CHAPITRE

Chapitre 6

Les traitements avancés

Objectifs :

Utiliser l'analyse modulaire pour résoudre des problèmes de tri et de recherche.

Plan du chapitre :

Leçon 1

Methodes de tri

Objectifs spécifiques :

- Connaître les différentes méthodes de tri.
- Utiliser les différentes méthodes de tri pour résoudre des problèmes.
- Présenter des solutions sous forme d'un algorithme puis d'un programme.

Plan de la leçon :

Leçon 1

Methodes de tri

Une place pour chaque chose et chaque chose à sa place.
Samuel SMILES

I. Introduction

Le problème de tri est un classique de l'informatique ; les méthodes de tri sont utilisées dans différentes applications.
Par exemple :
- classer les élèves par ordre alphabétique ou par ordre de mérite
- mettre en ordre un dictionnaire
- trier l'index d'un livre
- afficher une liste triée d'un correcteur d'orthographe
- …

Dans cette leçon, nous exposerons la problématique du tri de séquences de valeurs et nous nous limiterons à l'étude de quelques méthodes de tri parmi les plus connus, notamment le tri par sélection, le tri à bulles et le tri par insertion.

Pour chaque méthode de tri, nous présentons :
- le principe
- l'analyse du problème, la décomposition en modules, l'élaboration des algorithmes et la traduction en Pascal.

Activité 1

On se donne une suite de N nombres entiers, et on se propose de les trier par ordre croissant. Ainsi, pour N=9, la suite [17,2,9,8,2,10,12,22,8] deviendra [2,2,8,8,9,10,12,17,22]

Définition

Un algorithme de tri est une suite finie d'instructions servant à réordonner une séquence d'éléments suivant un critère fixé à priori. Ce critère est en fait une relation d'ordre total sur les éléments à trier.

La conception d'un algorithme de tri dépend beaucoup plus du support matériel de la séquence de valeurs à trier. Celle-ci peut se trouver en mémoire centrale ou sur une mémoire secondaire.

Ces supports ont des caractéristiques physiques assez différentes :
Les mémoires centrales sont rapides (en nanosecondes) mais d'une taille limitée (en mégaoctets).
Les mémoires secondaires, par contre, sont lentes (en microsecondes) mais de grande taille (en gigaoctets). Par conséquent, les algorithmes de tri vont devoir en tenir compte.

Les algorithmes de tri que nous allons définir traitent des tableaux situés dans la mémoire centrale.

Remarques :

- Il faut faire la distinction entre le tri d'un grand nombre d'éléments et le tri de quelques éléments.

- On peut trier autres données que des entiers. Il suffit de disposer d'un domaine de valeurs muni d'une relation d'ordre totale. On peut donc trier des caractères, des mots en ordre alphabétique...

II. Les méthodes de tri

Dans la suite de ce cours, on choisit de trier les séquences par ordre croissant c'est-à-dire du plus petit au plus grand relativement à un ordre total noté ≤.

II.1 Le tri par sélection

II.1.1 Principe

L'algorithme de tri le plus simple est le *tri par sélection*. Il consiste à trouver l'emplacement de l'élément le plus petit du tableau. Une fois cet emplacement trouvé, on compare son contenu avec T[1] et s'ils sont différents, on permute l'élément de l'emplacement trouvé par l'élément de la première position T[1]. Après ce parcours le premier élément est bien placé.

On recommence le même procédé pour le reste du tableau (T[2],..,T[N]), ainsi on recherche le plus petit élément de cette nouvelle partie du tableau et on l'échange éventuellement avec T[2]. Et ainsi de suite ... jusqu'à la dernière partie du tableau formée par les deux derniers éléments (T[N-1], T[N]).

N.B.

- La recherche du plus petit élément d'un tableau sera assurée par une fonction renvoyant la position du minimum. Ce minimum est éventuellement répété plusieurs fois dans T ; on décidera de choisir le premier.
- L'échange de deux éléments d'un tableau sera assurée par une procédure de permutation.

II.1.2 Résolution du problème : tri par sélection

Soit à trier un tableau T de n entiers initialement dans un ordre quelconque en ordre croissant en utilisant la méthode de tri par sélection.

Exemple :

On considère le tableau T contenant les 10 éléments suivants :

T	17	5	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

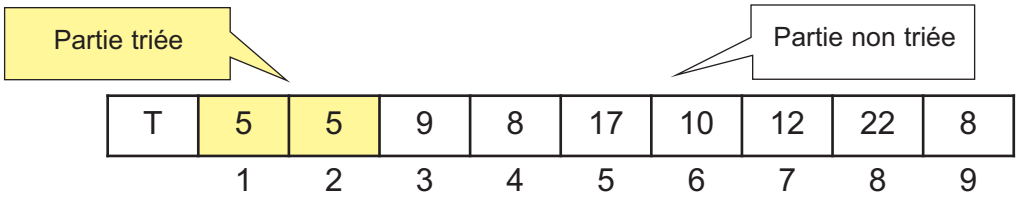
1) - On se pointe à la 1ère case du tableau et on parcourt la totalité de T pour repérer l'indice du minimum c'est-à-dire l'entier 5, ce minimum est répété plusieurs fois dans T, on décidera de choisir le premier c'est-à-dire d'indice 2.

T	17	**5**	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

- On permute T[1] et ce minimum.
- Après cette opération, on est sûr que T[1] contient la plus petite valeur de T et est donc à sa bonne place.

T	5	17	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

2) - On cherche la valeur minimale dans (T[2], T[3],…,T[N]) : cette valeur étant 5 et son indice est 5.
- On permute T[2] et ce minimum.



3) - On cherche la valeur minimale dans (T[3],…,T[N]) : cette valeur étant 8, et son indice est 4. Ce minimum est répété plusieurs fois dans T, on décidera de choisir le premier.

T	2	2	9	**8**	17	10	12	22	8
	1	2	3	4	5	6	7	8	9

- On permute T[3] et ce minimum.

T	2	2	8	9	17	10	12	22	8
	1	2	3	4	5	6	7	8	9

4) - On cherche la valeur minimale dans (T[4],…,T[N]) : cette valeur étant 8 et son indice est 9.

T	2	2	8	9	17	10	12	22	**8**
	1	2	3	4	5	6	7	8	9

- On permute T[4] et ce minimum.

T	2	2	8	8	17	10	12	22	**9**
	1	2	3	4	5	6	7	8	9

5) - On cherche la valeur minimale dans (T[5],…,T[N]) : cette valeur étant 9 et son indice est 9.

T	2	2	8	8	17	10	12	22	**9**
	1	2	3	4	5	6	7	8	9

- On permute T[5] et ce minimum.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

6) - On cherche la valeur minimale dans (T[6],…,T[N]) : cette valeur étant 10 et son indice est 6.

T	2	2	8	8	9	**10**	12	22	17
	1	2	3	4	5	6	7	8	9

- Puisque la valeur minimale correspond à la première case de la partie non triée du tableau T, l'élément reste dans la même position.

7) - On cherche la valeur minimale dans (T[7],…,T[N]) : cette valeur étant 12 et son indice est 7.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

- Puisque la valeur minimale correspond à la première case de la partie non triée du tableau T, l'élément reste dans la même position.

8) - On cherche la valeur minimale dans (T[8],…,T[N]) : cette valeur étant 17 et son indice est 9.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

- On permute T[9] et ce minimum.

T	2	2	8	8	9	10	12	17	22
	1	2	3	4	5	6	7	8	9

Résultat : tableau trié

T	2	2	8	8	9	10	12	17	22
	1	2	3	4	5	6	7	8	9

Remarque :

Il faut faire la distinction entre le tri d'un grand nombre d'éléments et le tri de quelques éléments.

Analyse

On choisira à chercher la position du premier minimum quand il y en aura plusieurs occurrences.
Le tableau T sera de type TAB.

	DEF PROC tri_sélection (n : Entier ; VAR T : TAB)	
S	**L.D.E.**	**O.U.**
1	**Résultat** = T_Trié T_Trié = [ ] **Pour** i de 1 à n-1 **Faire** [ppm ←**FN** premposmin (T,n,i)] **Si** (T[ppm] ≠ T[i] **Alors** PROC permut(T[ppm), T[i]) **FinSi** **FinPour**	i ppm premposmin permut
2	**Fin tri_séléction**	

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableaux de déclaration des objets de la procédure tri_sélection

Objet	Type/nature	Rôle
i	Entier	Compteur
ppm	Entier	Contiendra la première position du minimum
premposmin	Fonction entière	Renvoie la première position du minimum
permut	Procédure	Permute le contenu de deux variables

	DEF FN premposmin (A: TAB, n, m : Entier ;) : Entier	
S	**L.D.E.**	**O.U.**
2	**Résultat** = (premposmin ← posmin)	i
1	posmin = [posmin ← m] **Pour** j de m+1 à n **Faire**	ppm
	[ ] **Si** A[posmin] > A[j] **Alors**	premposmin
	posmin ← j	permut
	FinSi	
	FinPour	
3	**Fin premposmin**	

Tableaux de déclaration des objets de la fonction premposmin

Objet	Type/nature	Rôle
posmin	Entier	Contient la position du minimum
j	Entier	compteur

	DEF PROC permut (VAR x, y : Entier)	
S	**L.D.E.**	**O.U.**
	Résultat = (x, y)	aux
2	x ← y	x
3	y ← aux	y
1	aux ← x	
4	**Fin permut**	

Tableaux de déclaration des objets de la procédure permut

Objet	Type/nature	Rôle
aux	Entier	Variable auxiliaire nécessaire à la permutation

Algorithmes

```
0)  DEF PROC  tri_sélection (n : Entier ; VAR T : TAB)
1) Pour  i de 1  à  n-1  répéter
        [ppm ← FN premposmin (T, n, i)]  Si  T[ppm] ≠ T[i]  Alors
                                              PROC permut ( T[ppm], T[i] )
                                        FinSi
   FinPour
2) Fin PROC tri_sélection
```

```
0)  DEF FN premposmin (A: TAB, n , m : Entier) : Entier
1) [ posmin ← m]  Pour  j de m+1  à  n  répéter
                      Si A[posmin] > A[j]  Alors
                                          posmin ← j
                      FinSi
                FinPour
2) premposmin  ← posmin
3) Fin premposmin
```

```
0) DEF PROC permut ( VAR x, y : Entier)
1) aux ← x
2)  x  ← y
3)  y  ← aux
4) Fin permut
```

Traduction en Pascal

On suppose que T est un tableau de type TAB.

```
PROCEDURE tri_sélection (n : INTEGER ; VAR T : TAB);
VAR  ppm,i :INTEGER;

PROCEDURE permut(VAR x, y : INTEGER);
VAR  aux : INTEGER;
BEGIN
     aux := x;
     x := y;
     y := aux;
END;

FUNCTION premposmin (A : TAB; n, m : INTEGER):INTEGER;
VAR posmin, j :INTEGER;
BEGIN
     posmin := m;
     FOR j:=m+1 TO n DO
       BEGIN
         IF A[posmin] > A[j] THEN posmin := j;
       END;
     premposmin := posmin ;
END;

BEGIN

FOR i:=1 TO n-1 DO
       BEGIN
         ppm := premposmin (T, n, i) ;
         IF T[ppm]<>T[i] THEN  permut(T[ppm], T[i]);
       END;
END;
```

Remarque :

Il est facile de compter le nombre d'opérations nécessaires pour trier tout le tableau T. A chaque itération, on démarre à l'élément T[i] et on le compare successivement à T[i+1], T[i+2], ..., T[n]. On fait donc n-i comparaisons.
On commence avec i=1et on finit avec i=n-1.
Donc, on fait (n-1) + (n-2) + … + 2 + 1=n(n-1)/2 comparaisons, et au maximum n-1 permutations.

II.2 Le tri à bulles

II.2.1 Principe

Le tri à bulles consiste à faire remonter le plus grand élément du tableau en comparant les éléments successifs. C'est-à-dire qu'on va comparer le 1er et le 2^{ième} élément du tableau, s'ils ne sont pas dans le bon ordre, on les permute, on passe ensuite au 2^{ième} et 3^{ième}, puis au 3^{ième} et 4^{ième} et ainsi de suite jusqu'au (n-1)^{ième} et le n^{ième} éléments.

A la fin du premier parcours, on aura pousser le plus grand élément du tableau vers sa place finale qui est le n^{ième} élément du tableau. On recommence cette opération en parcourant de 1 à n-1 puis de 1 à n-2 et ainsi de suite.

On arrête quand la partie à trier est réduite à un seul élément ou que le tableau est devenu trié et dans ce cas on doit avoir un indicateur qui donne cette information. En effet, l'idée est de vérifier si lors du dernier parcours aucune permutation n'a été faite ce qui signifie que le tableau est devenu trié.

II.2.2 Résolution du problème : tri à bulles

Soit à trier un tableau T de n entiers par ordre croissant en utilisant la méthode de tri à bulles.

Exemple

On considère le tableau T contenant les 5 éléments suivants :

T	3	1	9	0	3
	1	2	3	4	5

1) - On se pointe à la 1ère case du tableau et on compare T[1] et T[2].

T	3	1	9	0	3
	1	2	3	4	5

- Puisque 3 >1, on les permute.

T	1	3	9	0	3
	1	2	3	4	5

2) On compare T[2] et T[3]. Puisque ils sont dans le bon ordre on ne fait rien.

T	1	3	9	0	3
	1	2	3	4	5

3) - On compare T[3] et T[4].

T	1	3	9	0	3
	1	2	3	4	5

- Puisque 9>0, on les permute.

T	1	3	0	9	3
	1	2	3	4	5

4) - On compare T[4] et T[5].

T	1	3	0	9	3
	1	2	3	4	5

- Puisque 9>3, on les permute.

T	0	1	3	3	9
	1	2	3	4	5

On a parcouru les éléments (T[1],T[2], … T[n]) en permutant toute paire d'éléments consécutifs (T[j-1], T[j]) non ordonnés. Ainsi après un parcours, l'élément maximum 9 se retrouve en T[n]. En effet au fur et à mesure des échanges, la méthode fait avancer le plus grand élément rencontré.

A la fin de la première boucle, T[n] contient le plus grand élément du vecteur et cette $n^{ème}$ composante ne doit plus être prise en compte dans la suite du tri puisqu'elle est à sa bonne place.

Il suffit de recommencer ensuite le même processus avec les éléments de T allant de la première composante à n-1, et ainsi de suite on recommence avec les éléments (T[1],T[2], … T[n-2]) jusqu'à épuisement de tous les sous tableaux.

Analyse

	DEF PROC tri_bulles (**n : Entier ; VAR T : TAB**)	
S	**L.D.E.**	**O.U.**
1	**Résultat** = T_Trié Trié = [ ] **Répéter** [Echange ← faux] **Pour** i de 1 à n-1 **faire** [ ] **Si** (T[i]>T[i+1]) **alors** PROC Permut(T[i], T[i+1]) PROC Echange ← vrai **Fin SI** **Fin Pour** **n ← n-1** **Jusqu'à** (n=1) ou non (Echange)	Echange i Permut
2	**Fin tri_bulles**	

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableau de déclaration des objets de la procédure tri_bulles

Objet	Type/nature	Rôle
Echange	Bouléen	Reçoit la valeur vrai si une permutation a u lieu
i	Entier	Compteur
T	TAB	Tableau d'entiers
permut	Procédure	Permute le contenu de deux variables

	DEF PROC permut **(VAR x, y : Entier)**	
S	**L.D.E.**	**O.U.**
	Résultat = (x, y)	aux
2	x ← y	x
3	y ← aux	y
1	aux ← x	
4	**Fin permut**	

Tableau de déclaration des objets de la procédure permut

Objet	Type/nature	Rôle
aux	Entier	Variable auxiliaire nécessaire à la permutation

Algorithmes

```
0)  DEF PROC tri_bulles (n : Entier ; VAR T : TAB)
1) Répéter
     Echange ← faux
      Pour i de 1 à n-1 faire
         Si (T[i]>Tt[i+1]) Alors
                              PROC Permut(T[i], T[i+1])
                              PROC Echange ← vrai
         Fin SI
       Fin Pour
     n ← n-1
  Jusqu'à (n=1) ou non (Echange)
2) Fin PROC tri_bulles
```

0) DEF PROC ***permut*** *(VAR x, y : Entier)*
1) aux ← x
2) x ← y
3) y ← aux
4) Fin permut

Traduction en Pascal

Le programme pascal ci-dessous comporte une procédure remplissage et une procédure d'affichage permettant de mettre en œuvre la procédure du tri à bulles.

```
PROGRAM TRI_A_BULLES;
USES WINCRT;
TYPE TAB = ARRAY [1..50] OF INTEGER;
VAR
   T: TAB;
   n: INTEGER;
   i: INTEGER;

PROCEDURE RemplirT(var n:INTEGER ; var T:TAB);
BEGIN
   WRITE('Entrer la taille du tableau : ');
   READLN(n);
   FOR i := 1 TO n DO
    BEGIN
      WRITE('Entrer élément ',i,' : ');
      READLN (T[i]);
    END;
END;

PROCEDURE tri_bulles (n:INTEGER ; var T: TAB);
   VAR
    i: INTEGER;
    Echange: BOOLEAN;

PROCEDURE Permut(var X,Y:INTEGER);
   VAR
    TEMP:INTEGER;
      BEGIN
         TEMP := x;
         x := y;
         y := TEMP;
      END;
```

```
BEGIN
  REPEAT
    Echange:=false;
    FOR i := 1 TO n-1 DO
        BEGIN
          IF T[i] > T[i+1] THEN
                                BEGIN
                                  Permut(T[i],T[i+1]);
                                  Echange:= true;
                                END;
        END;
       n := n-1 ;
     UNTIL (n=1) OR NOT(Echange);
  END;

  PROCEDURE AfficherT(n:INTEGER ; T:TAB);
  BEGIN
    WRITELN('Résultat du programme TRI_A_BULLES :');

    FOR i := 1 TO n DO
     WRITELN('T[', i, ']: ', T[i])
  END;

BEGIN
 RemplirT(n,T);
 TRI_Bulles(n,T);
 AfficherT(n,T);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 6
Entrer élément 1 : 14
Entrer élément 2 : 19
Entrer élément 3 : 4
Entrer élément 4 : 7
Entrer élément 5 : 14
Entrer élément 6 : 2
Résultat du programme TRI_A_BULLES :
T[1]: 2
T[2]: 4
T[3]: 7
T[4]: 14
T[5]: 14
T[6]: 19
```

Remarque :

Dans le pire des cas le tri à bulles fait (n-1)+(n-2)+…+2+1 comparaisons et autant de permutations.

II.3 Le tri par insertion

II.3.1 Principe

Le tri par insertion consiste à chercher la position du $i^{ème}$ élément dans la partie du tableau commençant de 1 à i sachant que les i-1 premiers éléments sont triés. Si cette position est i, l'élément est donc à sa bonne place, sinon, supposons que cette position est j. Ce j est forcément entre 1 et i-1. On décale d'un pas vers l'avant (à droite) tous les éléments de j à i-1 puis on insère l'élément d'indice i à la position j.
On commence ce procédé à partir du $2^{ème}$ élément i=2.
Cette méthode ressemble étroitement au rangement d'une main de cartes par un joueur de belotte.

II.3.2 Résolution du problème : tri par insertion

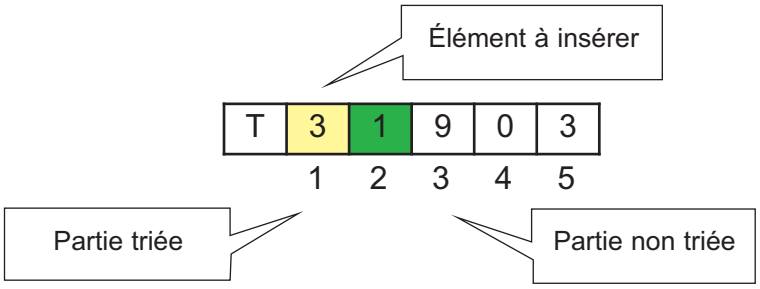
Soit à trier un tableau T de 5 entiers en ordre croissant en utilisant la méthode de tri par insertion.

Pré-analyse
On considère le tableau T contenant les 5 éléments suivants :

T	3	1	9	0	3
	1	2	3	4	5

Au départ, on considère que la partie triée contient le seul premier élément qui vaut 3, le reste formant la partie non triée. Nous commençons donc par le deuxième élément du tableau à savoir l'élément contenant 1 et d'indice 2.

1) On insère l'élément n°2 dans la première partie du tableau à sa place.



- On affecte à la variable auxiliaire TEMP la valeur de l'élément d'indice 2
- On décale d'une case à droite l'élément d'indice 1
- Affecter à la dernière case décalée la valeur de TEMP

T	1	3	9	0	3
	1	2	3	4	5

2) - On insère l'élément n°3 dans la première partie du tableau à sa place.
- Puisque 9>3, donc les trois premiers éléments sont déjà en ordre.

T	1	3	9	0	3
	1	2	3	4	5

3) - On insère l'élément n°4 dans la première partie du tableau à sa place.

T	1	3	9	0	3
	1	2	3	4	5

- On affecte à TEMP la valeur de l'élément d'indice 4.
- Pour insérer l'élément n° 4, nous allons parcourir la partie triée de la droite vers la gauche en décalant d'une case à droite l'élément n° 3, puis l'élément n° 2 et ainsi de suite jusqu'à avoir un élément ayant une valeur inférieure à celle de l'élément à insérer.
- Affecter à la dernière case décalée la valeur de TEMP.

T	0	1	3	9	3
	1	2	3	4	5

4) - On insère l'élément n°5 dans la première partie du tableau à sa bonne place.

T	0	1	3	9	3
	1	2	3	4	5

- On affecte à TEMP la valeur de l'élément d'indice 5.
- On décale d'une case à droite l'élément n°4, jusqu'à avoir un élément inférieur à 3.
- Affecter à la dernière case décalée la valeur de TEMP.

Résultat : tableau trié.

T	0	1	3	3	9
	1	2	3	4	5

Analyse

L'analyse ci-dessous comporte une procédure **Decaler_d** et une procédure **Inserer** permettant de mettre en œuvre la procédure du tri par insertion.

	DEF PROC Tri_insertion (n : Entier ; VAR T : TAB)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T_trié T_trié = [ ] **Pour** i de 2 **à** n **faire** v ← T[i] j ← i *PROC Decaler_d(T,j,v)* *PROC Inserer(T,j,v)* **Fin Pour** **Fin Tri_insertion**	i v j Decaler_d Inserer

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableau de déclaration des objets de la procédure Tri_insertion

Objet	Type/nature	Rôle
i	Entier	Compteur
v	Entier	élément à insérer
j	Entier	Position de l'élément à insérer
Decaler_d	Procédure	Décale les éléments d'un tableau vers la droite
Inserer	Procédure	Insère un élément dans un tableau dans une position bien définie

	DEF PROC Decaler_d (VAR T:TAB; VAR p:Entier ; e:Entier)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T **T=[ ]Tant que** (T[p-1] > e) **Répéter** T[p] := T[p-1] p:= p-1 **Fin Tantque** **Fin Decaler_d**	T p e

	DEF PROC Inserer (VAR T:TAB; p:Entier ; e:Entier)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T T[p] := e **Fin Inserer**	T p e

Algorithmes

*0) DEF PROC **Tri_insertion** (n :Entier ; VAR T : TAB)*
1) Pour i de 2 à n faire
v ← T[i]
j ← i
PROC Decaler_d(T,j,v)
PROC Inserer(T,j,v)
Fin Pour
2) Fin PROC Tri_insertion

*0) DEF PROC **Decaler_d** (var T:TAB; var p:Entier ; e:Entier)*
1) Tant que T[p-1] > e Répéter
T[p] := T[p-1]
p:= p-1
Fin tantque
2) Fin PROC Decaler_d

*0) DEF PROC **Inserer** (var T:TAB; p:Entier ; e:Entier)*
1) T[p] := e;
2) Fin PROC Inserer

II.3.3 Traduction en Turbo Pascal

```
PROCEDURE Tri_insertion(n:INTEGER ; var T: TAB);
    VAR  i,j,v: INTEGER;
    PROCEDURE Decaler_d(var T:TAB; var p:INTEGER;e:INTEGER);
      BEGIN
        WHILE T[p-1] > e DO
          BEGIN
            T[p] := T[p-1];
            p:= p-1
          END;
      END;
    PROCEDURE Inserer(var T:TAB; p:INTEGER; e:INTEGER);
      BEGIN
        T[p] := e;
      END;

    BEGIN
      FOR i := 2 TO n DO
       BEGIN
         v := T[i];
         j := i;
         Decaler_d(T,j,v);
         Inserer(T,j,v);
       END;
    END;
```

Remarque :
Dans le pire des cas le tri par insertion fait 1+2+…+n-1 comparaisons et autant de décalages.

En résumé, il fait n(n-1) opérations (comparaisons et décalages confondus) ; on verra plus loin que ce nombre sera considérablement réduit en utilisant la méthode de dichotomie pour la recherche de la position d'insertion.

Retenons

Un algorithme de tri consiste à ranger dans un ordre croissant ou décroissant une suite d'éléments quelconques selon un critère préalablement défini .
On se propose de trier un tableau T par ordre croissant.

La méthode du tri par sélection du tableau T consiste à :

1. Commencer par i=1 et chercher la position pmin du plus petit élément de T. Si T[i] ≠T[pmin] alors on les permute. T[1] est maintenant à sa bonne place.

2. On recommence ces opérations pour la partie du tableau de 2 à n. Après quoi T[2] sera à sa bonne place, et ainsi de suite jusqu'au traitement de la partie formée par les deux derniers éléments. Sachez que si les n-1 éléments du tableau sont bien placés le dernier le sera d'office.

La méthode du tri à bulles du tableau T consiste à :

1. Commencer par i=1 et comparer T[1] et T[2] s'ils ne sont pas dans le bon ordre on les permute puis on passe à la paire T[2] et T[3] puis la paire T[3] et T[4] jusqu'à la dernière paire T[n-1] et T[n]. On aura pousser ainsi le plus grand élément de T vers la dernière position du tableau, donc T[n] est maintenant à sa bonne place donc plus la peine d'y revenir.

2. On recommence les opérations de l'étape 1 en parcourant de 1 à n-2 puis de 1 à n-3 et ainsi de suite. On arrête quand la partie à trier est réduite à un seul élément ou si lors du dernier parcours aucune permutation n'a été faite ; ce qui signifie que le tableau est devenu trié.

La méthode du tri par insertion du tableau T consiste à :

Nous supposons que les éléments de 1 à i-1 sont bien placés (triés), nous allons placer le $i^{ème}$ élément.

1. On cherche sa position d'insertion de l'élément suivant, à savoir celui de la position i, à sa position parmi les i-1 premiers éléments. Si cette position est i, l'élément est donc à sa bonne place, sinon, supposons que cette position est j. On décale d'un pas vers l'avant tous les éléments de j à i-1 puis on insère l'élément d'indice i à la position j.

2. On recommence les opérations de l'étape 1 jusqu'à atteindre la fin du tableau.

Exercices

Exercice 1

On se propose d'écrire une analyse permettant :

1. de saisir les éléments d'un tableau T composé de n chaînes de caractères non vides.
2. de trier le tableau T dans un ordre croissant selon les deux critères suivants :
   - ✓ longueur de la chaîne en premier lieu
   - ✓ ordre alphabétique en cas d'égalité pour les longueurs.

Exercice 2

Il existe plusieurs variantes de l'algorithme du tri à bulles :

Une autre version est le tri bidirectionnel. Elle consiste à parcourir le tableau de gauche à droite, puis de droite à gauche, le changement de direction ayant lieu chaque fois que l'une des extrémités est atteinte. Ainsi, les plus petits éléments du tableau descendent au même rythme que remontent les plus grands éléments.

Question :

Écrire un programme Pascal permettant de saisir n entiers (10<n<30) dans un tableau T et de le trier en utilisant le principe mentionné ci-dessus.

Exercice 3

Nous disposons de deux tableaux Tnom de n chaînes de caractères et Tcouleur de n caractères. Une couleur peut être blanche de caractère 'B' ou noire de caractère 'N'.

Écrire un programme en Pascal qui permet de réarranger les éléments de Tnom et Tcouleur de manière à ce que les éléments de couleur 'B' précèdent les éléments de couleur 'N'. Si deux éléments ont des couleurs identiques , l'ordre alphabétique des chaînes intervient.

Exemple :

(Ali, B)(Salah, N), (Sonia, B), (Tounsi, N), (Salma, N) et (Ahmed, B) sont réarrangés comme suit :
(Ahmed, B), (Ali, B), (Sonia, B), (Salah, N), (Salma, N) et (Tounsi, N).

Exercice 4

Faire une analyse, écrire un algorithme puis la traduction en Pascal d'un programme qui permet de créer un tableau d'entiers croissants à partir de deux tableaux d'entiers non croissants.

Exemple :

V1 = 1, 3, 2, -6
V2 = 0, 4, -5
résultat = -6, -5, 0, 1, 2, 3, 4

Exercice 5

On veut écrire un programme pascal permettant de faire une étude comparative des algorithmes de tri (sélection, bulles et insertion).

Questions :

1. Le programme devra remplir aléatoirement un tableau de N entiers, l'afficher, le trier à l'aide des différentes méthodes de tri proposées (toujours à partir du même tableau initial) et afficher les tableaux triés obtenus.

2. Comparer les temps d'exécution de ces algorithmes pour différentes valeurs de N.

Exercice 6

Écrire, en s'inspirant du tri par sélection, une procédure qui permet de construire à partir d'un tableau T de n entiers un tableau RANG tel que RANG[i] soit l'indice dans T du ieme élément dans l'ordre croissant sans modifier le tableau T.

Exemple :

T	80	50	91	34	20
	1	2	3	4	5

RANG	5	4	2	1	3
	1	2	3	4	5

Leçon 2

Algorithmes de recherche d'un element dans un tableau

Objectifs spécifiques

- Connaître les différentes méthodes de recherches.
- Savoir choisir la méthode de recherche la plus adaptée au problème traité.

Plan de la leçon

Leçon 2

Algorithmes de recherche d'un element dans un tableau

la vraie méthode est la voie par laquelle la vérité elle-même, ou les essences objectives des choses ou leurs idées sont cherchées dans l'ordre dû.

SPINOZA

I. Introduction

On a souvent besoin de rechercher, dans un tableau, un élément donné ou son éventuelle position. Un point particulier à ne pas oublier pour tous les algorithmes est le traitement du cas où l'élément cherché n'est pas dans le tableau. Une autre caractéristique importante d'un algorithme de recherche est son comportement désiré en cas d'éléments identiques (doit-il donner le premier, le dernier ou tous les éléments identiques ?).
Dans cette leçon, nous nous limiterons à l'étude de deux méthodes de recherche parmi les plus connues, notamment la recherche séquentielle et la recherche dichotomique.

II. La recherche séquentielle

Activité 1

Écrire un programme qui saisit un entier naturel n suivi de n entiers à mettre dans un tableau A puis une valeur v, ensuite il appelle une fonction recherche qui cherche si v figure ou non dans le tableau A.

Pré-analyse

Nous disposons d'un tableau A à n éléments. Nous nous proposons de chercher si une valeur v existe dans le tableau A. Nous allons écrire une fonction booléenne qui renvoie VRAI si v existe dans A et FAUX sinon.
La méthode que nous allons adopter consiste à parcourir le tableau A à partir du premier élément et comparer chaque élément de A avec v. Le parcourt s'arrête si on trouve v ou on arrive à la fin du tableau.

Analyse

	Nom = Cherche_elem	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire ("Existe = ", exist)	exist
3	exist ← FN Recherche (A, n, v)	A
2	v = Donnée	n
1	A = [n = Donnée] **Pour** i de 1 à n **Faire**	v
	A[i] = Donnée	i
	FinPour	Recherche
5	**Fin cherche_elem**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
exist	Booléen	Paramètre effectif résultat booléen de la recherche
A	Tableau d'entiers	Tableau des éléments
n	Entier	Nombre d'éléments de A
v	Entier	Élément à rechercher
i	Entier	Compteur
Recherche	Fonction	Renvoie VRAI si v existe dans A et FAUX sinon.

Algorithme

1) Début Cherche_elem
2)Lire (n)
3)Pour i de 1 à n Répéter
Lire (A(i))
FinPour
4)Lire (v)
5)exist ← FN Recherche (A, n, v)
6)Ecrire ("Existe = ", exist)
7)Fin Chercher_elem.

Analyse de la fonction Recherche

	DEF FN Recherche (n, p : Entier ; TE : tableau de 20 entiers) : Booléen	
S	**L.D.E.**	**O.U.**
	Résultat = Rechercher	TE
2	Recherche = [ ]**Si** (TE[i] = p) **Alors**	n
	Recherche ← VRAI	p
	Sinon	i
	Recherche ← FAUX	
	FinSi	
1	i = [i← 0] **Répéter**	
	i ← i +1	
	Jusqu'à (TE[i] = p) **OU** (i = n)	
3	**Fin Recherche**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
i	Entier	Compteur

Algorithme

0) DEF FN Recherche (n, p : entier ; TE : tableau de 20 entiers) : booléen
1) [i ← 0] Répéter
i ← i +1
Jusqu'à (TE[i] = p) OU (i = n)
2) Recherche = Si TE[i] = p Alors
Recherche ← VRAI
Sinon
Recherche ← FAUX
FinSi
3) Fin Recherche

Traduction en Pascal

```
PROGRAM Cherche_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VARexist : BOOLEAN ;
   Indice : INTEGER;
   T : Typtab ;
   n,i,v : INTEGER;

FUNCTION Recherche (n,p:INTEGER ; TE:Typtab):BOOLEAN;
VAR
  i : INTEGER ;
BEGIN
  Exist := False ;
  i := 0 ;
  REPEAT
     i := i + 1 ;
  UNTIL  (TE[i] = p)OR( i = n );
  IF  TE[i] = p THEN
                  Recherche:= TRUE
               ELSE
                  Recherche:= FALSE ;
END ;
```

```
BEGIN
WRITE('Entrer la taille du tableau : ');
READLN(n);
  FOR i := 1 TO n DO
    BEGIN
     WRITE('Entrer un élément :  ');
     READLN(T[i]);
   END;
WRITE('Entrer l''élément à rechercher :  ');
READLN(v);
  Exist:=Recherche (n, v, T);
  WRITELN ('Exist =', exist);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 7
Entrer un élément :  3
Entrer un élément :  8
Entrer un élément :  9
Entrer un élément :  6
Entrer un élément :  7
Entrer un élément :  6
Entrer un élément :  9
Entrer l'élément à rechercher :  6
Exist =TRUE
```

Remarque :

Cet algorithme effectue au maximum n comparaisons.

II.1 Définition

La méthode de recherche séquentielle d'un élément dans un tableau consiste à parcourir le tableau élément par élément en les comparant avec l'élément à chercher jusqu'à trouver ce dernier ou achever le tableau.

II.2 Application

Développez un programme qui saisit un entier naturel n suivi de n réels à mettre dans un tableau A, puis une valeur v dont on veut chercher la dernière position dans le tableau si elle y figure. Dans le cas où v ne figure pas dans le tableau le programme affichera la position 0.

Pré-analyse

Nous disposons d'un tableau A à n éléments. Nous nous proposons de chercher la dernière position d'une valeur v si elle existe dans le tableau A. Nous allons écrire une fonction booléenne qui renvoie la dernière position si v existe dans A et la valeur 0 sinon.
La méthode que nous allons adopter consiste à parcourir le tableau A à partir du dernier élément et comparer chaque élément de A avec v. Le parcourt s'arrête après avoir trouvé v ou arrivé au début du tableau.

Analyse

	Nom = Cherche_der_pos_elem	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire ("Dernière Position = ", pos)	pos
3	pos ← FN Recherche_der_pos (A, n, v)	A
2	v = Donnée	n
1	(A,n) = [n = Donnée] **Pour** i de 1 à n **Répéter**	v
	T[i] = Donnée	i
	FinPour	Recherche_der_pos
5	**Fin cherche_der_pos_elem**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Pos	Entier	Dernière position de l'élément v à chercher
A	Tableau d'entiers	Tableau des éléments
n	Entier	Nombre d'éléments de A
v	Entier	Élément à rechercher
i	Entier	Compteur
Recherche_der_pos	fonction	Renvoie VRAI si v existe dans A et FAUX sinon

Algorithme

0) Début Cherche_der_pos_elem
1) Lire (n)
Pour i de 1 à n Répéter
Lire (A(i))
FinPour
2) Lire (v)
3) pos ← FN Recherche_der_pos (A, n, v)
4) Ecrire ("Dernière Position = ", pos)
5) Fin Chercher_der_pos_elem

Analyse de la fonction Recherche_der_pos

	DEF FN Recherche_der_pos (TE : tableau de 20 entiers, n,m : Entier) : Entier	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Recherche_der_pos	TE
3	Recherche_der_pos ← p	n
2	p = [p← 0] **Si** (TE[i] = m) **Alors**	m
	p ← i	i
	FinSi	p
1	[i ← n+1] **Répéter**	
	i ← i -1	
	Jusqu'à (TE[i] = m) OU (i = 1)	
5	**Fin Recherche_der_pos**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
p	Entier	Dernière position
i	Entier	Compteur

Algorithme

0) DEF FN Recherchder_der_pos (TE : tableau de 20 entiers, n, m : entier) : entier
1) [i ← n+1] Répéter
i ← i -1
Jusqu'à (TE[i] = m) OU (i = 1)

2) p = [p ← 0] Si TE[i] = m
Alors
p ← i
FinSi
3) Recherchder_der_pos←P
4) Fin Recherche-der_pos

Traduction en Pascal

```
PROGRAM Cherche_der_pos_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VAR
    T : Typtab ;
    pos,n,v,i: INTEGER;
```

```
PROGRAM Cherche_der_pos_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VAR
    T : Typtab ;
    pos,n,v,i: INTEGER;

FUNCTION Recherche_der_pos (TE:Typtab; n,m:INTEGER):INTEGER;
VAR
  p,i : INTEGER ;
BEGIN
  i := n+1;
  REPEAT
     i := i - 1;
  UNTIL(TE[i] = m)OR(i = 1);
  p:=0;
  IF  TE[i] = m THEN
                    p:= i;
  Recherche_der_pos:=P;
END ;

BEGIN
  WRITE('Entrer la taille du tableau : '); READLN(n);
  FOR i := 1 TO n DO
    BEGIN
      WRITE('Entrer un élément : ');
      READLN(T[i]);
    END;
WRITE('Entrer l''élément à rechercher : ');READLN(v);
  pos:=Recherche_der_pos(T, n, v);
  WRITELN ('Dernière Position =', pos);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 6
Entrer un élément : 4
Entrer un élément : 8
Entrer un élément : 9
Entrer un élément : 7
Entrer un élément : 3
Entrer un élément : 8
Entrer l'élément à rechercher : 8
Dernière Position =6
```

III. La recherche dichotomique

Activité 1

On vous propose de chercher dans un dictionnaire la signification du mot informatique. Décrivez en détail comment vous allez faire cette recherche.

La première action est d'ouvrir le dictionnaire "en son milieu" et de voir si on'est pas tombé par hasard sur la page comportant le mot informatique.

Dans le cas contraire, en comparant la lettre initiale du mot informatique avec celle affichée en haut de la page du dictionnaire on saura si on doit rechercher ce mot dans la partie gauche ou la partie droite.

On continue ce processus jusqu'à ouvrir la page contenant le mot informatique. Cette approche profite du fait qu'un dictionnaire est trié.

Activité 2

Effectuer une analyse, un algorithme et la traduction en Pascal de la fonction intitulée RECHERCHE qui utilise la méthode de recherche dichotomique pour vérifier l'existence d'un élément donné dans un tableau de n entiers triés par ordre croissant.

Pré-analyse
On regarde l'élément au milieu du tableau.
- s'il est égal à la valeur cherchée, l'élément cherché est existant.
- s'il est inférieur à la valeur recherchée, il ne reste à traiter que la moitié droite du tableau.
- s'il est supérieur à la valeur recherchée, il ne reste à traiter que la moitié gauche du tableau.

On continue ainsi la recherche en diminuant à chaque fois de moitié le nombre d'éléments du tableau restants à traiter.

DEF FN Recherche (n, m : Entier ; TE:Typtab): Bouléen

S	L.D.E.	O.U.
1	**Résultat** = Result_recherche Result_recherche = [a←1, b←n, Result_recherche←Faux] **Répéter** [p←(a+b) DIV 2] **Si** (m=TE[p]) **Alors** Result_recherche←Vrai **Sinon Si** (m < TE[p]) **Alors** b ←p-1 **Sinon** a←p+1 **FinSi** **Jusqu'à** (Result_recherche) OU (a>b)	TE n m a b p
2	**Fin Recherche**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
a	Entier	Borne droite
b	Entier	Borne gauche
p	Entier	compteur

Algorithme

```
0)DEF FN Recherche (n,m:Entier ; TE:Typtab): Bouléen
1)a←1
  b←n
  Result_recherche←Faux
  Répéter
        p←(a+b) DIV 2
        Si (m=TE[p]) Alors  Result_recherche←Vrai
        Sinon Si (m < TE[p]) Alors
                                  b ←p-1
        Sinon
            a←p+1
        FinSi
    Jusqu'à (Result_recherche) OU (a>b)
 2)Fin Recherche
```

Traduction en Pascal

```
PROGRAM Cherche_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER;
VAR  Ok: BOOLEAN;
     T : Typtab;
     n,i,p : INTEGER;

FUNCTION Recherche (n,m:INTEGER ; TE:Typtab): BOOLEAN;
VAR
   a,b :INTEGER;
Result_recherche:BOOLEAN ;
BEGIN
  a:=1;
  b:=n;
  Result_recherche:=False ;
```

```
  REPEAT
    p := (a+b) DIV 2 ;
    IF m = TE[p] THEN Result_recherche:=True
    ELSE IF m < TE[p] THEN
                        b :=p-1
    ELSE
      a:=p+1;
   UNTIL (Result_recherche) OR (a>b);
   Recherche:=Result_recherche;
END;

BEGIN
WRITE('Entrer la taille du tableau : ');
READLN (n);
FOR   i := 1 TO n DO
   BEGIN
       WRITE('Entrer un élément : ');
       READLN (T[i]);
   END;
WRITE('Entrer l''élément à rechercher : ');
READLN (p);
OK:= Recherche (n,p,T);
WRITELN ('Exist = ' ,OK);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 7
Entrer un élément : 12
Entrer un élément : 15
Entrer un élément : 15
Entrer un élément : 19
Entrer un élément : 25
Entrer un élément : 27
Entrer un élément : 99
Entrer l'élément à rechercher : 25
Exist = TRUE
```

Remarques :

- Nous pouvons constater qu'à la première étape, nous devons prendre le milieu d'un tableau de n éléments, à l'étape suivante le milieu d'un tableau de n/2 éléments, puis de n/4 éléments, ...
- La recherche s'arrête quand le nombre d'éléments est réduit à 1 ou l'élément est trouvé.

Définition

En algorithmique, la dichotomie (du grec « couper en deux ») est un processus itératif ou récursif de recherche où à chaque étape l'espace de recherche est restreint à l'une de deux parties.
On suppose bien sûr qu'il existe un test relativement simple permettant à chaque étape de déterminer l'une des deux parties dans laquelle peut se trouver l'élément.

L'algorithme s'applique typiquement à la recherche d'un élément dans un ensemble fini ordonné.

La dichotomie peut être vue comme une variante simplifiée de la stratégie plus générale diviser pour régner (en anglais, divide and conquer).

Retenons

– La méthode de la recherche séquentielle consiste à parcourir le tableau progressivement du début vers la fin en les comparant avec l'élément à chercher jusqu'à trouver ce dernier ou achever le tableau.

L'algorithme de recherche séquentielle n'utilise comme information que le test d'égalité sur les éléments, avec deux résultats possibles : égalité ou non égalité.

– La méthode de la recherche dichotomique consiste à chercher un élément dans le cas d'un tableau trié.

On compare l'élément à chercher à l'élément central du tableau, s'ils sont différents, un test permet de trouver dans quelle moitié du tableau on peut trouver l'élément. On continue ce processus jusqu'à trouver l'élément ou bien on arrive à un sous-tableau de taille 1.

Exercices

Exercice 1

Effectuer une analyse, un algorithme et la traduction en Pascal du programme intitulé ANNUAIRE1 qui, saisit un nom puis détermine le numéro de téléphone correspondant.

Avec deux tableaux Tnom et Ttel, indicés en parallèle, le numéro de téléphone de Tnom[i] étant Ttel[i], on peut faire un annuaire téléphonique. L'information à rechercher est le numéro de téléphone.

Exemple :

Tnom	Index	Ttel
Jamel	1	71222
Naoufel	2	75123
Imed	3	73632
Ghazi	4	72098
Salma	5	78453
Ibtissem	6	72331
Hakim	7	72776
Afef	8	77345
Abdelkader	9	78080
Fadhila	10	77777
Zoubeir	11	72987

Si le nom est Salma alors le numéro de téléphone correspondant est **78453**.

Exercice 2

Supposons que la table des noms est triée par ordre alphabétique (comme l'annuaire des PTT). Au lieu de faire une recherche séquentiellement, utiliser la méthode de recherche dichotomique pour résoudre l'exercice 1.

Exercice 3

Écrire un programme qui lit n caractères introduits au clavier par l'utilisateur et les place au fur et à mesure dans un tableau T. Ensuite le programme recherche dans le tableau la plus longue suite de caractères identiques et affiche le caractère concerné ainsi que le nombre de fois qu'il est répété. On suppose que les caractères tapés par l'utilisateur sont différents des blancs.

Exemple :

si on introduit (a a b c c e d e e e e e f f g a a a),
le programme doit afficher (e,5).

Exercice 4

Écrire un programme en Pascal qui réalise le traitement suivant :
Compactage d'un tableau d'entiers positifs ou nuls : remplacer chaque suite de zéros par le nombre de zéros multiplié par -1.

Exemple :
Vecteur = 1 2 3 0 0 0 2 0 0
résultats = 1 2 3 -3 2 -2

Exercice 5

On veut insérer un nouveau élément dans un tableau trié de n entiers en maintenant le tableau ordonné. Pour insérer un nouvel élément dans le tableau, il faut d'abord trouver son emplacement par une recherche dichotomique, puis décaler tous les éléments pour pouvoir insérer le nouveau élément au bon endroit.
Écrire un programme en Pascal, qui permet d'insérer un élément dans un tableau trié.

Exercice 6

Écrire un programme Pascal permettant :
- d'insérer un réel x dans un tableau à la position p. Si le tableau est plein le programme doit d'abord construire un nouveau tableau dont le nombre d'éléments est 10% supérieur au tableau initial.
- de supprimer l'élément à la position p et de décaler le reste des éléments T[p+1..n] vers la gauche.