

# LES ALGORITHMES DE TRI

## TRI PAR SÉLECTION :

### Principe :

- ☑ Comparer tous les éléments de la liste non triée afin de sélectionner le plus petit.
- ☑ Permuter le plus petit élément avec le premier élément de la liste non triée.
- ☑ Répéter les étapes 1 et 2 un nombre de fois égal à  $n-1$ , en considérant toujours la liste non triée uniquement.

## TRI À BULLES :

### Principe :

- ☑ On commence par la première paire d'éléments que l'on compare.
- ☑ Si  $t[i] > t[i+1]$ , alors on permute ces deux éléments et on tient compte de la permutation. (permuter  $\leftarrow$  vrai)
- ☑ On considère la paire suivante et on répète 1 et 2, jusqu'à comparer la dernière paire
- ☑ Si une (ou plusieurs) permutation a été réalisée, on recommence tout le traitement (étape 1, 2 et sinon, c'est la fin et la liste sera triée.

## TRI À insertion :

- ☑ Commencer par le deuxième élément,
- ☑ Comparer l'élément choisi avec tous les éléments précédents dans la liste et l'insérer à la bonne place de sorte que la liste formée par les éléments traités reste toujours triée.
- ☑ Répéter les étapes 1 et 2 jusqu'à traiter le dernier élément de la liste.

## LES algorithmes de tri Et de recherche

### Objectifs :

- Manipulation des algorithmes de tri et de recherche, à savoir :
  - **Tri** : par sélection, à bulles et par insertion.
  - **Recherche** : séquentielle et dichotomique

### A. Le tri d'un tableau :

#### I. Introduction :

Le tri est une opération qui consiste à répartir ou organiser une collection d'objets selon un ordre déterminé. Dans le domaine de l'informatique, il existe plusieurs méthodes de tri (algorithmes). Dans ce chapitre nous allons découvrir trois méthodes de tri :

- Tri par sélection,
- Tri à bulles,
- Tri par insertion.

#### II. Tri par sélection :

**Activité :** Ecrire un programme qui permet de saisir un tableau **T** de **n** entiers, puis trier en ordre croissant ce tableau en utilisant la **méthode de tri par sélection** et afficher le résultat.

**a) Principe :** Cette méthode de tri consiste à :

1. Se pointer à la **1<sup>ère</sup>** case du tableau **T** et de parcourir la totalité du tableau pour **repérer l'indice de la première position du minimum**.
2. **Comparer** ce minimum avec **T [1]**. S'ils sont différents on **les permute**.
3. Le **sous tableau** de **T** allant de **2 à n** est à priori non trié, on applique l'étape 1 et 2 et ainsi de suite jusqu'à l'avant dernier élément (**n-1**).


#### b) Exemple :

Soit un tableau **T** contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

**Etape 1 :** Parcourir la totalité du tableau pour repérer le minimum (indice de la première position du minimum) et le comparer avec **T [1]**

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10


  
Indice du minimum

**T [1] <> T [4] alors permutation**

On obtient :

T :	-5	10	0	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Le sous tableau allant de 2 à n est à priori non trié, on applique l'étape 1 et 2 et ainsi de suite jusqu'à l'avant dernier élément (**n-1**).

Etape2 :

T :	-5	10	0	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

$T[2] \diamond T[7]$  alors permutation

On obtient :

T :	-5	-2	0	12	8	12	10	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Etape3 :

T :	-5	-2	0	12	8	12	10	2	40	-1
	1	2	3	4	5	6	7	8	9	10

$T[3] \diamond T[10]$  alors permutation

On obtient :

T :	-5	-2	-1	12	8	12	10	2	40	0
	1	2	3	4	5	6	7	8	9	10

Etape4 :

T :	-5	-2	-1	12	8	12	10	2	40	0
	1	2	3	4	5	6	7	8	9	10

$T[4] \diamond T[10]$  alors permutation

On obtient :

T :	-5	-2	-1	0	8	12	10	2	40	12
	1	2	3	4	5	6	7	8	9	10

Etape5 :

T :	-5	-2	-1	0	8	12	10	2	40	12
	1	2	3	4	5	6	7	8	9	10


$T[5] \diamond T[8]$  alors permutation

On obtient :

T :	-5	-2	-1	0	2	12	10	8	40	12
	1	2	3	4	5	6	7	8	9	10

Etape6 :

T :	-5	-2	-1	0	2	12	10	8	40	12
	1	2	3	4	5	6	7	8	9	10

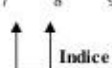

  
 Indice du minimum

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Etape7 :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

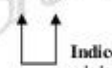

  
 Indice du minimum

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Etape8 :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

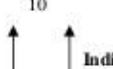

  
 Indice du minimum

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Etape9 :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10


  
 Indice du minimum

On obtient :

T :	-5	-2	-1	0	2	8	10	12	12	40
	1	2	3	4	5	6	7	8	9	10

**Remarque :**

- On est arrivé à l'élément numéro n-1, alors arrêt du traitement.
- Nous n'avons pas besoin de traiter le dernier élément, puisque si les neuf premiers éléments sont triés alors automatiquement le dernier sera le plus grand et par conséquent il se trouve à la bonne position.

**c) Spécifications et algorithmes du problème :****1. Spécification du programme principale :**

- **Résultat** : Afficher le tableau T trié en utilisant une procédure **Affiche**
- **Traitements** : Il faut trier le tableau T, en utilisant une procédure **Tri**
- **Données** : Il faut remplir le tableau T et saisir la taille n, en utilisant la procédure **Saisie**

**2. Algorithme du programme principal :**0) **Début** Tri\_Selection1) **Saisie** (T, N)2) **Tri** (T, N)3) **Affiche** (T, N)4) **Fin** Tri\_Selection**Tableau de déclaration des nouveaux types**

Types
TAB = Tableau de 100 entiers

**Tableau de déclaration des Objets**

Objets	Type/Nature
T	TAB
N	Entier
Saisie, Affiche, Tri	Procédure

**3. Spécification de la procédure Saisie :**

- **Résultat** : Saisir **Nf** et remplir **Tf**
- **Traitements** : Le remplissage d'un tableau est une action répétitive, ou on connaît le nombre de répétition qui est égale à **Nf**, d'où utilisation de la boucle POUR ... FAIRE ... La saisie de l'entier **Nf** doit être contrôlée pour ne pas saisir un entier négatif ou supérieur à 100. Cette procédure admet deux paramètres formels qui sont **Nf** et **Tf**.

**4. Algorithme de la procédure Saisie :**0) **Début** procédure Saisie (VAR Tf : TAB ; VAR Nf : Entier)1) **Répéter**

Ecrire (" Donner la taille du tableau : "), Lire (Nf)

Jusqu'à (Nf dans [1..100])

2) **Pour** i de 1 à Nf **Faire**

Ecrire (" Donner l'élément N°", i, " : "), Lire (Tf[i])

**Fin Pour**3) **Fin** Saisie**Tableau de déclaration des Objets locaux**

Objets	Type/Nature
i	Entier

**5. Spécification de la procédure Tri :**

- **Résultat** : Trier le tableau **Tf**
- **Traitements** : Il s'agit d'un traitement répétitif jusqu'à l'avant dernier élément du tableau, d'où utilisation de la boucle POUR ... FAIRE ..., pour chaque élément nous allons exécuter deux actions :



- **Action1** : Chercher la première apparition du minimum
- **Action2** : Comparer l'indice du minimum et celui de l'élément en cours, s'ils sont différents, alors on applique la permutation à ces deux éléments.

Donc on fera appel à une fonction intitulée **Premposmin** qui retourne le premier indice du minimum et à une procédure intitulée **Permut** permettant de permuter deux éléments.

Les paramètres formels pour cette procédure sont le tableau **Tf** et sa taille **Nf**.

#### 6. Algorithme de la procédure Tri :

0) **Début procédure Tri** (VAR Tf : TAB ; Nf : Entier)

1) **Pour i de 1 à Nf-1 Faire**

Pmin ← **Premposmin** (Tf, Nf, i)

**Si** i ≠ Pmin **Alors**

**Permut** (Tf[i], Tf[Pmin])

**Finsi**

**Fin Pour**

2) **Fin Tri**

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
i, Pmin	Entier
Premposmin	Fonction
Permut	Procédure

#### 7. Spécification de la fonction Premposmin :

- **Résultat** : Déterminer la première position du minimum dans un sous tableau
- **Traitements** : Il s'agit d'un traitement répétitif jusqu'à le dernier élément du tableau, d'où utilisation de la boucle POUR ... FAIRE. On doit initialiser la position du minimum à i puis faire le parcours du sous tableau, dès que on trouve un élément inférieur à ce minimum on change la position par l'indice de cet élément.

Les paramètres formels de cette fonction sont **Tf**, **Nf** et **pd**

#### 8. Algorithme de la fonction Premposmin :

0) **Début fonction Premposmin** (Tf : TAB ; Nf, pd : Entier) : Entier

1) [pm ← pd] **Pour J de pd+1 à Nf Faire**

**Si** Tf[j] < Tf [pm] **Alors**

Pm ← j

**Finsi**

**Fin Pour**

2) **Premposmin** ← pm

3) **Fin Premposmin**

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
i, pm	Entier

#### 9. Spécification de la procédure Permut :

- **Résultat** : Permuter deux variables entiers X et Y
- **Traitements** : pour permuter deux variables, on procède généralement à utiliser une variable intermédiaire et faire la permutation.

Les paramètres formels de cette procédure sont **X** et **Y**.

#### 10. Algorithme de la procédure Permut :

0) **Début procédure Permut** (VAR X, Y : Entier)

1) Int ← X

2) X ← Y

3) Y ← Int

4) **Fin Permut**

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
Int	Entier

#### 11. Spécification de la procédure Affiche :

- **Résultat** : Afficher le tableau **Tf**
- **Traitements** : Il s'agit d'un traitement répétitif pour afficher chaque élément du tableau **Tf**, donc l'instruction d'affichage va être exécuter **Nf** fois, le nombre de répétition est connu d'avance, d'où utilisation de la boucle POUR ... FAIRE...

Les paramètres formels de cette procédure sont **Tf** et **Nf**.

#### 12. Algorithme de la procédure Affiche :

0) **Début procédure Affiche** (Tf : TAB ; Nf : Entier)

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
i	Entier

- |  |            |    |    |    |    |    |    |    |    |    |    |              |
|--|------------|----|----|----|----|----|----|----|----|----|----|--------------|
| $T[1] > T[2]$ alors permutation        | T :        | 12 | 10 | 0  | -5 | 8  | 12 | -2 | 2  | 40 | -1 | Test<br>Vrai |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[2] > T[3]$ alors permutation        | T :        | 10 | 12 | 0  | -5 | 8  | 12 | -2 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[3] > T[4]$ alors permutation        | $\Gamma$ : | 10 | 0  | 12 | -5 | 8  | 12 | -2 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[4] > T[5]$ alors permutation        | $\Gamma$ : | 10 | 0  | -5 | 12 | 8  | 12 | -2 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[5] = T[6]$ alors pas de permutation | $\Gamma$ : | 10 | 0  | -5 | 8  | 12 | 12 | -2 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[6] > T[7]$ alors permutation        | T :        | 10 | 0  | -5 | 8  | 12 | 12 | -2 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[7] > T[8]$ alors permutation        | T :        | 10 | 0  | -5 | 8  | 12 | -2 | 12 | 2  | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |
| $T[8] < T[9]$ alors pas de permutation | $\Gamma$ : | 10 | 0  | -5 | 8  | 12 | -2 | 2  | 12 | 40 | -1 | Test<br>Faux |
|  |            | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |              |



T[9] > T[10] alors permutation	T :	10	0	-5	8	12	-2	2	12	40	-1	Test
		1	2	3	4	5	6	7	8	9	10	Faux
	T :	10	0	-5	8	12	-2	2	12	-1	40	Test
		1	2	3	4	5	6	7	8	9	10	Faux

Puisqu'on a atteint la fin du tableau et le contenu de la variable **test** est **Faux**, alors on doit recommencer une nouveau passage et ainsi de suite jusqu'à ce qu'on fasse un passage complet du tableau sans modifier le contenu de **test (Vrai)**.

### c) Spécifications et algorithmes du problème :

#### 1. Spécification du programme principale :

- **Résultat** : Afficher le tableau T trié en utilisant une procédure **Affiche**
- **Traitements** : Il faut trié le tableau T, en utilisant une procédure **Tri**
- **Données** : Il faut remplir le tableau T et saisir la taille N, en utilisant la procédure **Saisie**

#### 2. Algorithme du programme principal :

- 0) Début Tri\_Bulle
- 1) Saisie (T, N)
- 2) Tri (T, N)
- 3) Affiche (T, N)
- 4) Fin Tri\_Bulle

Tableau de déclaration des nouveaux types

Types
TAB = Tableau de 100 entiers

Tableau de déclaration des Objets

Objets	Type/Nature
T	TAB
N	Entier
Saisie, Affiche, Tri	Procédure

#### 3. Spécification de la procédure Tri :

- **Résultat** : Trier le tableau **Tf**
    - **Traitements** : Il s'agit d'un traitement répétitif jusqu'à ce que la valeur de la variable **test** reste à vrai, d'où on utilise la structure Répéter ... Jusqu'à
- Le passage du tableau est un parcours du premier élément jusqu'au dernier élément, d'où on utilise la boucle POUR ... FAIRE, et comparer chaque deux éléments consécutifs, s'ils ne sont pas dans le bon ordre on fait la permutation. Donc on fera appel à une procédure intitulée **Permut** permettant de permuter deux éléments.

Les paramètres formels pour cette procédure sont le tableau **Tf** et sa taille **Nf**.

#### 4. Algorithme de la procédure Tri :

- 0) Début procédure Tri (VAR Tf : TAB ; Nf : Entier)
- 1) Répéter
  - Test ← Vrai
  - Pour i de 1 à Nf-1 Faire
    - Si Tf[i] > Tf[i+1] Alors
    - Permut (Tf[i], Tf[i+1])
  - Finsi
- Fin Pour
- Jusqu'à (Test = Vrai)

Tableau de déclaration des Objets locaux

Objets	Type/Nature
i	Entier
Test	Booléen
Permut	Procédure

## 2) Fin Tri

## IV. Tri par insertion :

**Activité :** Ecrire un programme qui permet de saisir un tableau **T** de **n** entiers, puis trier en ordre croissant ce tableau en utilisant la **méthode de tri par insertion** et afficher le résultat.

**a) Principe :** Cette méthode de tri consiste à :

1. Considérer que les **i-1** premiers éléments du tableau **T** sont triés et insérer l'élément **N°i** dans sa position parmi les **i-1** déjà triés.
2. Répéter cette action jusqu'à le dernier élément du tableau **T**.

**NB :** L'insertion se traduit par le sauvegarde de l'élément **N° i** dans une variable intermédiaire (**Int**), puis le décalage d'un cran à droite des éléments **i-1, i-2, ...** jusqu'à avoir un élément inférieur à **Int** et finalement affecter le contenu de **Int** dans l'élément libre.

**b) Exemple :**

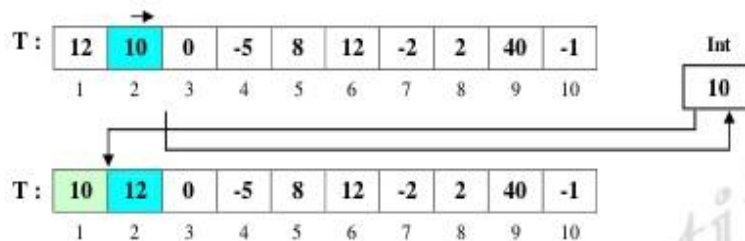
Soit un tableau **T** contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1	Int
	1	2	3	4	5	6	7	8	9	10	

On commence par l'élément **N°2** puisque si le tableau contient un seul élément, il est considéré trié.

**Etape1 :**

Recherche de la position d'insertion de **T[2]**



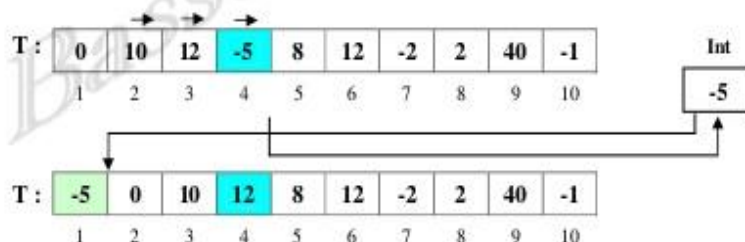
**Etape2 :**

Recherche de la position d'insertion de **T[3]**



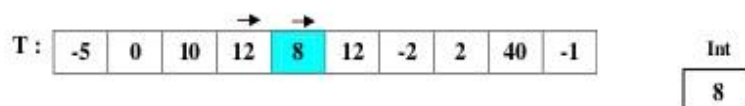
**Etape3 :**

Recherche de la position d'insertion de **T[4]**



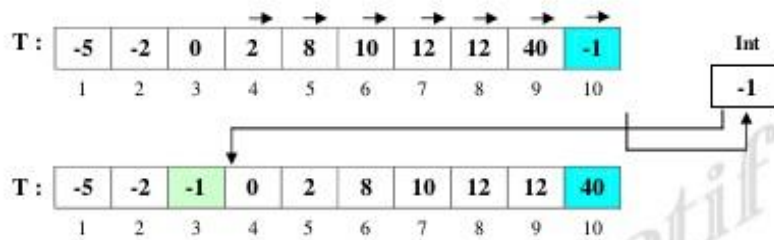
**Etape4 :**

Recherche de la position d'insertion de **T[5]**





**Etape9 :**  
**Recherche de la**  
**position d'insertion**  
**de T[10]**  
 On obtient :



### c) Spécifications et algorithmes du problème :

#### 1. Spécification du programme principale :

- **Résultat** : Afficher le tableau T trié en utilisant une procédure **Affiche**
- **Traitements** : Il faut trier le tableau T, en utilisant une procédure **Tri**
- **Données** : Il faut remplir le tableau T et saisir la taille N, en utilisant la procédure **Saisie**

#### 2. Algorithme du programme principal :

- 0) **Début** Tri\_Insertion
- 1) **Saisie** (T, N)
- 2) **Tri** (T, N)
- 3) **Affiche** (T, N)
- 4) **Fin** Tri\_Insertion

#### Tableau de déclaration des nouveaux types

Types
TAB = Tableau de 100 entiers

#### Tableau de déclaration des Objets

Objets	Type/Nature
T	TAB
N	Entier
Saisie, Affiche, Tri	Procédure

#### 3. Spécification de la procédure Tri :

- **Résultat** : Trier le tableau **Tf**
    - **Traitements** : Il s'agit d'un traitement répétitif pour tous les éléments du tableau, sauf le premier, d'où utilisation de la boucle **POUR ... FAIRE ...**,
  - **L** pour chaque élément nous allons exécuter ces actions :
    - **Action1** : Chercher la position correcte de l'élément en question
    - **Action2** : Le ranger dans une variable intermédiaire (**Int**)
    - **Action3** : Décaler d'une position les éléments supérieurs à cet élément
    - **Action4** : Affecter au dernier élément décalé la valeur de **Int**
- Les paramètres formels pour cette procédure sont le tableau **Tf** et sa taille **Nf**.

#### 4. Algorithme de la procédure Tri :

- 0) **Début** procédure Tri (VAR Tf : TAB ; Nf : Entier)
- 1) **Pour** i de 2 à Nf **Faire**
  - Pos ← 1
  - Tant que** (Pos < i) **Faire**

#### Tableau de déclaration des Objets locaux

Objets	Type/Nature
i, Pos, Int	Entier



```

    Si Tf[Pos] > Tf[i] Alors
        Int ← Tf[i]
        Pour j de i à Pos +1 Faire
            Tf[j] ← Tf[j-1]
        Fin Pour
        Tf[pos] ← Int
    Finsi
    Pos ← Pos +1
Fin Tant que
Fin Pour
2) Fin Tri

```

## B. La recherche d'un élément dans un tableau :

### I. Introduction :

La recherche d'un élément dans un tableau ou dans une liste de valeur est un traitement très utile en informatique. Parmi les méthodes de recherches, on cite :

- La recherche séquentielle,
- La recherche dichotomique.

### II. La recherche séquentielle :

**Activité :** Ecrire un programme qui permet de saisir un tableau **T** de **n** entiers, puis vérifier si un entier donné **X** existe dans le tableau ou non en utilisant la méthode de recherche séquentielle.

#### a) Principe :

Cette méthode de recherche consiste à parcourir les éléments du tableau un par un jusqu'à trouver la valeur cherchée ou arriver à la fin du tableau.

#### b) Exemple :

Soit un tableau T contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Pour **X = -2** le programme affichera "**-2 existe dans le tableau**"

Pour **X = 5** le programme affichera "**5 n'existe pas dans le tableau**"

### c) Spécifications et algorithmes du problème :

#### 1. Spécification du programme principale :

- **Résultat :** Afficher le résultat de l'existence de X dans le tableau T en utilisant une procédure **Affiche**
- **Traitements :** Il faut vérifier si X existe dans le tableau T ou non, en utilisant une procédure **Recherche**
- **Données :** Il faut remplir le tableau T, saisir la taille N et la valeur a cherchée X, en utilisant la procédure **Saisie**

#### 2. Algorithme du programme principal :

- 0) Début Recherche\_Seq
- 1) Saisie (T, N, X)
- 2) Verif ← Recherche (T, N, X)
- 3) Affiche (Verif)
- 4) Fin Recherche\_Seq

Tableau de déclaration des Objets

Objets	Type/Nature
T	TAB
N, X	Entier
Saisie, Affiche	Procédure
Recherche	Fonction



## Tableau de déclaration des nouveaux types

Types
TAB = Tableau de 100 entiers

## 3. Spécification de la procédure Saisie :

- **Résultat** : Saisir **Nf**, **Xf** et remplir **Tf**
- **Traitements** : Le remplissage d'un tableau est une action répétitive, ou on connaît le nombre de répétition qui est égale à **Nf**, d'où utilisation de la boucle POUR ... FAIRE ... La saisie de l'entier **Nf** doit être contrôlée pour ne pas saisir un entier négatif ou supérieur à 100. Cette procédure admet trois paramètres formels qui sont **Nf**, **Xf** et **Tf**.

## 4. Algorithme de la procédure Saisie :

0) **Début procédure** Saisie (**VAR Tf** : TAB ; **VAR Nf**, **Xf** : Entier)1) **Répéter**

Ecrire ("Donner la taille du tableau : "), Lire (Nf)

**Jusqu'à** (Nf dans [1..100])2) **Pour** i de 1 à Nf **Faire**

Ecrire ("Donner l'élément N°", i, " : "), Lire (Tf[i])

**Fin Pour**

3) Ecrire ("Donner la valeur à cherchée : "), Lire (Xf)

4) **Fin Saisie**

## Tableau de déclaration des Objets locaux

Objets	Type/Nature
i	Entier

## 5. Spécification de la fonction Recherche :

- **Résultat** : Retourner un résultat booléen déterminant l'existence.
- **Traitements** : Il s'agit de comparer **Xf** avec chaque élément du tableau **Tf** jusqu'à trouver la valeur ou atteindre la fin du tableau, donc utilisation d'une structure itérative à condition d'arrêt et au minimum on doit faire une comparaison si on trouve la valeur à la première case du tableau, d'où utilisation de la boucle REPETR ... JUSQU'A

Les paramètres formels de cette fonction sont **Xf**, **Nf** et **Tf**.

## 6. Algorithme de la fonction Recherche :

0) **Début fonction** recherche (**Tf** : TAB ; **Nf**, **Xf** : Entier) : Booléen1) [Trouve ← Faux, i ← 1] **Répéter****Si** Tf[i] = Xf **Alors**

Trouve ← Vrai

**Sinon**

i ← i+1

**Finsi****Jusqu'à** (Trouve = Vrai) **OU** (i > Nf)2) **Recherche** ← Trouve3) **Fin Recherche**

## Tableau de déclaration des Objets locaux

Objets	Type/Nature
i	Entier
Trouve	Booléen

## 7. Spécification de la procédure Affiche :

- **Résultat** : Afficher un commentaire pour dire si la valeur existe dans le tableau ou non
- **Traitements** : Il s'agit d'une structure conditionnelle suivant la valeur de **Veriff**, on affiche un commentaire

Les paramètres formels de cette procédure sont **Veriff**

## 8. Algorithme de la procédure Affiche :

0) **Début procédure** Affiche (**Veriff** : Booléen)1) **Si** Veriff = vrai **Alors**

Ecrire (X, " existe dans le tableau")  
**Sinon**  
 Ecrire (X, " n'existe pas dans le tableau")  
**Finsi**  
**2) Fin Affiche**

### III. La recherche dichotomique :

**Activité :** Ecrire un programme qui permet de saisir un tableau **T** de **n** entiers triés dans l'ordre croissant, puis vérifier si un entier donné **X** existe dans le tableau ou non en utilisant la méthode de recherche dichotomique.

#### a) Principe :

Cette méthode de recherche consiste à :

1. Fixer le début (**Deb**) et la fin (**Fin**) du tableau,
2. Fixer le milieu du tableau ( $Mil = (Fin + Deb) \text{ Div } 2$ ),
3. Comparer **X** et **T[Mil]**, Si ( $X > T[Mil]$ ) alors rechercher **X** dans le sous tableau [**Mil + 1 ... Fin**] sinon si ( $X < T[Mil]$ ) alors dans le tableau [**Deb ... Mil - 1**],
4. Répéter les étapes 1, 2 et 3 jusqu'à ( $X = T[Mil]$ ) ou ( $Deb > Fin$ )

#### b) Exemple :

Soit un tableau **T** contenant les dix éléments suivants :

<b>T :</b>	-5	-2	-1	0	2	8	10	12	12	40
	1	2	3	4	5	6	7	8	9	10

Pour **X = -2** le programme affichera "**-2 existe dans le tableau**"

Pour **X = 5** le programme affichera "**5 n'existe pas dans le tableau**"

#### c) Spécifications et algorithmes du problème :

##### 1. Spécification du programme principale :

- **Résultat :** Afficher le résultat de l'existence de **X** dans le tableau **T** en utilisant une procédure **Affiche**
- **Traitements :** Il faut vérifier si **X** existe dans le tableau **T** ou non, en utilisant une procédure **Recherche**
- **Données :** Il faut remplir le tableau **T**, saisir la taille **N** et la valeur **a** cherchée **X**, en utilisant la procédure **Saisie**

##### 2. Algorithme du programme principal :

- 0) **Début** Recherche\_Dicho
- 1) **Saisie** (T, N, X)
- 2) **Verif** ← **Recherche** (T, N, X)
- 3) **Affiche** (Verif)
- 4) **Fin** Recherche\_Dicho

Tableau de déclaration des nouveaux types

Types
<b>TAB</b> = Tableau de 100 entiers

Tableau de déclaration des Objets

Objets	Type/Nature
T	TAB
N, X	Entier
Saisie, Affiche	Procédure
Recherche	Fonction

##### 3. .

- **Résultat :** Saisir **Nf, Xf** et remplir **Tf**
- **Traitements :** Le tableau est formé par des entiers triés dans l'ordre croissant, donc on doit saisir l'élément N° 1 puis à chaque saisie on doit vérifier que l'élément est supérieur à celui qui le précède et ainsi de suite jusqu'à le dernier

élément. C'est un traitement répétitif, la structure adéquate est la boucle POUR ... FAIRE ...

... La saisie de l'entier **Nf** doit être contrôlée pour ne pas saisir un entier négatif ou supérieur à 100. Cette procédure admet trois paramètres formels qui sont **Nf**, **Xf** et **Tf**.

#### 4. Algorithme de la procédure Saisie :

0) **Début procédure** Saisie (**VAR Tf** : TAB ; **VAR Nf**, **Xf** : Entier)

1) **Répéter**

Ecrire ("Donner la taille du tableau : "), Lire (Nf)

**Jusqu'à** (Nf dans [1..100])

2) [Ecrire ("Donner l'élément N°1, " : "), Lire (Tf[1])]

**Pour i de 2 à Nf Faire**

**Répéter**

Ecrire ("Donner l'élément N°", i, " : "), Lire (Tf[i])

**Jusqu'à** (Tf[i] >= Tf[i - 1])

**Fin Pour**

3) Ecrire ("Donner la valeur a cherchée : "), Lire (Xf)

4) **Fin Saisie**

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
i	Entier

#### 5. Spécification de la fonction Recherche :

- **Résultat** : Retourner un résultat booléen déterminant l'existence.
- **Traitements** : Il s'agit de calculer l'indice de l'élément du milieu du tableau et le comparer avec **Xf**, s'ils sont égaux alors fin du traitement si non calculer de nouveau le début et la fin du sous tableau ou se trouve **Xf** et refaire le même traitement jusqu'à trouver **Xf** ou arriver à un état où début est supérieur à fin, donc utilisation d'une structure itérative à condition d'arrêt et au minimum on doit faire une comparaison, d'où utilisation de la boucle REPETR ... JUSQU'A

Les paramètres formels de cette fonction sont **Xf**, **Nf** et **Tf**.

#### 6. Algorithme de la fonction Recherche :

0) **Début fonction** recherché (Tf : TAB ; Nf, Xf : Entier) : Booléen

1) [Trouve ← Faux, Deb ← 1, Fin ← Nf] **Répéter**

Mil ← (Deb + Fin) Div 2

**Si** Tf[Mil] > Xf **Alors**

Fin ← Mil - 1

**Sinon**

**Si** Tf[Mil] < Xf **Alors**

Deb ← Mil + 1

**Sinon**

Trouve ← Vrai

**Finsi**

**Jusqu'à** (Trouve = Vrai) **OU** (Deb > Fin)

2) **Recherche** ← Trouve

3) **Fin Recherche**

**Tableau de déclaration  
des Objets locaux**

Objets	Type/Nature
i, Deb, Fin	Entier
Trouve	Booléen

#### 7. Spécification de la procédure Affiche :

- **Résultat** : Afficher un commentaire pour dire si la valeur existe dans le tableau ou non
- **Traitements** : Il s'agit d'une structure conditionnelle suivant la valeur de **Veriff**, on affiche un commentaire

Les paramètres formels de cette procédure sont **Veriff**

8. Algorithme de la procédure Affiche :
- 0) **Début** procédure Affiche (Veriff : Booléen)
  - 1) **Si** Veriff = vrai **Alors**
    - Ecrire (X, " existe dans le tableau")
    - Sinon**
      - Ecrire (X, " n'existe pas dans le tableau")
      - Finsi**
  - 2) **Fin** Affiche