esprit

Ecole Supérieure Privée
d'Ingénierie et de Technologies

Algorithmique et Programmation C

2012/2013

Plan du cours

- Chapitre 1 : notions de base
- Chapitre 2 : Les structures conditionnelles
- Chapitre 3 : Les structures Répétitives
- Chapitre 4 : Les tableaux
- Chapitre 5 : Les chaînes de caractères
- Chapitre 6 : Les fonctions
- Chapitre 7 : Les pointeurs

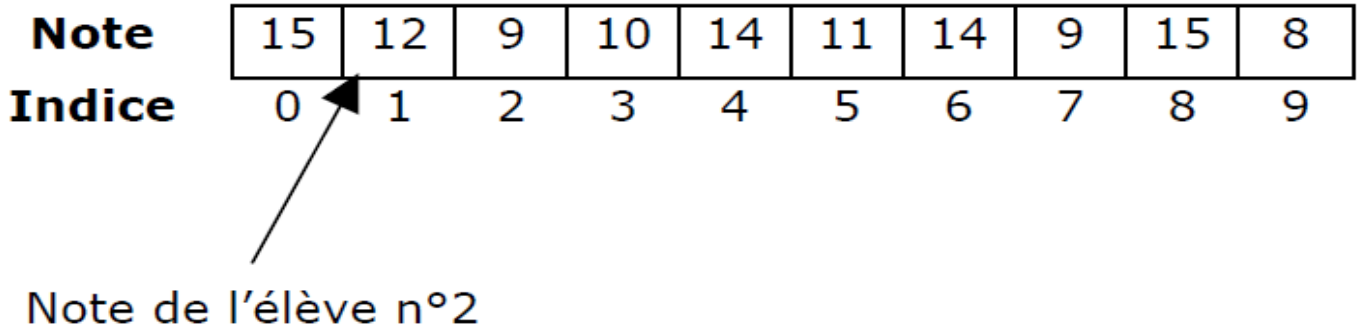
Introduction

- Problème:
  - Pour conserver par exemple simultanément les notes de 10 élèves, il nous faut 10 variables différentes (et si on avait 100 ou 1000 élèves?!!).
- Inconvénients:
  - Un nom pour chaque variable;
  - Aucun lien entre les différentes variables.

Introduction

- Solution:

Disposer d'un objet plus complexe, pour stocker ces notes, et y accéder à l'aide d'un indice :



- Intérêts:
  - Gain de temps;
  - rétrécissement du volume du programme;
  - possibilité de réutilisation de toutes les valeurs ultérieurement dans le programme.

Définition

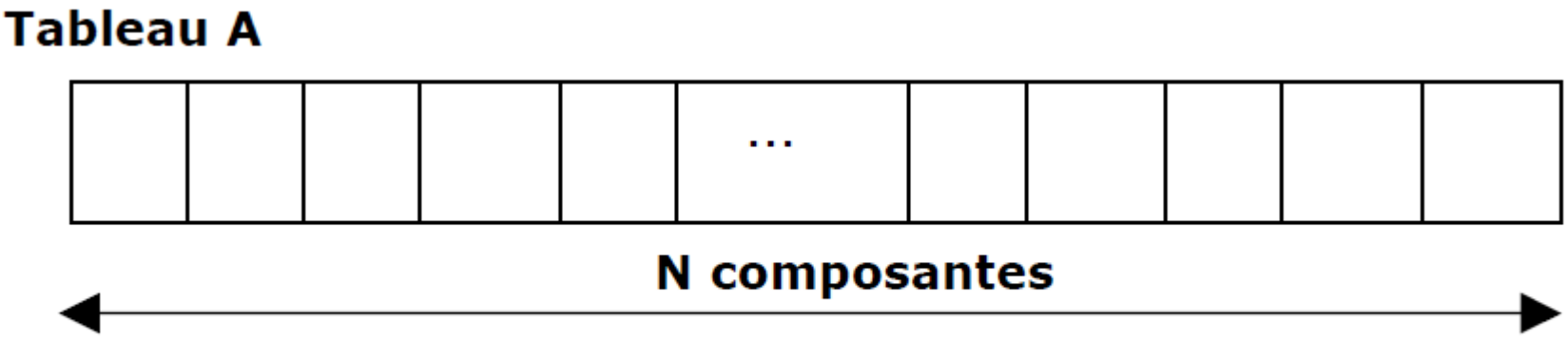
- Un tableau est une structure de donnée permettant de stocker des données de même type.

- Chacune des valeurs est repérée par un indice indiquant la position de la donnée dans le tableau.

- Un tableau est caractérisé par
  - Son nom;
  - Sa taille;
  - Sa dimension;
  - Le type de ses éléments.

Tableau à une dimension (uni-dimensionnel)

- Un tableau uni-dimensionnel (vecteur) est une manière de ranger des éléments ou des valeurs de même type.
- Il regroupe ces éléments dans une structure fixe et permet d'accéder à chaque élément par l'intermédiaire de son rang ou indice.



Déclaration

- En langage C, la déclaration d'un tableau à une dimension est définie comme suit :

  <TypeSimple> <NomTableau> [<Taille>];

- Exemples:

  float Notes [20];

  int A [100];

  char Alphabet [26];

Déclaration

- <Taille> est nécessairement une valeur numérique. Ce ne peut être en aucun cas une combinaison des variables du programme
- Pour un tableau de taille N, le compilateur C réserve N places en mémoire pour ranger les éléments du tableau
- Un élément du tableau est repéré par son indice. En langage C les tableaux commencent à l'indice 0. L'indice maximum est donc N-1
- En langage C, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau
- Les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse
- Espace occupé en mémoire:Taille d'une composante en octets x N
- Exemples:
  - short A [4]; /* Taille = 2 octets x 4 */
  - int TAB [10]; /* Taille = 4 octets x 10 */

Initialisation

- Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau en indiquant la liste des valeurs respectivement entre {}
- Exemples:
  - short A [5] = {1, 10, 9, 4, 5} ;
    Réservation de (2 x 5) octets
  - char Lettres [26] = {' A ', ' B '};
    Les autres composantes initialisées à 0
  - float B [ ] = {2.4, 1.6, 3.33};
    Réservation automatique de (4x3) octets (la dimension n 'est pas indiquée explicitement)
  - short A [4] = {1, 10, 9, 4, 5} ;
    Erreur!

Accès

- Syntaxe:

  <NomTableau> [<indice>]

- Exemple:
  - Pour un tableau T de taille N:

    T[0] pour accéder au premier élément

    T[N-1] pour accéder au dernier élément

Affectation

- Syntaxe:
  <NomTableau> [<indice>] = <expression>;

- Exemples:

```
T[2] = 3;
int Notes [100];
int i;
for (i = 0; i < 100; i++) /* Remplissage du tableau */
{
    printf ("Entrez la note de l 'étudiant N° %d\n", i+1 );
    scanf ("%d ", &Notes[i]);
}
```

Les tableaux

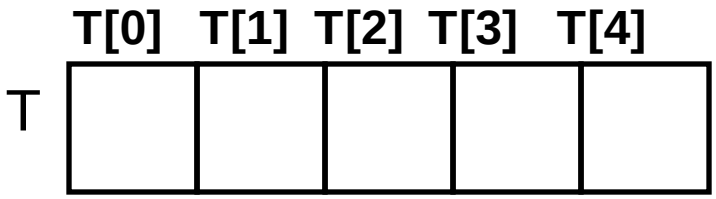
Nombre d'éléments

indice

```
int T[5];
```

Type des éléments

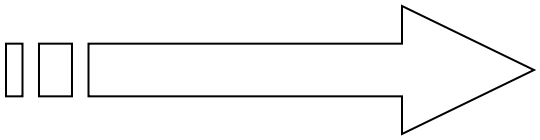
Nom



En C:

1. *Les indices commencent par 0*
2. *T[i] : i+1ème élément*

```
int Premiers [4];

Premiers [0] = 2;
Premiers [1] = 3;
Premiers [2] = 5;
Premiers [3] = 7;
printf("Le 3e nombre premier: %d\n", Premiers [2]);
printf("Somme du 2e et du 3e: %d", Premiers[1]+ Premiers[2]);
```



```
Le 3e nombre premier: 5
Somme du 2e et du 3e: 8
```

Remplir un tableau

- Remplir un tableau à l'aide d'une boucle

```
int T [4], i, x;
for (i= 0; i< 4; i++)
{
   printf ("Donnez l'élément n°:%d", i);
   scanf (“%d”, &x);
   T[i] = x;
}
```

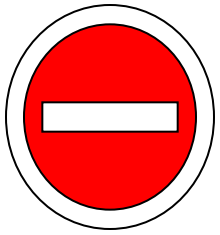
`scanf (“%d”, &T[i]);`

- On peut initialiser un tableau lors de sa déclaration

```
int Tpremiers [4] = {2, 3, 5, 7};
```

Et même:

```
int Tpremiers [] = {2, 3, 5, 7};
```

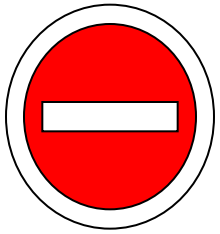


Par contre, on ne peut pas utiliser l'affectation !

~~int Tpremiers [4];~~
~~Tpremiers = {2, 3, 5, 7};~~

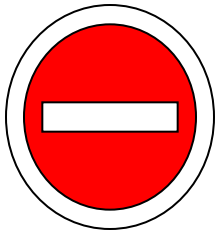
Exemple: Copie, comparaison: Attention !

On ne peut pas affecter un tableau à un autre par =



```
int T1 [4] = {2, 3, 5, 7};
int T2 [4];
T2 = T1;
```

Ni comparer un tableau avec un autre par ==



```
int T1 [4] = {2, 3, 5, 7};
int T2 [4] = {2, 3, 5, 7};
if (T1 == T2)
   printf ("Tableaux identiques.");
else
   printf ("Tableaux différents.");
```

Tableau Multidimensionnel

- extensions des tableaux à un seul indice
- représenter des objets plus complexes comme par exemple les matrices.
- Un tableau à N dimensions est en fait un tableau unidimensionnel de tableaux de N-1 dimensions.
- un tableau A à 2 dimensions est à interpréter comme un tableau (uni-dimensionnel) de taille L dont chaque composante est un tableau (uni-dimensionnel) de taille C.
- Donc on a :
  L : Nombre de lignes
  C : Nombre de colonnes
  L x C : Nombre de composantes

Déclaration

- Déclaration :

  <TypeSimple> <NomTabl> [<TailleLigne>][<TailleCol>];

- Exemples:

  ```
  int Notes [3][20]; /* 20 étudiants, 3 matières */
  char T[12][18];
  float A[2][2];
  ```

- Le nom d'un tableau est le représentant de l'adresse du premier élément du tableau

- Les composantes d'un tableau à 2 dimensions sont stockés ligne par ligne dans la mémoire

- Espace occupé en mémoire:
  - Taille d'une composante en octets x L x C
  - Exemple:
  - short A [4][5];/* Taille = 2 octets x 4 x5 */

Initialisation

- Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau en indiquant la liste des valeurs respectivement entre {}.

- A l 'intérieur de la liste, les composantes de chaque ligne sont entre {}

Exemples

- short A [3][5] = { {1, 10, 9, 4, 5}, {1, 1, 3, 4, 5},{1, 10, 3, 4, 4}};
  Réservation de (2 x 3 x 5) octets

- int C [4][4] = {{1, 1, 1, 1}};
  Les autres composantes initialisées à 0

- float B [ ][3] = { {2.4, 1.6, 3.33},{0.5, 0.2, 0.1}} ;
  Réservation automatique de (4x2 x3) octets (Pour les lignes seulement!)

- int C [4][4] = {{1, 1, 1, 1, 1}};
  Erreur!

Accès

- Syntaxe:

  <NomTableau> [<Ligne>][<Colonne>]

- Exemple:

  Pour un tableau T de taille L et C :

  - T[0][0] pour accéder au premier élément
  - T[L-1][C-1][pour accéder au dernier élément

Affectation

- Syntaxe:
  <NomTableau> [<Ligne>][<Colonne>] = <expression>;

- Exemples:

```
T[0][1] = 2;
int Notes [5][100];
int i, j;

for (i = 0; i < 5; i++)
{
        printf ("Matière %d \n", i+1);

        for (j=0; j < 100; j++)
        {
        printf ("Entrez la note de l 'étudiant n° %d\n", j+1);
        scanf ("%d ", &Notes[i][j]);
        }

}
```

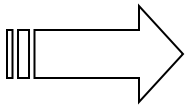
Exercices

- Lecture / affichage
- Somme des éléments
- Max et indice
- Insertion d'un élément (indice)
- Suppression d'un élément (indice)

Utilisation

Indexation par une variable

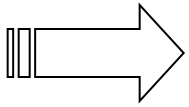
```
int T [4];
int i = 3;
T[0]=2; T[1]=3; T[2]=5; T[3]=7;
printf ("L'élément d'indice %d est %d", i, T[i]);
```



```
L'élément d'indice 3 est 7
```

- Parcours d'un tableau à l'aide d'une boucle

```
int T [4];
int i;
T[0]=2; T[1]=3; T[2]=5; T[3]=7;
for (i= 0; i< 4; i++)
{
 printf ("Indice %d: Valeur %d\n", i, T[i]);
}
```



```
Indice 0: Valeur 2
Indice 1: Valeur 3
Indice 2: Valeur 5
Indice 3: Valeur 7
```

Plan

- RECHERCHE DICHOTOMIQUE
- TRI PAR SELECTION
- TRI A BULLE
- Tri par insertion

Introduction

- On suppose qu'on se donne une suite de N nombres entiers et on veut les ranger en ordre croissant au sens large.

- Ainsi pour N = 10, la suite :
  (17, 8, 10, 25, 11, 5, 13, 5, 39, 9)

devra devenir
  (5, 5, 8, 9, 10, 11, 13, 17, 25, 39)

- Ce problème est un classique de l'informatique et en tant qu'algorithme pratique, on le rencontre souvent. Par exemple, pour classer des élèves selon les notes obtenues, mettre en ordre un dictionnaire, établir un annuaire téléphonique …

- Il faudrait bien faire la distinction entre le tri d'un grand nombre d'éléments (plusieurs centaines), et le tri de quelques éléments (une dizaine). Dans ce dernier cas, la méthode importe peu.

- En général, on exige que le tri se fasse au même endroit que la suite initiale.

Introduction

- On peut bien sûr trier autre chose que des entiers. Il suffit de disposer d'un domaine de valeurs muni d'une relation d'ordre total.

- On peut donc trier des caractères, des mots en ordre alphabétique, des enregistrements selon un certain champ. On suppose, pour simplifier, dans tout ce qui suit que l'on trie des nombres entiers.

- Dans ce cours, il sera question de complexité d'algorithme de tri, c'està-dire du nombre d'opérations élémentaires comparaisons, échanges (déplacements des éléments).

Recherche dans un tableau d'entiers

Problème: Rechercher dans un tableau d'entiers A une valeur VAL entrée au clavier. Afficher la position de VAL si elle se trouve dans le tableau, sinon afficher un message correspondant. La valeur POS qui est utilisée pour mémoriser la position de la valeur dans le tableau, aura la valeur -1 aussi longtemps que VAL n'a pas été trouvée.

Considérer deux cas :

a)Le tableau n'est pas ordonné

Implémenter la recherche séquentielle

b)Le tableau est ordonné

Implémenter la recherche séquentielle

Implémenter la recherche dichotomique, dont le principe est le suivant : Comparer le nombre recherché à la valeur au milieu du tableau, s'il y a égalité ou si le tableau est épuisé, arrêter le traitement avec un message correspondant.

si la valeur recherchée précède la valeur actuelle du tableau, continuer la recherche dans le demi- tableau à gauche de la position actuelle.

si la valeur recherchée suit la valeur actuelle du tableau, continuer la recherche dans le demi- tableau à droite de la position actuelle.

RECHERCHE DICHOTOMIQUE

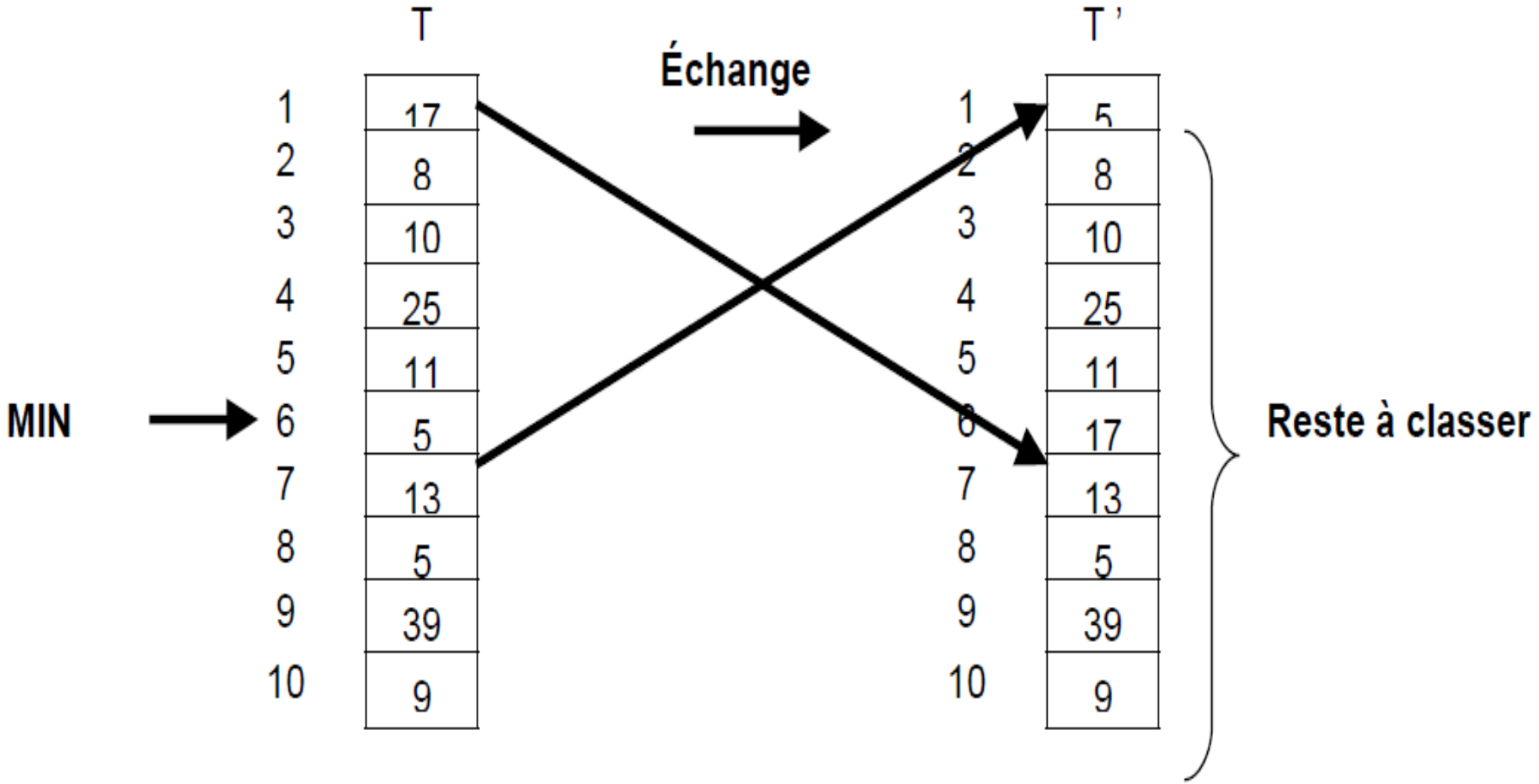
```
#include<stdio.h>
int main()
{
        int iTableau[]={1,2,3,5,6,8,9}; /* Tableau TRIE d'entiers */
        int iRecherche, iPremier, iDernier, iMilieu, iTrouve;

        printf("Quel élément recherchez-vous ? ");
        scanf("%d",&iRecherche);

        iPremier=0;
        iDernier=6;
        iTrouve=0;


while((iPremier <= iDernier)&&(iTrouve==0))
{
iMilieu=(iPremier+iDernier)/2;
if(iTableau[iMilieu]==iRecherche)
 iTrouve =1;
else
{
if(iTableau[iMilieu]>iRecherche)
 iDernier = iMilieu -1;
else iPremier = iMilieu +1;
}
}
if(!iTrouve) printf("Cette valeur n'appartient pas à la liste\n");
else printf("Cette valeur appartient à la liste\n");
printf("Voulez-vous continuer ? (Taper 0 pour sortir du programme) : ");
```

TRI PAR SELECTION

Le tri par sélection est souvent celui auquel on pense naturellement lorsqu'on est confronté pour la première fois à l'écriture d'un algorithme de tri.
Son fonctionnement est simple :

- Trouvons d'abord l'emplacement du minimum parmi les indices de 1 à N.
- Lorsque cet emplacement est déterminé, échangeons son contenu avec celui se trouvant à l'emplacement 1.
- Ensuite, trouvons l'emplacement du minimum parmi les indices de 2 à N puis, échangeons son contenu avec celui de l'emplacement 2.
- Ensuite, trouvons l'emplacement du minimum parmi les indices de 3 à N puis, échangeons son contenu avec celui de l'emplacement 3.
- On continue de la sorte jusqu'à ce qu'il ne reste qu'un seul emplacement à parcourir.

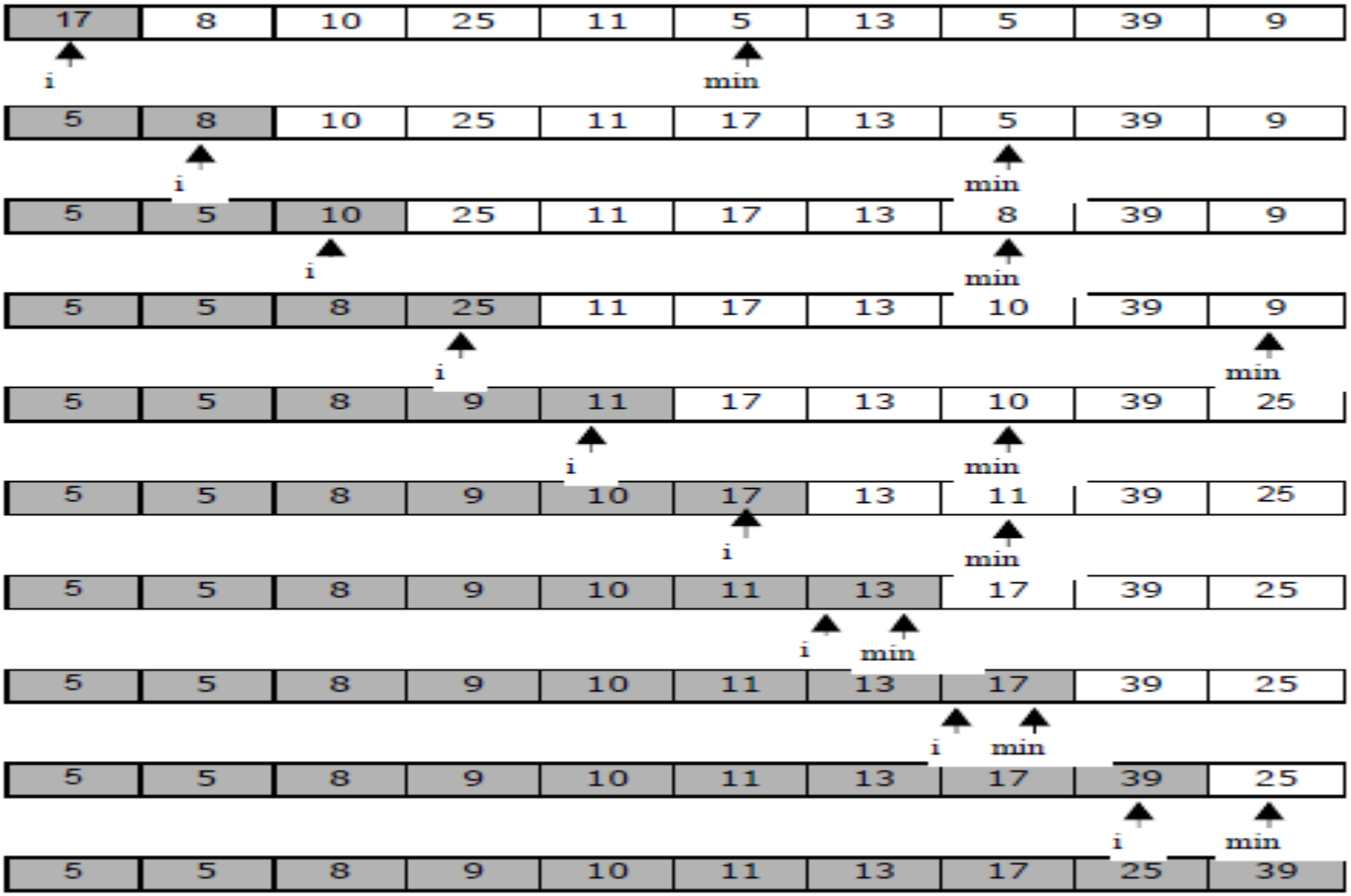
TRI PAR SELECTION



Programme

```
int tab[], int N;
int i, j, min, tampon ;

for (i=0;i<N-1;i++)
{
min=i ;
for (j=i+1;j<N;j++)
{
      if (tab[j])<tab[min])
      {
            min=j;
           }
}
tampon=tab[i];
tab[i];=tab[min];
tab[min]=tampon;
}
```

Exemple


17 8 10 25 11 5 13 5 39 9
i
min
5 8 10 25 11 17 13 5 39 9
i
min
5 5 10 25 11 17 13 8 39 9
i
min
5 5 8 25 11 17 13 10 39 9
i
min
5 5 8 9 11 17 13 10 39 25
i
min
5 5 8 9 10 17 13 11 39 25
i
min
5 5 8 9 10 11 13 17 39 25
i
min
5 5 8 9 10 11 13 17 39 25
i
min
5 5 8 9 10 11 13 17 39 25
i
min
5 5 8 9 10 11 13 17 25 39

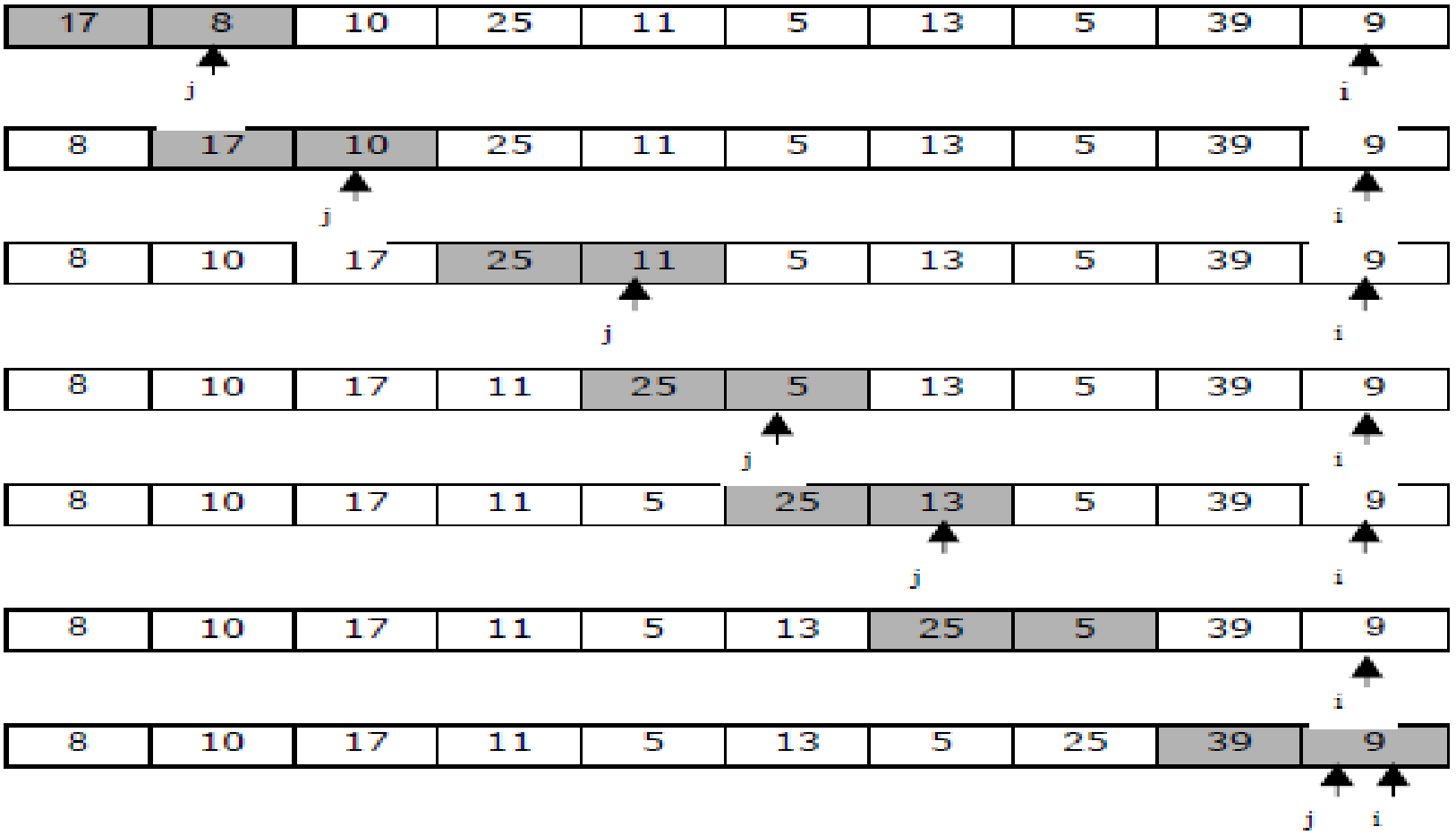
conclusion

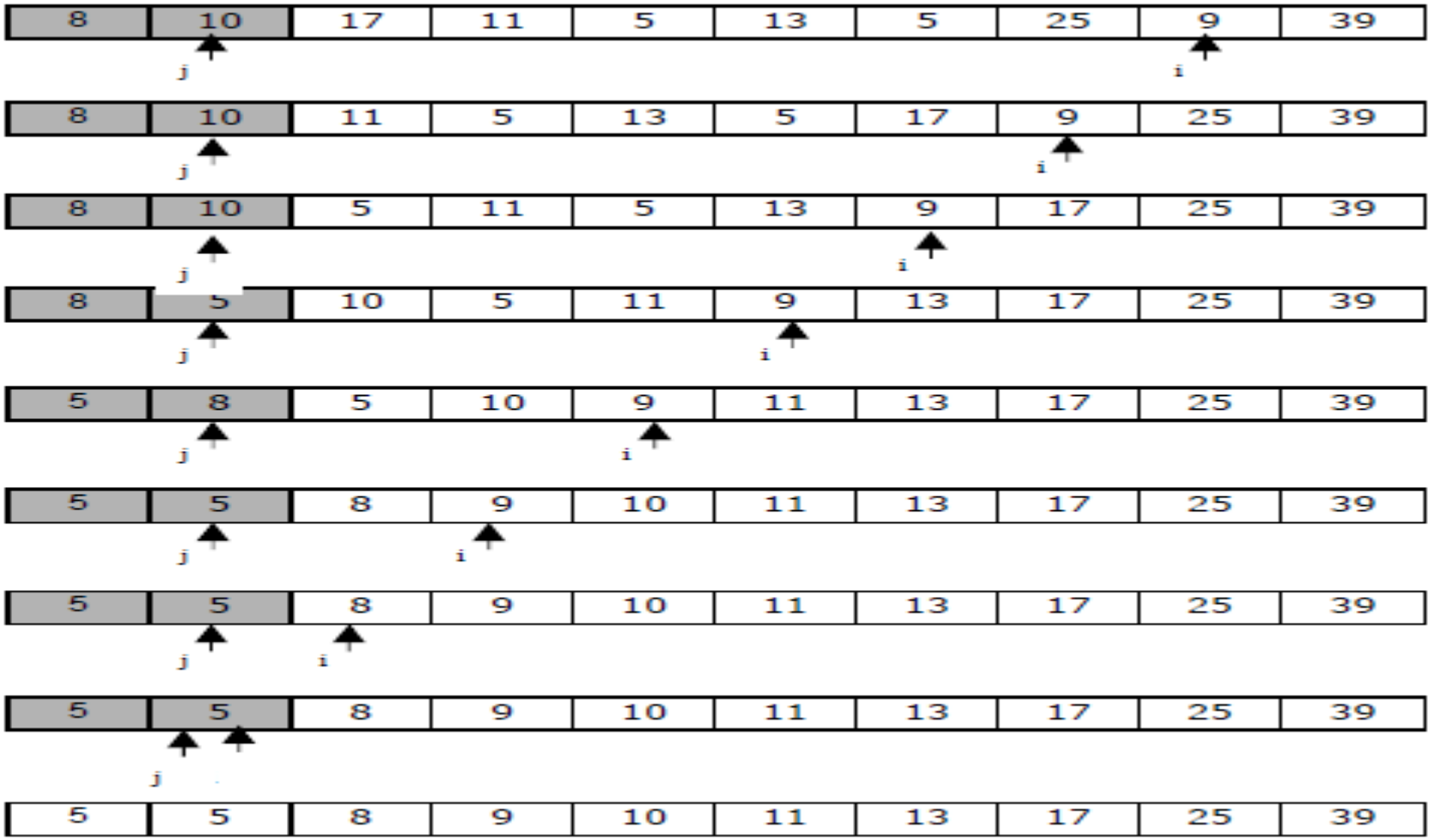
Si l'on considère le pire des cas (ex. les entiers sont présentés en ordre décroissant et l'on doit les replacer en ordre croissant). On fera successivement:
N-1 comparaisons lors de la première passe
N-2 comparaisons lors de la seconde passe
N-3 comparaisons lors de la troisième passe
...
1 comparaison lors de la N-1 ième passe.
En gros, on aura: 1 + 2 + ... + N-1 comparaisons, soit : N * (N -1)/2 ou (N2 - N) /2. C'est pourquoi l'on dit que cet algorithme est d'ORDRE N2/2. En analyse d'algorithme, on exprime cela avec la notation "Grand O" de la façon suivante : O(N2/2).
Une propriété importante de cet algorithme, en dehors de sa simplicité, est qu'il bouge très peu les éléments. On a au maximum 3N déplacements d'enregistrements (qui correspondent à N échanges). Donc le nombre d'échanges est O(N).
Ce tri n'est pas très avantageux pour les tableaux dont une partie est déjà triée, par contre pour les tableaux qui sont totalement désordonnés ce tri peut être très intéressant.

TRI A BULLE

- On compare le premier élément avec le deuxième et on les échange si le premier est plus grand que le second.
- On compare ensuite le second avec le troisième (le second peut ainsi contenir l'ancien premier) et on les échange si le second est plus grand que le troisième.
- On continue jusqu'à la fin de la première passe.
- À la fin de cette passe, le plus grand est nécessairement au bout du tableau.
- On recommence le même processus mais cette fois, on se rendra jusqu'à l'indice N-1 ce qui aura pour effet de repousser (faire remonter comme une bulle) le prochain plus grand à sa place.
- On l'appelle le tri bulle parce que son fonctionnement rappelle la montée des bulles dans un verre de boisson gazeuse par exemple.

Exemple


17 8 10 25 11 5 13 5 39 9
j
i
8 17 10 25 11 5 13 5 39 9
j
i
8 10 17 25 11 5 13 5 39 9
j
i
8 10 17 11 25 5 13 5 39 9
j
i
8 10 17 11 5 25 13 5 39 9
j
i
8 10 17 11 5 13 25 5 39 9
i
8 10 17 11 5 13 5 25 39 9
j
i


8 10 17 11 5 13 5 25 9 39
j
i
8 10 11 5 13 5 17 9 25 39
j
i
8 10 5 11 5 13 9 17 25 39
j
i
8 5 10 5 11 9 13 17 25 39
j
i
5 8 5 10 9 11 13 17 25 39
j
i
5 5 8 9 10 11 13 17 25 39
j
i
5 5 8 9 10 11 13 17 25 39
j
i
5 5 8 9 10 11 13 17 25 39
j
5 5 8 9 10 11 13 17 25 39

programme

```
void tri_bulle(int tab[], int N)
{
int i, j, tampon ;
for (i=N-1;i>0;i--)
for (j=1;j<=i;j++)
if (tab[j-1])>tab[j])
{
tampon=tab[j-1];
tab[j-1];=tab[j];
tab[j]=tampon;
}
}
```

conclusion

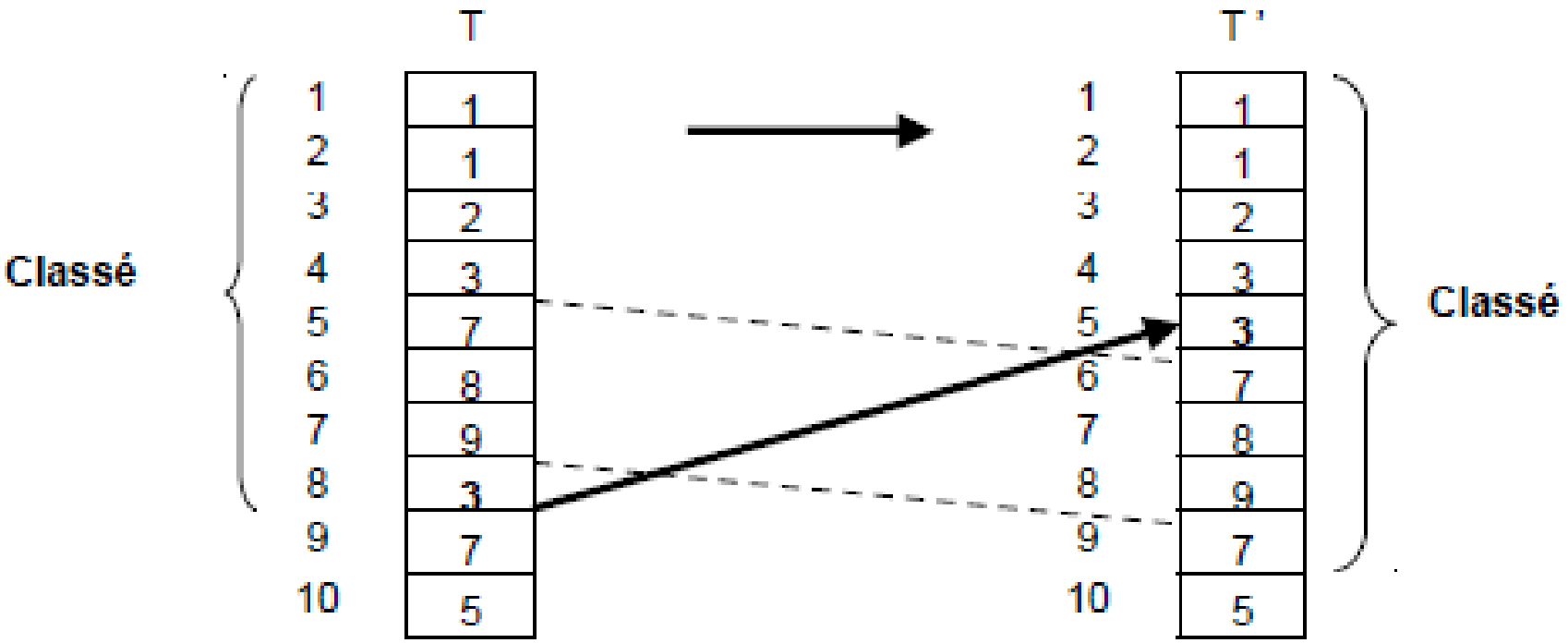
La complexité du **tri bulle est la suivante :**

- le nombre de comparaisons est de l'ordre de N2/2.
- le nombre d'échanges est de l'ordre de N2 /2.

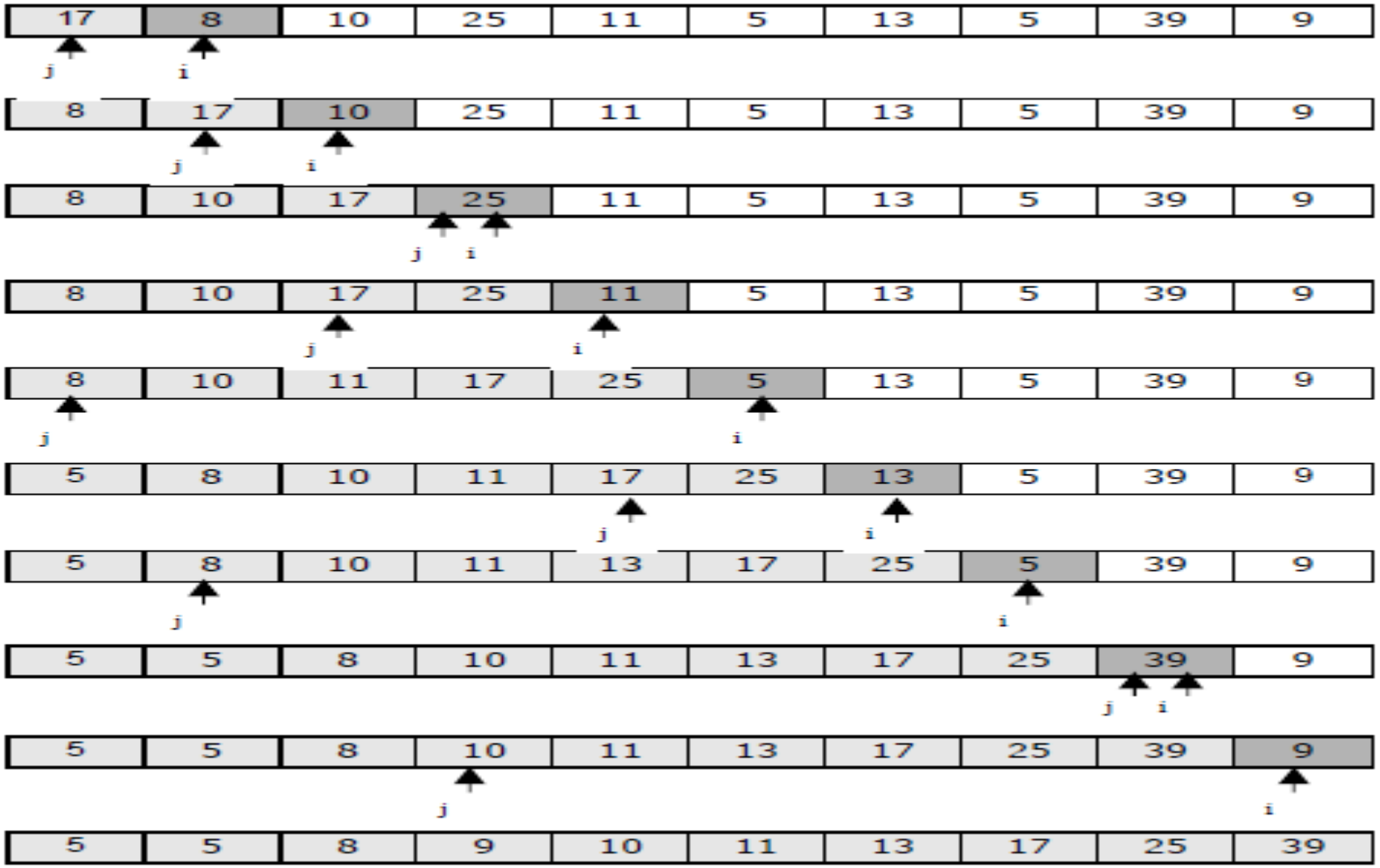
Ce tri exige énormément de déplacements, il ne peut être avantageux que pour un tableau pré-trié où les éléments ne sont pas trop éloignés de leur position finale.

Tri par insertion

- La méthode s'apparente à celle que beaucoup de joueurs utilisent pour trier leurs mains, après la donne,au jeu de cartes.
- Considérer que le premier élément est à sa place.
- Prendre le second élément et l'insérer dans la partie triée (celle-ci est de longueur 1 pour l'instant) en déplaçant les éléments triés au besoin pour faire de la place.
- Prendre le troisième élément et l'insérer dans la partie triée (celle-ci est de longueur 2 maintenant) en déplaçant les éléments triés au besoin pour faire de la place.
- Prendre le quatrième élément et l'insérer dans la partie triée (celle-ci est de longueur 3 maintenant) en déplaçant les éléments triés au besoin pour faire de la place.
- On continue jusqu'à l'élément N.


T
T '
Classé
Classé
1
2
3
4
5
6
7
8
9
10
1
1
2
3
7
8
9
3
7
5
1
2
3
4
5
6
7
8
9
10
1
1
2
3
3
7
8
9
7
5

Exemple


17 8 10 25 11 5 13 5 39 9
j i
8 17 10 25 11 5 13 5 39 9
j i
8 10 17 25 11 5 13 5 39 9
j i
8 10 17 25 11 5 13 5 39 9
j i
8 10 11 17 25 5 13 5 39 9
j i
5 8 10 11 17 25 13 5 39 9
j i
5 8 10 11 13 17 25 5 39 9
j i
5 5 8 10 11 13 17 25 39 9
j i
5 5 8 10 11 13 17 25 39 9
j i
5 5 8 9 10 11 13 17 25 39

Programme

```
void tri_Insertion(int tab[], int N)
{
int i, j, elem;
for (i=1;i<N;i++)
{
      elem= tab[i]); j=i;
      while(j>0 && tab[j-1])>elem)
{
tab[j]= tab[j-1];
j--;
}
tab[j]=elem;
}
}
```

conclusion

La complexité du **tri par insertion est la suivante :**

- le nombre de comparaisons est de l'ordre de N2/4.
- le nombre d'échanges est de l'ordre de N2/8.

- Ce tri peut être intéressant pour l'insertion de données dans un tableau déjà pré-trié.