        Begin
                If  (T[j] < Min) Then Begin
                                                Min := T[j] ;
                                                indice := j ;
                                        End ;
        End ;
  Cherche_Min := indice ;
End ;


{-------------------------------------------------------------------------------}
PROCEDURE Permute (VAR a, b : Real ) ;
VAR   Temp : Real ;
BEGIN
        temp := a ;
        a := b ;
        b := temp ;
END ;

{-------------------------------------------------------------------------------}
PROCEDURE Tri_Select (VAR T : Vect ; N : Integer ) ;
VAR
        i, Pos_Min : Integer ;
BEGIN
        For i := 1 To N-1 Do
        Begin
                Pos_Min := Cherche_Min (T, i, N) ;
                If Pos_Min <> i  Then  Permute (T[i] , T[Pos_Min]) ;
        End ;
END ;


{-------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Vect ; N : Integer) ;
VAR   i : Integer ;
BEGIN
        For i := 1 To  N  Do
                WriteLn (T[i] : 8 : 3) ;

END ;


{= = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
```

```
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        Tri_Select (T, N) ;
        WriteLn ('Tableau trié " );
        Affiche_Tab (T, N) ;
END.
```

II/ Tri à bulles

Activité

Ecrire un programme nommé Tri_Bulles, qui permet le tri d'un tableau T de N réels, par la méthode du tri à bulles.
Ce programme affiche le contenu du tableau non trié puis le contenu du tableau trié par ordre décroissant.
N est entier compris entre 4 et 25.
Le tableau est remplit de façon aléatoire un tableau T par N Réels.

* - * - * - * - * - * - * - * - * - * - *

Analyse du programme Tri_Bulles

Résultat = Tableau trié
Traitement :
- Les procédures de saisie de N, du remplissage du tableau et de l'affichage sont les mêmes que ceux de l'activité précédente.
- Une procédure nommée Bulles, permet le tri du tableau par une des méthodes du tri à bulles.

Fin Analyse

Algorithme

0) **Début** Tri_Bulles
1) Proc Saisie (N)
2) Proc Remplir_Hasard (T, N)
3) Ecrire ("Tableau non trié ")
4) Proc Affiche_Tab (T, N)
5) Proc Bulles (T, N)
6) Ecrire ("Tableau trié ")
7) Proc Affiche_Tab (T, N)
8) **Fin Tri_Sélection**

Analyse de la procédure Bulles

Il existe plusieurs méthodes du tri à bulles, en voici une :
L'algorithme du tri à bulles (bubble sort en anglais) consiste à comparer les différentes valeurs adjacentes du tableau T, et à les permuter s'ils ne sont pas dans le bon ordre.

Pour i de 1 à N-1 Faire
Si (T[i] > T[i+1]) Alors Proc Permuter (T[i], T[i+1])

Si au moins une permutation est faite, une variable booléenne (échange) reçoit par exemple Vrai, si elle a été initialisée au départ à Faux

échange ← Vrai

Le tri se termine quand il n'y a plus de permutations (échange = faux) sinon on répète le même traitement.

Jusqu'à échange = Faux

Fin Analyse

Algorithme

```
0) Procédure Bulles (VAR T : Vect ; N : Entier)
1) Répéter
        échange ← Faux
        Pour i de 1 à N-1 Faire
                Si (T[i] > T[i +1] )   Alors Proc Permute (T[i], T[i+1])
                                                échange ← Vrai
                Fin Si
        Fin Pour
   Jusqu'à échange = Faux
2) Fin Bulles
```

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur
échange	Booléen	Drapeau de test
Permuter	Procédure	Permute le contenu de deux variables

Programme complet en Pascal

```
PROGRAM Tri_Bulles ;
USES Crt ;

TYPE
        Vect = ARRAY [1 .. 25 ] Of Real ;

VAR
        N : Integer ;
        T : Vect ;
```

```
{-------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
        Repeat
                Write ('Entrer la taille du tableau : ') ;
                ReadLn (N) ;
        Until N  In  [4 .. 25 ] ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR  T : Vect ; N : Integer ) ;
VAR   i : Integer ;
BEGIN
        Randomize ;
        For i := 1 To N Do
                T[i] := Random  * 100 ;
END ;


{-------------------------------------------------------------------------------}
PROCEDURE Permute (VAR a, b : Real ) ;
VAR   Temp : Real ;
BEGIN
        temp := a ;
        a := b ;
        b := temp ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE Bulles (VAR T : Vect ; N : Integer ) ;
VAR
        i : Integer ;
        echange : Boolean ;

BEGIN
        Repeat
                echange := False ;
                For  i := 1 To N-1 Do
                If  (T[i] > T[i +1] )   Then
                                        Begin
                                                Permute (T[i], T[i+1]) ;
                                                echange := True ;
                                        End ;
        Until (echange = False);
END ;
```

```
{-------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Vect ; N : Integer) ;
VAR   i : Integer ;
BEGIN
        For i := 1 To  N  Do
                WriteLn (T[i] : 8 : 3) ;
END ;

{= = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        Bulles (T, N) ;
        WriteLn ('Tableau trié ');
        Affiche_Tab (T, N) ;
END.
```

III/ Tri par insertion

Activité

Ecrire un programme nommé Tri_Insertion, qui permet le tri d'un tableau T de N réels, par la méthode du tri par insertion.
Ce programme affiche le contenu du tableau non trié puis le contenu du tableau trié par ordre décroissant.
N est entier compris entre 4 et 25.
Le tableau est remplit de façon aléatoire un tableau T par N Réels.

* - * - * - * - * - * - * - * - * - * - *

Analyse du programme Tri_Insertion

Résultat = Tableau trié
Traitement :
- Les procédures de saisie de N, du remplissage du tableau et de l'affichage sont les mêmes que ceux de l'activité précédente.
- Une procédure nommée T_Insertion, permet le tri du tableau par la méthode du tri par insertion.

Fin Analyse

Algorithme

0) **Début** Tri_Insertion
1) Proc Saisie (N)

2) Proc Remplir_Hasard (T, N)
3) Ecrire ("Tableau non trié ")
4) Proc Affiche_Tab (T, N)
5) Proc T_Insertion (T, N)
6) Ecrire ("Tableau trié ")
7) Proc Affiche_Tab (T, N)
8) **Fin Tri_Insertion**

Analyse de la procédure T_Insertion

Le principe du tri par insertion, est identique au classement qu'un joueur de "rami" ou de "belote" utilise pour ranger ses cartes. Il tire une carte et la met à sa place parmi les autres cartes déjà rangées puis il recommence avec la carte suivante jusqu'au rangement de toutes les certes dans sa main.

Le principe global est d'insérer ième élément à sa bonne place dans la liste formée par les (i-1) éléments qui le précèdent e qui sont déjà triés.
Cette action est répétée jusqu'au dernier élément (le $N^{ième}$).

Pour i de 2 à N Faire

L'itération commence à partir de 2, car on considère que le premier élément est trié et qu'il occupe sa place.

Le processus d'insertion consiste à :
- Utiliser une variable intermédiaire tmp pour conserver la valeur à insérer,
- Déplacer les éléments T[i-1], T[i-2], ... vers la droite tant que leur valeur est supérieure à celle de tmp.
- Affecter alors à l'emplacement laissé libre par ce décalage la valeur de tmp.

Tmp ← T[i]
j ← i
Proc Décaler (T, j, Tmp)
T[j] ← Tmp

Fin Analyse

Algorithme

0) **Procédure** T_Insertion (**VAR** T : Vect ; N : Entier)
1) Pour i de 2 à N Faire
Tmp ← T[i]
j ← i
Proc Décaler (T, j, Tmp)
T[j] ← Tmp
Fin Pour
2) **Fin T_Insertion**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur

Tmp	Réel	Variable intermédiaire
j	Entier	Position d'insertion
Décaler	Procédure	Permettant de décaler des éléments d'un tableau d'un certain nombre de positions

Analyse de la procédure Décaler

Résultat : Décaler à droite les éléments du tableau T d'indice début à l'indice fin
Traitement :
Il s'agit d'un traitement répétitif à condition d'arrêt réalisant le décalage :

```
Tant Que T[p -1] > Temporaire Faire
        T[p] ← T[p-1]
        p ← p -1
 Fin Tant Que
```

- L'action de décalage est une simple affectation : T[p]← T[p-1]
  Puis La décrémentation de 1 de la variable p

Fin analyse

Algorithme

0) **Procédure** Décaler (**VAR** T : Vect; **VAR** p : Entier ; temporaire : Réel)
1) Tant que (T[p -1] > Temporaire) Faire
   T[p] ← T[p-1]
   p ← p -1
  Fin tant que
2) **Fin Décaler**

Programme complet en Pascal

```
PROGRAM Tri_Insertion ;
USES Crt ;

TYPE
    Vect = ARRAY [1 .. 25 ] Of Real ;

VAR
    N : Integer ;
    T : Vect ;


{-------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
    Repeat
        Write ('Entrer la taille du tableau : ') ;
```

```
                ReadLn (N) ;
        Until N In [4 .. 25 ] ;
END ;


{-------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR T : Vect ; N : Integer ) ;
VAR   i : Integer ;
BEGIN
        Randomize ;
        For i := 1 To N Do
                T[i] := Random  * 100 ;
END ;


{-------------------------------------------------------------------------------}

PROCEDURE Decaler (VAR T : Vect; VAR p : Integer ; temporaire : Integer) ;
BEGIN
        While T[p -1] > Temporaire Do
        Begin
                T[p] := T[p-1] ;
                p := p -1 ;
        End ;
END ;

{-------------------------------------------------------------------------------}
 PROCEDURE  T_Insertion (VAR T : Vect ; N : Integer) ;
VAR   i,  j: Integer ;
        tmp : Real ;
BEGIN
        For   i:= 2  To N Do
        Begin
                Tmp := T[i] ;
                j := i ;
                Decaler (T, j, Tmp) ;
                T[j] := Tmp ;
        End ;
END ;

{-------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Vect ; N : Integer) ;
VAR   i : Integer ;
BEGIN
        For i := 1 To  N  Do
                WriteLn (T[i] : 8 : 3) ;

END ;
```

```
{= = = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        T_Insertion (T, N) ;
        WriteLn ('Tableau trié ' );
        Affiche_Tab (T, N) ;
END.
```

IV/ Tri Shell

En analysant l'algorithme du tri par insertion, nous pouvons remarquer qu'il fait n*(n-1) comparaisons et décalages.
Il est toutefois évident que si le vecteur est initialement presque trié dans le bon ordre, le nombre d'opérations sera beaucoup plus réduit.
Si la méthode de tri par insertion est efficace quand la liste est à peu près triée, elle est inefficace en moyenne car elle ne change les valeurs que d'une position par instruction.
En effet, cette méthode traite un à un les éléments de la liste à trier et réalise un décalage des éléments précédents jusqu'à avoir la position d'insertion de l'élément en cours.

Principe du tri Shell

Donald L. Shell proposa, en 1959, une variante du tri par insertion. Ce tri consiste à trier séparément des sous tableaux du tableau initial, formés par les éléments répartis en un nombre P calculé d'éléments.

Le tri Shell est donc une amélioration du tri par insertion. Au lieu de faire un décalage de tous les éléments, il fera un décalage par pas de P éléments, ce qui permet d'affiner le tri du tableau et de faire moins de déplacements d'éléments.

Le tri Shell commence par un pas assez élevé et il le diminue au fur et à mesure jusqu'à arriver à un pas de 1. Ceci permet de réduire le désordre donc de diminuer le travail aux étapes suivantes. Le pas est diminué à l'aide d'une suite calculée.

Shell propose la suite d'incréments vérifiant P1 = 1, $P_{n+1} = 3P_n+1$ en réalisant les tris du plus grand incrément possible vers le plus petit.

Au dernier stade, P_0 est égal à 1 (retour au tri par insertion normal).
Nous déterminerons le pas maximal par récurrence en inversant la relation donnée ci-dessus :

$P_k + 1 = 3 * P_k + 1$

et en s'assurant qu'il y a encore un nombre suffisant de composantes dans les sous tableaux considérés.

Conclusion :
Le tri Shell trie chaque liste d'éléments séparés de P positions chacun avec le tri par insertion. L'algorithme effectue plusieurs fois cette opération en diminuant le pas P jusqu'à un pas égal à 1 ce qui équivaut à trier tous les éléments ensemble (tri par insertion normal).

Activité

Ecrire un programme nommé Tri_Shell, qui permet le tri d'un tableau T de N réels, par la méthode du tri Shell.
Comme dans les activités précédentes, ce programme :
- affiche le contenu du tableau non trié puis le contenu du tableau trié par ordre décroissant.
- Lit un entier N compris entre 4 et 25.
- Le tableau est remplit de façon aléatoire par N Réels.

* - * - * - * - * - * - * - * - * - * - *

Analyse du programme Tri_Shell

Résultat = Tableau trié
Traitement :
- Les procédures de saisie de N, du remplissage du tableau et de l'affichage sont les mêmes que ceux des activités de tri précédentes.
- Une procédure nommée Shell, permet le tri du tableau par la méthode du tri Shell.

Fin Analyse

Algorithme

0) **Début** Tri_Shell
1) Proc Saisie (N)
2) Proc Remplir_Hasard (T, N)
3) Ecrire ("Tableau non trié ")
4) Proc Affiche_Tab (T, N)
5) Proc Shell (T, N)
6) Ecrire ("Tableau trié ")
7) Proc Affiche_Tab (T, N)
8) **Fin Tri_Shell**

Analyse de la procédure Shell

Résultat = Tableau T trié
Traitement : C'est trier séparément des sous tableaux du tableau initial dont les éléments sont distants de P cases par la méthode du tri par insertion.

Pour chaque valeur du pas (P) calculé, on utilise un traitement répétitif à condition d'arrêt appliqué à chacun des sous tableaux.

P← 1
Tant Que (P< N) Faire
P ←(3* P +1)
Pour i de P +1 à NFaire
Si T[i] n'est pas à sa place alors on réalise les actions suivantes :
- Ranger la valeur de T[i] dans la variable TMP
- Décaler vers la droite par un pas = P les valeurs de T[i - P], T[i-2* P], ...
Jusqu'à arriver à une valeur qui est inférieure à T[i].
- Affecter au dernier élément décalé la valeur de TMP

Pour avoir la valeur maximale du pas, on utilise une boucle à condition d'arrêt :

P← 1
Tant Que (P< N) Faire
P ←(3* P +1)
Fin Tant Que

Algorithme

Nous allons écrire côte à côte les deux tris. Le tri par insertion normal, c'est-à-dire avec un pas = 1 et le tri Shell qui applique un pas P à ce même tri.

Tri par insertion normal	**Tri Shell**
0) **Procédure** Tri_insertion (N: Entier; **VAR** T : Vect)	0) **Procédure** Shell (N : entier; **VAR** T : Vect)
	1) P← 1 Tant Que (P< N) Faire P ←(3* P +1) Fin Tant Que
	2) Tant Que (P ≠1) Faire P ← P DIV 3 Pour c de P + 1 à N Faire Si (T[c] < T[c-p]) Alors Tmp ← T[c] pos ← c Tant Que (pos > P-1) et (T[pos-P]> tmp) Faire T[pos]← T[pos-P] pos←pos-P Fin Tant que T[pos]← Tmp Fin Si Fin Pour Fin Tant Que
1) Pour c de 2 à N Faire Si (T[c] < T[c-1]) Alors Tmp ← T[c] pos ← c Tant que (pos >1) ET (T[pos-1]> tmp) Faire T[pos]← T[pos-1] pos ← pos-1 Fin Tant que T[pos]← Tmp Fin Si Fin Pour	
2) **Fin Tri_insertion**	3) Fin Shell

Programme complet en Pascal

```
PROGRAM Tri_Shell ;
USES Crt ;

TYPE
        Vect = ARRAY [1 .. 25 ] Of Real ;

VAR
        N : Integer ;
        T : Vect ;


{-------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
        Repeat
                Write ('Entrer la taille du tableau : ') ;
                ReadLn (N) ;
        Until N In [4 .. 25 ] ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR T : Vect ; N : Integer ) ;
VAR   i : Integer ;
BEGIN
        Randomize ;
        For i := 1 To N Do
                T[i] := Random  * 100 ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE Shell (VAR T : Vect ; N : Integer) ;
VAR
    P, c, pos   : Integer ;
    Tmp : Real ;
BEGIN
    P := 1 ;
   While ( P < N) Do
    Begin
      p :=  (3* P +1) ;
```

```
    End ;

  While  (P < > 1) Do
  Begin
    P := P DIV  3 ;
    For  c := P + 1   To  N Do
    Begin
        If  (T[c] < T[c-p]) Then
        Begin
            Tmp := T[c] ;
            pos := c ;
            While (pos > P-1) AND  (T[pos-P]> tmp)  Do
                Begin
                        T[pos] := T[pos-P] ;
                        pos := pos-P ;
                End ; {End While }
                T[pos] := Tmp ;
        End ;  {End If }
   End ; {End For}
 End ; {End While }
END ;


{-------------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Vect ; N : Integer) ;
VAR   i : Integer ;
BEGIN
        For i := 1 To  N  Do
                WriteLn (T[i] : 8 ; 3) ;

END ;


{= = = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        Shell (T, N) ;
        WriteLn ('Tableau trié ');
        Affiche_Tab (T, N) ;
END.
```

V/ Tri par fusion

Principe

Le tri fusion est construit suivant la stratégie "diviser pour régner". Le principe de base est que pour résoudre un gros problème, il est souvent plus facile de le diviser en petits problèmes élémentaires. Une fois chaque petit problème résolu, il n'y a plus qu'à combiner les différentes solutions pour résoudre le problème global.
La méthode "diviser pour régner" est tout à fait applicable au problème de tri : plutôt que de trier le tableau complet, il est préférable de trier deux sous tableaux de taille égale, puis de fusionner les résultats.
Un algorithme récursif est très pratique. En effet, les deux sous tableaux seront eux même triés à l'aide de même algorithme de tri fusion. Un tableau ne comportant qu'un seul élément sera considéré comme trié : c'est la condition de fin du tri (arrêt).

Etapes de l'algorithme :

- Division de l'ensemble de valeurs en deux parties
- Tri de chacun des deux ensembles
- Fusion des deux ensembles obtenus pour reconstituer le tableau trié.

Remarque :
Nous constatons que la méthode de tri par fusion nécessite un tableau intermédiaire aussi grand que le tableau initial à trier et c'est là où réside le principal inconvénient.

Exemple
Soit le tableau T Suivant :

T	12	3	0	15	-5	22	23	-7	10	8	8	12	34	35	3
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

On peut diviser ce tableau en deux sous tableaux d'entiers T1 et T2 de longueurs respectives 7 et 8.

T1	12	3	0	15	-5	22	23
	1	2	3	4	5	6	7

T2	-7	10	8	8	12	34	35	3
	1	2	3	4	5	6	7	8

Triés par ordre croissant les deux sous tableaux T1 et T2

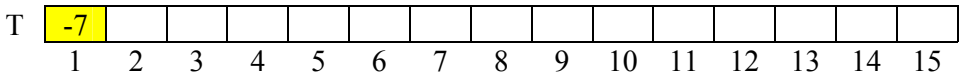
T1	-5	0	3	12	15	22	23
	1	2	3	4	5	6	7

T2	-7	3	8	8	10	12	34	35
	1	2	3	4	5	6	7	8

On propose d'utiliser la méthode de tri par fusion pour fusionner T1 et T2. Le résultat sera rangé dans le tableau T.

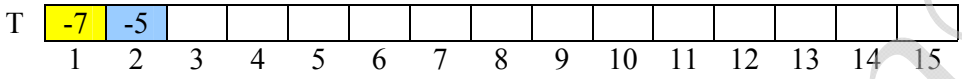
Etape 1
On commence par :
- comparer le premier élément de chacun des deux tableaux T1 et T2
  Le plus petit est T2 [1] = -7
- placer -7 dans T [1] et se pointer à l'élément n°2 de T2

- se pointer à l'élément n°2 de T
- Remarquez que nous sommes toujours à la première position de T1.



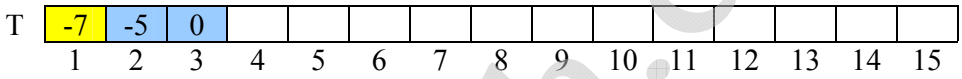
Etape 2

- comparer le premier élément de T1 et le deuxième élément de T2
  Le plus petit est T1 [1] = -5
- placer -5 dans T [2] et se pointer à l'élément n°2 de T1
- se pointer à l'élément n°3 de T

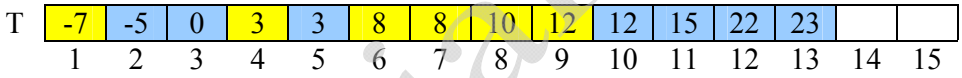


Etape 3

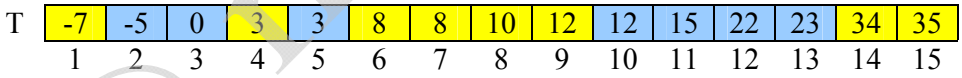
- comparer T1 [2] et T2 [2]
  Le plus petit est T1 [2] = 0
- placer 0 dans T [3] et se pointer à l'élément n°3 de T1
- se pointer à l'élément n°4 de T



- On poursuit la fusion de cette façon jusqu'à la fin de l'un des deux tableaux.



Tous les éléments du tableau T1 ont été rangés dans le tableau T, il ne reste que des éléments dans le tableau T2. Ces éléments vont être transférés directement dans le tableau T.



Activité

On propose d'écrire un programme qui réalise la méthode de tri décrite précédemment (diviser le tableau uniquement en 2 sous tableaux, sans appliquer la méthode complète du tri par fusion, c'est-à-dire sans les divisions jusqu'à un tableau à 1 seul élément).

- La saisie de N, le remplissage du tableau et l'affichage sont les mêmes que les activités précédentes.

* - * - * - * - * - * - * - * - * - * - * - * - *

Analyse du programme Tri_Fus1

Résultat = Tableau trié
Traitement :
- Les procédures de saisie de N, du remplissage du tableau et de l'affichage sont les mêmes que ceux des activités de tri précédentes.
- Une procédure nommée Fus, permet le tri du tableau par la méthode décrite.
Fin Analyse

Algorithme

0) **Début** Tri_Fus1
1) Proc Saisie (N)
2) Proc Remplir_Hasard (T, N)
3) Ecrire ("Tableau non trié ")
4) Proc Affiche_Tab (T, N)
5) Proc Fus (T, N)
6) Ecrire ("Tableau trié ")
7) Proc Affiche_Tab (T, N)
8) **Fin Tri_Fus1**

Analyse de la procédure Fus

Résultat = Tableau T trié
Traitement :
- Fusionner deux sous tableaux triés dans le tableau T, c'est le rôle de la procédure Fusionner.
- Trier successivement les deux sous tableaux, c'est le rôle de la procédure Trier.
- Diviser le tableau T en deux sous tableaux, c'est le rôle de la procédure Diviser.
Fin Analyse

Algorithme

0) **Procédure Fus** (**VAR** T : Vect; N : Entier)
1) Proc Diviser (T, N, T1, T2, N1, N2)
2) Proc Trier (T1, N1)
3) Proc Trier (T2, N2)
3) Proc Fusionner (T, N, T1, T2, N1, N2)
4) **Fin Fus**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
Diviser	Procédure	Diviser le tableau T en deux sous tableaux
Trier	Procédure	Trier un tableau
Fusionner	Procédure	Fusionner deux sous tableaux triés

Analyse de la procédure Diviser

Résultat = Diviser le tableau T en 2 sous tableaux
Traitement :
- Calculer les tailles des sous tableaux

N1 ← N DIV 2
N2 ← N - N1
- Diviser le tableau en deux sous tableaux presque égaux.
Pour i de 1 à N1 Faire
T1[i] ← T[i]
Fin Pour
Pour i de 1à N2 Faire
T2[i] ← T1[N1 + i]
Fin Pour
Fin Analyse

Algorithme

0) **Procédure Diviser** (T : Vect ; N : Entier, **VAR** T1, T2 : Vect2 ; **VAR** N1, N2 : Entier)
1) N1 ← N DIV 2
2) N2 ← N - N1
3) Pour i de 1 à N1 Faire
T1[i] ← T[i]
Fin Pour
4) Pour i de 1à N2 Faire
T2[i] ← T1[N1 + i]
Fin Pour
5) **Fin Diviser**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i	Entier	Compteur

Analyse de la procédure Trier

Résultat = Tableau trié
Traitement :
- Pour i de 1 à taille-1 Faire
Pour j de i+1 à taille Faire
Si Tx[i] > Tx[j] Alors aux ← Tx[i]
Tx[i] ← Tx[j]
Tx[j] ← aux
Fin Si
Fin Pour
Fin Analyse

Algorithme

0) **Procédure Trier** (**VAR** Tx : Vect2 ; taille : Entier)
1) Pour i de 1 à taille-1 Faire

```
        Pour j de i+1 à taille Faire
            Si Tx[i] > Tx[j] Alors aux ← Tx[i]
                                   Tx[i] ← Tx[j]
                                   Tx[j] ← aux
            Fin Si
        Fin Pour
```

2) **Fin Trier**

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
i, j	Entier	Compteurs
aux	Réel	Variable auxiliaire

Analyse de la procédure Fusionner

Résultat = Trier en fusionnant les deux tableaux T1 et T2

Traitement :

- Ranger les éléments des tableaux T1 et T2 jusqu'à la fin de l'un des tableaux T1 ou T2.

```
        c ← 0
        c1 ← 1
        c2 ← 1
            Répéter
                Inc (c, 1)
                Si (T1 [c1] < T2 [c2])
                 Alors
                    T[c] ¨  T1 [c1]
                    Inc (c1, 1)
                Sinon
                    T[c] ←  T2 [c2]
                    Inc (c2, 1)
                Fin Si
            Jusqu'à (c1 > N1) OU (c2 > N2)
```

- Ranger le reste des éléments du tableau T1 ou T2 (il s'agit d'une action de copie)

```
            Si (c1 > n1) Alors   {tous les éléments de T1 ont été rangés dans T,
                                  il reste à  copier les éléments de T2}
                Pour i de c2 à N2 Faire
                    T[c] ←  T2[i]
                    Inc (c, 1)
                Fin Pour
                    Sinon    {copier le reste des éléments de T2}
                Pour i de c1 à N1 Faire
                    T[c] ← T1[i]
                    Inc (c, 1)
                Fin Pour
            Fin Si
```

Fin Analyse

Algorithme

```
0) Procédure Fusionner (VAR T : Vect ; N : Entier ; T1, T2 : Vect2 ; N1, N2 : Entier)
1) c ← 0
2) c1 ← 1
3) c2 ← 1
4) Répéter
        Inc (c, 1)
        Si (T1 [c1] < T2 [c2])
                        Alors
                                T[c] ← T1 [c1]
                                Inc (c1, 1)
                        Sinon
                                T[c] ←  T2 [c2]
                                Inc (c2, 1)
                        Fin Si
   Jusqu'à (c1 > N1) OU (c2 > N2)

5) Si (c1 > N1) Alors
                Pour i de c2 à N2 Faire
                        T[c] ←  T2[i]
                        Inc (c, 1)
                Fin Pour
                Sinon
                Pour i de c1 à N1 Faire
                        T[c] ← T1[i]
                        Inc (c, 1)
                Fin Pour
        Fin Si
6) Fin Fusionner
```

Tableau de déclaration des objets locaux

Objet	Type / Nature	Rôle
c, c1, c2	Entier	Compteurs

Programme complet en Pascal

```
PROGRAM Tri_Fusion_Simple ;
USES Crt ;

TYPE
        Vect = ARRAY [1 .. 25 ] Of Real ;
        Vect2 = ARRAY [1.. (25 DIV 2) +1] Of Real ;
VAR
        N, N1, N2 : Integer ;
```

```
        T : Vect ;
        T1, T2 : Vect2;

{-------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
        Repeat
                Write ('Entrer la taille du tableau : ') ;
                ReadLn (N) ;
        Until N In [4 .. 25 ] ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR T : Vect ; N : Integer ) ;
VAR   i : Integer ;
BEGIN
        Randomize ;
        For i := 1 To N Do
                T[i] := Random  * 100 ;
END ;


{-------------------------------------------------------------------------------}
PROCEDURE Diviser (T : Vect ; N : Integer ; VAR  T1, T2 : Vect2 ; VAR N1, N2 :
Integer) ;
VAR   i : Integer ;
BEGIN
        N1 := N DIV 2 ;
        N2 := N - N1 ;
        For i :=  1 TO N1 Do
                T1[i] := T[i] ;
        For i := 1 TO N2 Do
                T2[i] := T[N1 + i] ;
END ;



{-------------------------------------------------------------------------------}
PROCEDURE  Fusionner (VAR T : Vect ; N : Integer ; T1, T2 : Vect2 ; N1, N2 : Integer) ;
VAR
        c, c1, c2, i : Integer ;
BEGIN
        c := 0  ;    c1 := 1;   c2 := 1 ;
        Repeat
                If (T1 [c1] < T2 [c2])  Then  Begin
                                        T[c] := T1 [c1] ;
                                        Inc (c1, 1) ;
```

```
                        End
                Else
                        Begin
                          T[c] :=  T2 [c2] ;
                           Inc (c2, 1) ;
                        End ;
        Inc (c, 1) ;
        Until (c1 > N1) OR (c2 > N2) ;
        If (c1 > N1) Then
                For i := c2 TO N2 Do
                Begin
                        T[c] := T2[i] ;
                        Inc (c, 1) ;
                End
                Else
                For i := c1 TO N1 Do
                Begin
                        T[c] := T1[i] ;
                        Inc (c, 1) ;
                End ;
END ;

{-------------------------------------------------------------------------------}
PROCEDURE Trier (VAR Tx : Vect2 ; taille : Integer ) ;
VAR
        i, j : Integer ;
        Aux : Real ;
BEGIN
        For i := 1 TO taille-1 Do
                For j := i+1 TO taille Do
                        If (Tx[i] > Tx[j]) Then Begin
                                                aux := Tx[i] ;
                                                 Tx[i] := Tx[j] ;
                                                 Tx[j] := aux ;
                        End ;
END ;

{-------------------------------------------------------------------------------}
PROCEDURE Fus (VAR T : Vect; N : Integer) ;
BEGIN
        Diviser (T, N, T1, T2, N1, N2) ;
        Trier (T1, N1) ;
        Trier (T2, N2) ;
        Fusionner (T, N, T1, T2, N1, N2) ;
END ;

{-------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Vect ; N : Integer) ;
VAR   i : Integer ;
```

```
BEGIN
        For   i := 1 To  N  Do
                WriteLn (T[i] : 8 : 3) ;
END ;


{= = = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        Fus (T, N) ;
        WriteLn ('Tableau trié ' );
        Affiche_Tab (T, N) ;
END.
```

Application :

Reprendre le programme précédent et remplacer la procédure de tri classique par une procédure de tri_fusion selon le principe "diviser pour régner". Cette procédure fait appel à la procédure fusion.

```
PROGRAM Tri_Fusion2 ;
USES WinCrt ;

TYPE
        Tab  = ARRAY [1 .. 25 ] Of Real ;

VAR
        N, N1, N2 : Integer ;
        T : Tab ;

{-------------------------------------------------------------------------------------}
PROCEDURE Saisie (VAR N : Integer );
BEGIN
        Repeat
                Write ('Entrer la taille du tableau : ') ;
                ReadLn (N) ;
        Until N In [4 .. 25 ] ;
END ;


{-------------------------------------------------------------------------------------}
PROCEDURE Remplir_Hasard (VAR T : Tab ; N : Integer ) ;
VAR   i : Integer ;
```

```
BEGIN
        Randomize ;
        For   i := 1 To N Do
                T[i] := Random  * 100 ;
END ;



{---- fusionner t[debut..milieu] et t[milieu +1..fi] ---------}
PROCEDURE fusion (VAR T : Tab ; debut, milieu, fin: Integer);
VAR
        aux : Tab;
        i, j, k, l, taille : Integer;
BEGIN
        i := debut;
        j := milieu+1;
        taille := fin - debut +1;
        For   k:=1 To taille Do
        Begin
           If ((j > fin) OR (I < = milieu) AND (t[i] < t[j]))
              Then Begin
                        aux[k] := t[i] ;
                        i := i +1;
                   End  Else
                   Begin
                        aux[k] := t[j] ;
                        j := j +1;
                   End;
        End;

        {Recopier  aux  dans T}
        l := 1;
        For   k := debut To fin Do
        Begin
                t[k] := aux[l];
                l := l +1;
        End;
END;


PROCEDURE tri_fusion (VAR T : Tab ; N : Integer);
VAR
        debut, fin, i, milieu : Integer;
BEGIN
        i := 1;
        While (i <n) Do
        Begin
                debut := 1;
                 fin := debut + i + i-1;  {Le pas}
```

```
                While (debut + i-1 <= N) Do
                Begin
                        fin := (fin + debut) - 1;
                        If (fin > N) Then fin := N;
                       Fusion (t, debut, debut + i-1, fin);
                        debut := debut + i + i;
              End;
    I := i+1;
   End;
END;


{-------------------------------------------------------------------------------------}
PROCEDURE Affiche_Tab (T : Tab ; N : Integer) ;
VAR   i : Integer ;
BEGIN
        For   i := 1  To   N  Do
                WriteLn (T[i] : 8 : 3) ;
END ;


{= = = = = = = = = = = = = = = = = = P P = = = = = = = = = = = = = = = = = = = = = = }
BEGIN
        Saisie (N) ;
        Remplir_Hasard (T, N) ;
        WriteLn ('Tableau non trié ');
        Affiche_Tab (T, N) ;

        Tri_Fusion (T, N) ;
        WriteLn ('Tableau trié ' );
        Affiche_Tab (T, N) ;
END.
```