Les Algorithmes De Tri

Trie à Bull

```
0) Def Proc bull(var t :tab; n :entier)
1) Repéter
        Permut ← vrais
        Pour i de 1 à n-1 faire
              Si(t[i]<t[i+1])alors
                    Proc permuter(t[i],t[i+1])
                    Permut ← faux
              Fin si
        Fin pour
   Jusqu'à (permut)
2) Fin bull
```

Procédure Permuter

```
0) Def Proc permuter(var A,B :entier)
1) Vaux ← A
2) A ← B
3) B ← C
4) Fin permuter
```

Tri par Sélection

```
0) Def Proc selection(var t :tab; n :entier)
1) Pour i de 1 a n faire
        Pmin (ou pma) ← Fn mini(i,n)(ou Fn maxi(i,n))
        Si (i<>pmin(ou pmax pour maximum)) alors
              Proc permuter(t[i],t[pmin(ou pmax)])
        Fin si
   Fin pour
2) Fin selection
```

Fonction Mini

```
0) Def Fn mini(a,b :entier):entier
1) [pmin←a]
        pour i de a+1 à b faire
              Si(t[pmin]>t[i])alors
                    Proc permuter(t[pmin],t[i])
              Fin si
        Fin pour
2) Mini ← pmin
```

Procédure Permuter

```
0) Def proc permuter(var A,B :entier)
1) Vaux ← A
2) A ← B
3) B ← C
4) Fin permuter
```

Tri par Insertion

```
0) Def Proc insertion (var t :tab; n :entier)
1) Pour i de 2 à n faire
        Vaux ← t[i]
        J ← i
        Tantque (j>1) et (t[j-1] < (ou >) vaux) faire
              t[j] ← t[j-1]
              j ← j-1
        fin tantque
   fin pour
2) Fin insertion
```

Tri Shell

```
0) Def Proc shell (var t :tab; n :entier)
1) [pas←1]Tantque (pas<n) faire
        Pas ← pas*3+1
   Fin tantque
2) Pour i de pas+1 à n faire
        Vaux ← t[i]
        J  ← i
        Tantque (j>pas) et (t[j-pas]< (ou >) vaux) faire
              T[j] ← t[j-pas]
              J ← j-p
        Fin tantque
        T[j] ← vaux
   Fin pour
3) Fin shell
```

Classement

```
0) Def Proc classement (var t :tab; n :entier)
1) Pour i de 1 à n faire
        S ← Fn compter (t,n,t[i])
        R[s+1] ← t[i]
   Fin pour
2) Fin classement
```

Fonction Compter

```
0) Def Fn classement (var t :tab; n,x :entier) :entier
1) [s←0]Pour i de 1 à n faire
        Si (t[i]< (ou >) x) alors
              s ← s+1
        Fin si
   Fin pour
2) Compter ← s
Fin compter
```

Procédure fusion

```
0) Def Proc fusion (var t :tab; d,mil,f :entier)
1) i←d
2) j←mil+1
3) k←d
4) Tantque (i<=mil) et (j<=f) faire
        Si(t[i]< (ou>) t[j])alors
              p[k]←t[i]
              i←i+1
        Sinon
              p[k]←t[j]
              j←j+1
        fin si
        k←k+1
   fin tantque
5) Si(i>mil)alors
        Pour l de i à f faire
              p[k]←t[l]
              k←k+1
        fin pour
   Sinon
        Pour l de i à mil faire
              p[k]←t[l]
              k←k+1
        fin pour
   fin si
6) Pour l de d à f faire
        t[l]←p[l]
   fin pour
7) Fin fusion
```

Tri Fusion

```
0) Def Proc tri_fusion (var t :tab; d,f:entier)
1) Si (d<f) alors
        mil← (d+f)div 2
        Proc tri_fusion(t,d,mil)
        Proc tri_fusion(t,mil+1,f)
        Proc fusion(t,d,mil,f)
   fin si
2) Fin tri_fusion
```

Chapitre 3

Les algorithmes de tri

Durée : 10 heures

Type : Théorique et Pratique

I- Introduction

Trier un tableau signifie ordonner les éléments de ce tableau selon l'ordre croissant ou décroissant. L'ordre peut être numérique ou alphabétique ou alphanumérique.

En 3ème année, vous avez étudié les 3 méthodes de tri les plus utilisés et qui sont :

- Tri par sélection
- Tri à bulles
- Tri par insertion

II- Tri par insertion

Activité 1

1. Donner le principe du tri par insertion.
2. En déduire une analyse et un algorithme de la procédure **TRI_INS (n : entier ; var T : VECT)**.
3. Ecrire un programme Pascal nommé **Tri_insertion**, qui permet :
   - ✓ De remplir aléatoirement un tableau par n entiers ($4 \leq n \leq 25$),
   - ✓ De trier ce tableau par la méthode de tri par insertion,
   - ✓ D'afficher le contenu du tableau avant et après le tri par ordre croissant.

Solution

1. Le principe général est le suivant :
   - Parcourir le tableau à trier à partir du 2ème élément et essaye de placer l'élément en cours à sa bonne place parmi les précédents.
   - De cette façon, la partie dans laquelle on cherche à placer l'élément en cours est trié.
   - L'opération de placement peut nécessiter le décalage d'un bloc d'un pas vers l'avant.
2. **Analyse de la procédure TRI_INS**

```
DEF PROC TRI_INS (N : entier ; VAR T : VECT)
Résultat = T
T= [ ]   Pour i de 2 à N faire
                TEMP←T[i]
                j ←i
                Proc DECALER(T, j, TEMP)
                T[j] ← TEMP
            Fin Si
         Fin Pour
Fin TRI_INS
```

- Ranger la valeur de T[i] dans la variable TEMP
- décaler d'un cran vers la droite les valeurs de T[i-1], T[i-2], … jusqu'à arriver à une valeur qui est inférieure à T[i]. (procédure DECALER).
- Affecter au dernier élément décaler la valeur de TEMP.

T.D.O.L

objets	Type /Nature	Rôle
i	Entier	Compteur/indice d'itération
TEMP	Entier	Variable intermédiaire nécessaire à l'insertion
j	entier	Position d'insertion
DECALER	Procédure	Décale les éléments d'un tableau vers la droite

Algorithme de la procédure TRI_INS

0) DEF PROC TRI_INS (N : entier ; VAR T : VECT)
1) Pour i de 2 à N faire
   TEMP←T[i]
   j ←i
   DECALER(T, j, TEMP)
   T[j] ← TEMP
   Fin Si
   Fin Pour
2) Fin TRI_INS

Analyse de la procédure DECALER

DEF PROC DECALER (VAR T : VECT ; VAR j: entier ; i : entier)
Résultat = DECALER
DECALER = [ ] Tant que (T[j-1]>i) et (j>1)Faire
   T[j] ← T[j-1] {l'action de décalage'}
   j ← j-1
   Fin Tant que
Fin DECALER

Algorithme de la procédure DECALER

0) Début procédure DECALER (VAR T : VECT ; VAR j : entier ; i : entier)
1) Tant que (T[j-1]>i) et (j>1)Faire
   T[j] ← T[j-1] {l'action de décalage}
   j ← j-1
   Fin Tant que
2) Fin DECALER

Application1

Proposer une analyse récursive de la procédure **TRI_INS (n : entier ; var T : VECT)**.

1ère méthode

Analyse récursive de la procédure TRI_INS

DEF PROC TRI_INS (N : entier ; VAR T : VECT)
Résultat = T
T= [ ] Si n > 1 Alors
   Tri_Ins (n – 1, T);
   DECALER(T, n, T[n])
   Fin Si
Fin TRI_INS

Analyse de la procédure DECALER

```
DEF PROC DECALER (VAR T : VECT ; VAR j: entier ; i : entier)
Résultat = DECALER
DECALER = [ ]  Tant que (T[j-1]>i) et (j>1)Faire
                        T[j] ← T[j-1]          {l'action de décalage'}
                        j ← j-1
                        T[j]←i
                Fin Tant que
Fin DECALER
```

2ème méthode

Analyse récursive de la procédure TRI_INS

```
DEF PROC TRI_INS (N : entier ; VAR T : VECT)
Résultat = T
T= [ ] Si n > 1 Alors
        Tri_Ins (T,n - 1);
                Si T[n] < T[n - 1] Alors
                        Aux ← T[n]
                        i ← n
                        Répéter
                                T[i] ← T[i – 1]
                                i ← i – 1
                        jusqu'à (i = 1) ou (aux > t[i – 1])
                T[i] ← aux
        Fin Si
Fin TRI_INS
```

T.D.O.L

objets	Type /Nature	Rôle
i	Entier	Compteur/indice d'itération
aux	Entier	Variable intermédiaire nécessaire à l'insertion

3ème méthode

Analyse récursive de la procédure TRI_INS

```
DEF PROC TRI_INS (G, N : entier ; VAR T : VECT)
Résultat = T
T= [ ] Si G ≤ N Alors
                Si T[G] < T[G - 1] Alors
                        DECALER(T, G-1, T[G])
                        TRI_INS (G+1, N, T)
                Fin Si
        Fin Si
Fin TRI_INS
```

T.D.O.L

objets	Type /Nature	Rôle
DECALER	Procédure	Décale les éléments d'un tableau vers la droite

Analyse récursive de la procédure DECALER

```
DEF PROC DECALER (VAR T : VECT ; VAR j : entier ; i : entier)
Résultat = DECALER
DECALER =[ ]   Si j ≥ 1 Alors
                        Si T[j] > i Alors
                                T[j+1] ← T[j]
                                DECALER(T, j-1, i)
                        Sinon
                                T[j+1] ← i
                Sinon
                        T[j+1] ← i
                        Fin Si
                Fin Si
Fin DECALER
```

III- Tri Shell

1. Introduction

Activité 1

Q : Qui peut me rappeler de principe de tri par insertion ?
R : Le principe général est le suivant :

- Parcourir le tableau à trier à partir du $2^{ème}$ élément et essaye de placer l'élément en cours à sa bonne place parmi les précédents.
- De cette façon, la partie dans laquelle on cherche à placer l'élément en cours est trié.
- L'opération de placement peut nécessiter le décalage d'un bloc d'un pas vers l'avant.

Soit le tableau T suivant :

87	13	26	31	38	47	53	59	62	10

Essayer d'appliquer le principe de tri par insertion jusqu'à l'avant dernier élément du T
Q : Qu'est–ce qu'il faut pour insérer le dernier élément de T à sa place ?
R : Il faut décaler tous les éléments du T.

Inconvénient : Pour ramener un élément de la fin vers la tête, il faut décaler tous les éléments plus grands.
Donc la question qui ce pose maintenant à les élèves c'est quoi la solution ?
R : Donc c'est de chercher une nouvelle méthode de tri dont on prend en considération cette inconvénient c'est la méthode de tri Shell. Enfaîte c'est une amélioration du tri par insertion.
D'où le titre de notre leçon aujourd'hui.

Activité 2

Reprenons l'activité précédente et essayer de réaliser sur le même tableau le tri Shell. Sachant que cette méthode est basée sur le calcul de pas.

Q : C'est quoi le pas ?

R : Dans ce tri, les éléments sont décalés de plusieurs éléments. La distance qui les sépare est appelée **"pas"**.

Q : la question qui se pose maintenant comment on va choisir le pas ?

R : Donald L.Shell proposa en 1959 la suite d'incréments vérifiant :

$$u(n+1) = 3 * u(n) + 1 \quad avec\ u(0) = 0$$

Calcul du pas pour un tableau de taille n = 100 :

u(0) = 0
u(1) = 3 * u(0) + 1 = 3 * 0 + 1 = 1
u(2) = 3 * u(1) + 1 = 3 * 1 + 1 = 4
u(3) = 3 * u(2) + 1 = 3 * 4 + 1 = 13
u(4) = 3 * u(3) + 1 = 3 * 13 + 1 = 40
u(5) = 3 * u(4) + 1 = 3 * 40 + 1 = 121

Les valeurs successives que le pas prendra seront donc :

pas1 = 121 DIV 3 = 40
pas2 = pas1 DIV 3 = 40 DIV 3 = 13
pas3 = pas2 DIV 3 = 13 DIV 3 = 4
pas4 = pas3 DIV 3 = 4 DIV 3 = 1

Les grandes étapes de l'algorithme :

53	31	10	87	13	59	62	26	47	38
13	31	10	87	53	59	62	26	47	38
13	31	10	87	53	59	62	26	47	38
13	31	10	87	53	59	62	26	47	38
13	31	10	26	53	59	62	87	47	38
13	31	10	26	47	59	62	87	53	38
13	31	10	26	47	38	62	87	53	59

Maintenant le **pas = 1** donc on passe à un **tri par insertion**

13	31	10	26	47	38	62	87	53	59
13	31	10	26	47	38	62	87	53	59
10	13	31	26	47	38	62	87	53	59
10	13	26	31	47	38	62	87	53	59
10	13	26	31	47	38	62	87	53	59
10	13	26	31	38	47	62	87	53	59
10	13	26	31	38	47	62	87	53	59
10	13	26	31	38	47	62	87	53	59
10	13	26	31	38	47	53	62	87	59
10	13	26	31	38	47	53	59	62	87

Q : Essayer de dégager le principe de tri Shell ?
R : Donald L. Shell proposa, en 1959, une variante du tri par insertion. Dans ce tri, les éléments sont décalés de plusieurs éléments.
La distance qui les sépare est appelée "**pas**". A chaque étape le tableau est affiné (mieux organisé) et le pas réduit. Lorsque le pas est de 1, cela revient à un tri par insertion.
Le pas est généralement calculé à partir de la formule suivante :

$$u(n+1) = 3 * u(n) + 1 \quad avec\ u(0) = 0$$

Q : Que constatez-vous ?
R : Donc on a appliquer une nouvelle méthode de tri sur le tableau a fin de minimiser le nombre de décalage qu'on a fait dans la méthode de tri par insertion d'où cette méthode de tri (qu'on a appeler Shell qui porte le nom de son constructeur) est une excellente amélioration de tri par insertion.

2. Applications (Résolution du problème)

Application1

On se propose d'écrire un programme Pascal nommé **TRI_SHELL** qui permet :

- ✓ De remplir un tableau aléatoirement par N entiers (N compris entre 4 et 25),
- ✓ De trier ce tableau par la méthode de tri Shell,
- ✓ D'afficher le contenu du tableau avant et après le tri par ordre croissant.

Questions :

1) Analyser ce problème en le décomposant en modules.
2) Analyser chaque module proposé.
3) Déduire les algorithmes correspondants.
4) Traduire la solution obtenue en Pascal.

- **Analyse & Algorithme du programme principal**
  - **Analyse du programme TRI_SHELL**

Résultat = Proc Affiche_Tab (T, N)
T= Proc Shell (T, N)
T= Proc Remplir_Hasard (T, N)
N= Proc Saisie (N)
Fin TRI_SHELL

Tableau de déclaration des nouveaux types

Type
Vect = tableau de 25 entiers

Tableau de déclaration des objets globaux

Objets	Type/Nature	Rôle
N	Entier	Nombre d'éléments du tableau
T	Tab	Tableau des entiers
Saisie	Procédure	Saisie et test de N
Remplir_hasard	Procédure	Remplit au hasard le tableau T
Affiche Tab	Procédure	Affiche le contenu du tableau
Shell	Procédure	Trier le tableau par la méthode de tri Shell

- **Algorithme du programme TRI_SHELL**

```
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
```

- **Analyses & Algorithmes des modules proposés**
- **Analyse de la procédure Saisie**

```
DEF PROC Saisie (VAR N : Entier)
Résultat = N
N= [ ]  Répéter
                N = Donnée ("Entrer la taille du tableau : ")
        Jusqu'à (4 ≤ N ≤ 25)
Fin Saisie
```

- **Algorithme de la procédure Saisie**

```
0) DEF PROC Saisie (VAR N : Entier)
1) Répéter
                Ecrire ("Entrer la taille du tableau : ")
                Lire (N)
   Jusqu'à (4<N<25)
2) Fin Saisie
```

- **Analyse de la procédure Remplir_Hasard**

```
DEF PROC Remplir_Hasard (VAR T : Vect ; N : Entier)
Résultat = T
```

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

```
T=[Randomize]
        Pour i de 1 à N Faire
                T[i] ←Hasard(99)
         Fin Pour
Fin Remplir_Hasard
```

T.D.O.L

Objets	Type/Nature	Rôle
i	entier	compteur

- **Algorithme de la procédure Remplir_Hasard**

```
0) DEF PROC Remplir_Hasard (VAR T : Vect ; N : Entier)
1) Randomize
2) Pour i de 1 à N Faire
        T[i] ←Hasard (99)
   Fin Pour
3) Fin Remplir_Hasard
```

- **Analyse de la procédure Shell**

```
DEF PROC Shell (N : entier ; var T : Vect)
Résultat = T_trié
T_trié = [p ←0]
    (* la suite récurrente : Un+1 = 3.Un + 1        *)
    (* Tel que Un < n (Nombre d'éléments du tableau)  *)
    Tant que (p < N) faire
        p ← (3*p+1)
    Fin Tant que
    Tant que (p ≠ 0) faire
        p ← (p div 3)
        Pour i de p à N Faire
            k ← T[i]        (* Valeur à insérer *)
    (* Recherche de la position d'insertion *)
            j← i
            Tant que (j>p-1) et (T[j-p]> k) faire
                T[j] ← T[j-p]
                j ←j-p
            Fin Tant que
    (* Insertion de la valeur à son emplacement *)
        T[j] ← k
        Fin Pour
    Fin Tant que
Fin Shell
```

T.D.O.L

Objets	Type/Nature	Rôle
i, j	entier	compteur
p	Entier	Représente le pas
k	Entier	Valeur à insérer

- **Algorithme de la procédure Shell**

```
0) DEF PROC Shell (N : entier ; var T : Vect)
1) p ←0
   Tant que (p < N) faire
        p ← (3*p+1)
```

```
        Fin Tant que
    2)  Tant que (p ≠ 0) faire
            p ← (p div 3)
            Pour i de p à N Faire
                k ← T[i]
                j← i
                Tant que (j>p-1) et (T[j-p]> k) faire
                    T[j] ← T[j-p]
                    j ←j-p
                Fin Tant que
            T[j] ← k
            Fin Pour
        Fin Tant que
    3)  Fin Shell
```

- **Analyse de la procédure Affiche_Tab**

```
DEF PROC Affiche_Tab (T : Vect ; N : Entier)
Résultat = T.
T=[ ]   Pour i de 1 à N Faire
            Ecrite (T[i])
        Fin Pour
Fin Affiche_Tab
```

T.D.O.L

Objets	Type/Nature	Rôle
i	entier	compteur

- **Algorithme de la procédure Affiche_Tab**

```
0)  DEF PROC Affiche_Tab (T : Vect ; N : Entier)
1)  Pour i de 1 à N Faire
        Ecrite (T[i])
    Fin Pour
2)  Fin Affiche_Tab
```

Application2

Proposer une analyse récursive de la procédure Shell (N, h: entier ; var T : Vect)

- **Analyse récursive de la procédure Shell**

```
DEF PROC Shell (N, h: entier ; var T : Tab)
Résultat = T
T= [ ] Si h > 0 Alors
            Si n>h Alors
                Shell(n-h, h, T)
                Si T[n] < T[n - h] Alors
                    aux← T[n]
                    i ←n
                Répéter
```

```
                    T[i] ←T[i – h]
                    i ←i – h
                jusqu'à (i = h) Or (aux > T[i – h])
                T[i] ←aux
                Fin Si
            Fin Si
            Shell(n, h div 3, T)
    Fin Si
```

T.D.O.L

Objets	Type/Nature	Rôle
i	entier	compteur
aux	Entier	Représente le pas

Remarque:

Tester cette procédure sur des tableaux de petites tailles, car si n'est pas le cas le nombre des appels devient important et en aura le problème de débordement de la pile (la limite technique de la récursivité est la mémoire).

On peut augmenter la taille du tableau test on augmentant la taille de la pile (Option\Compilateur\Paramètres mémoire\Taille pile).