Chapitre 1 : Partie 2

Recherches :

séquentielle, dichotomique

Tris :

sélection, bulle, insertion, shell

ALGORITHMIQUE
Recherche et Tris

- **RECHERCHES** : relation d'équivalence du style « égal ».
- **TRIS** : relation d'ordre total du style « inférieur ou égal ».
- Application possible : entiers, réels et chaînes de caractères (strcmp)
- Application impossible : nombres complexes et structures (en C).
- Pour les structures en C il faut disposer d'un **champ** correspondant à la **clé** de recherche et/ou tri.

Algorithmes présentés

RECHERCHES

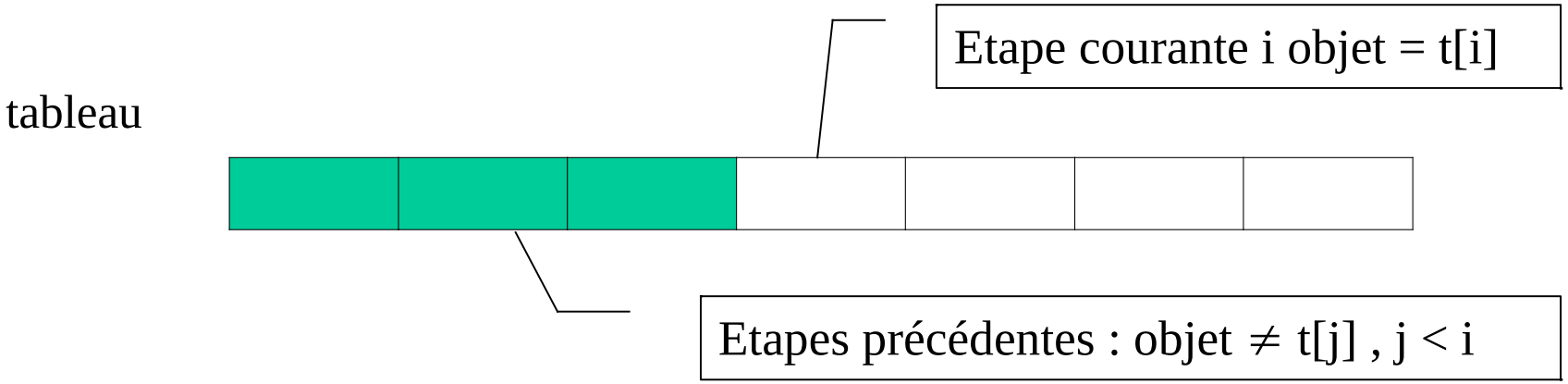
- Séquentielle (linéaire)
- Dichotomique (dans un tableau ou une liste triée)

TRIS

- Tri par insertion
- Tri par sélection
- Tri par bulle v1 et v2
- Quicksort (voir séance récursivité)

Recherche séquentielle ou linéaire Principe

- Principe : comparer les uns après les autres tous les éléments du tableau avec l'objet recherché. Arrêt si :
  - l'objet a été trouvé
  - tous les éléments ont été passés en revue et l'objet n'a pas été trouvé



Recherche séquentielle Programme

```
int rechSequentielle(int t[], int quoi, int
max)
{
        int i;
        int arret= 0;
        i= 0;
        while( (i < max) && ( arret == 0))
        {       if (t[i] == quoi)
                        arret = 1;
                else
                        i++;
        }
        if (i == max)
                return -1;
        else
                .       .
}
```

Recherche dichotomique
Principe

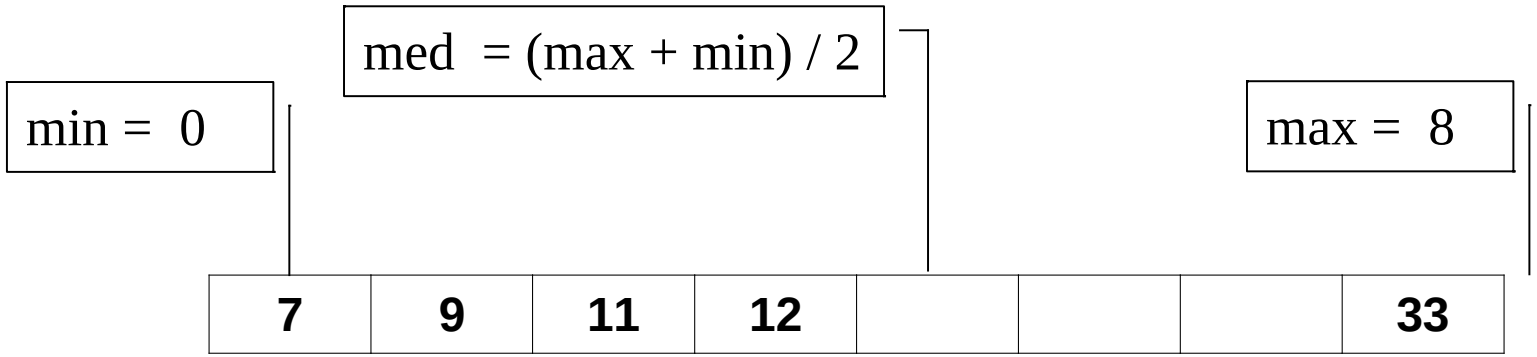
ATTENTION : toute collection d'objet préalablement triée.

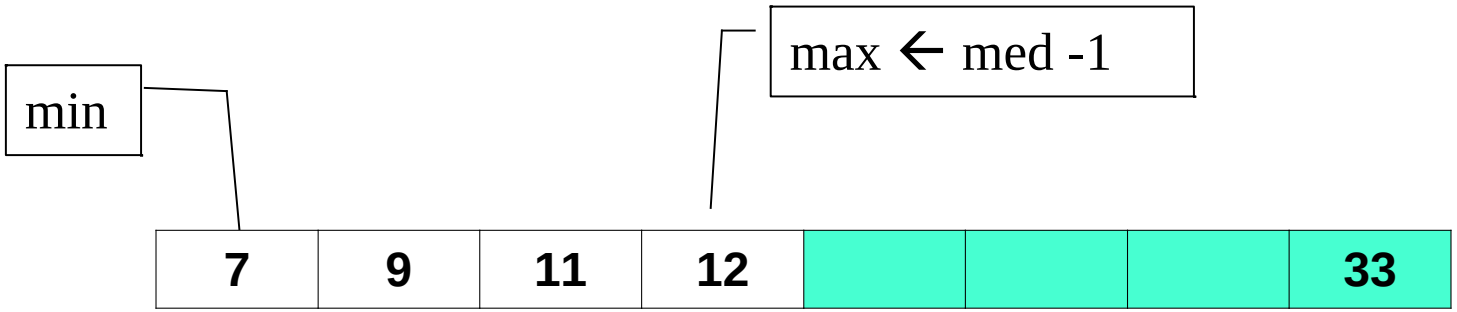
A chaque étape :

- découpage du tableau en deux sous-tableau à l'aide d'un indice médian (tableau inférieur) et (tableau supérieur)
- comparaison de la valeur située à l'indice moyen et de l'objet recherché,
  1. si l'objet est supérieur ou égal à la valeur t[moyen] relancer la recherche avec le tableau supérieur,
  2. sinon relancer la recherche avec le tableau inférieur.

Recherche dichotomique
Exemple

quoi = 11 : quoi < t[med]





Recherche dichotomique Programme

```
int rechDichotomique(int t[], int quoi, int max)
{       int min = 0 , med, pos, arret = 0;

        do{     med= (max+min)/2;
                if(quoi ==  t[med])
                {       arret = 1;
                        pos = med;
                }
                else
                {       if(quoi > t[med])   min = med + 1;
                        else                 max = med - 1;
                }
                if(min > max)
                {       arret = 1;
                        pos = -1;
                }
        } while(arret != 1);
        return pos;
}
```

sortie de boucle (arret = 1)
ou (quoi = t[med])
ou (min > max)

Algorithmes de recherche complexité : exemple

Exemple sur une recherche de 256 objets (2^8)

- Recherche linéaire : Au pire 256 (n) en bout de tableau
  Au mieux en début du tableau 1
  En moyenne 128 (n/2)

- Recherche dichotomique : Etape 1 : 256 Etape 5 : 16
  Etape 2 : 128 Etape 6 : 8
  Etape 3 : 64 Etape 7 : 4
  Etape 4 : 32 Etape 8 : 2

Coût au pire, au mieux, en moyenne : $\log_2$ (256) soit 8. Attention disposer d'une collection triée.

Algorithmes de recherche complexité : évaluation

Type de recherche	**Cout moyen d'une recherche**	**Coût maximal (élément absent)**
Linéaire	n/2	n
Dichotomique	$\log_2 n$	$\log_2 n$

Le coût du tri est plus important qu'une simple recherche.

La comparaison entre une recherche simple et un tri suivi d'une recherche dichotomique est à l'avantage de la recherche simple.

Tris

Prérequis : existence d'une relation d'ordre total du type ≤ .

Généralisation :

Ensemble ou collection d'informations possédant une clé.

Existence d'une relation d'ordre total de type $\leq$ sur la clé

Exemple : ordre lexicographique *(utiliser strcmp en C)*

Dans la suite utilisation de tableaux d'entiers :

t[1..n] est ordonné : si pour tout i, j : i < j alors t[i]$\leq$ t[j]

Tri sélection - Principe

A chaque étape i :

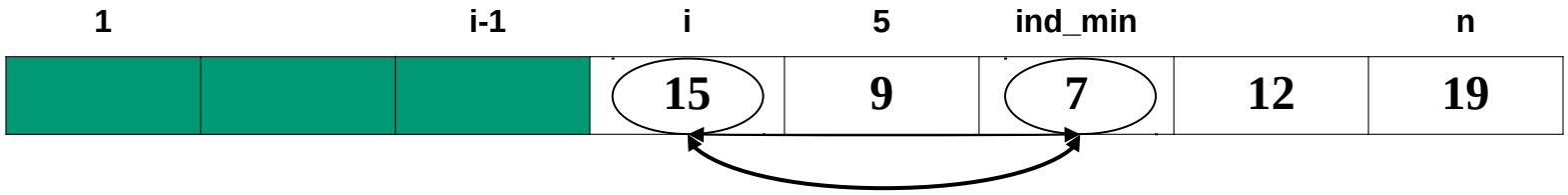
1. On recherche parmi les t[i],.. t[n] le plus petit élément qui doit être positionné à la place i.
2. Supposons cet élément à **ind_min** t[ind_min] et t[i] sont échangés.
3. i = i+1, retourner en 1

Nota : au cours du tri les éléments de 0 à i –1 sont tous ordonnés et bien placés

Tri sélection - Exemple

Eléments de 1 à i –1 bien placés

t[ind_min] le plus petit de tous les éléments entre i et n, à permuter avec t[i]



Tri sélection
Programme

```
1.  void triSelection(int t[ ], int max)
2.  {   int ind_min, i, j;
3.      int temp;
4.      i= 0;
5.      while(i <  max)
6.      {       ind_min=  i;
7.       // RECHERCHE DE L'INDICE CONTENANT
8.       // LE PLUS PETIT ELEMENT
9.
10.             temp= t[i];
11.             t[i]= t[ind_min];
12.             t[ind_min]= temp;
13.             i= i+ 1;
14.     }
15. }
```

Tri sélection Programme

```
1.  void triSelection(int t[ ], int max)
2.  {   int ind_min, i, j;
3.      int temp;
4.      i= 0;
5.      while(i <  max)
6.      {       ind_min=  i;
7.              for(j = i+1; j < max;  j++)
8.                      if(t[j] < t[ind_min])
9.                              ind_min= j;
10.             temp= t[i];
11.             t[i]= t[ind_min];
12.             t[ind_min]= temp;
13.             i= i+ 1;
14.     }
15. }
```

Tri bulle ou échange
Principe

Tableau de n éléments : au début max = n.

A la fin de chaque étape la partie du tableau de t[max+1,.. n] est constituée d'éléments bien placés et tous supérieurs aux éléments de t[0,.. max-1].

1. Pour les éléments i, de 0 à max-1 comparer et permuter successivement (si nécessaire) les éléments t[i] et t[i+1].
2. Faire max = max -1, (fin de l'étape) recommencer en 1

Arrêt de l'algorithme lorsque max = 0.

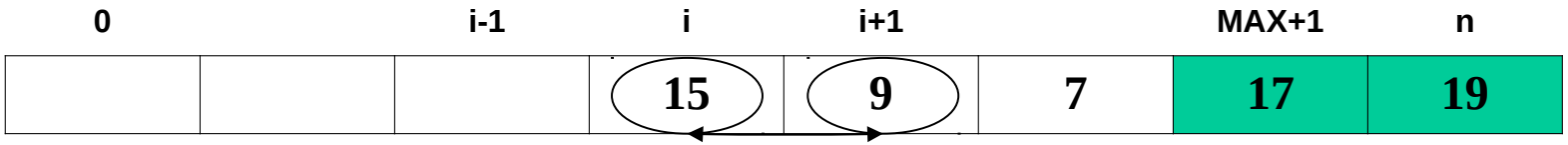
Remarque : Possibilité d'arrêter l'algorithme si à une étape il n'y a pas eu de changement.

Tri bulle - exemple

Tri bulle version de base

- De i = 1 à max -1

1. Comparaison et échange si nécessaire entre t[i] et t[i+1]
2. max = max -1
3. Retourner en 1



Tri bulle
Programme

```
1.  void triBulle(int t[ ], int max)
2.  {   int i, temp;
3.      while(max > 0)
4.      {       i = 0;
5.              while(i < max -1)
6.
7.                // COMPARAISON DES TERMES
8.                // ADJACENTS, ECHANGE SI
9.                // NECESSAIRE
10.
11.
12.
13.             max= max -1;
14.     }
15. }
```

Tri bulle
Programme

```
1.  void triBulle(int t[ ], int max)
2.  {   int i, temp;
3.      while(max > 0)
4.      {       i = 0;
5.              while(i < max -1)
6.              {       if( t[i] > t[i+1])
7.                      {  temp = t[i];
8.                              t[i] = t[i+1];
9.                              t[i+1] = temp;
10.                     }
11.                     i = i+1;
12.             }
13.             max= max -1;
14.     }
15. }
```

Tri bulle – améliorations

- **Test d'arrêt :** mise en place d'un indicateur d'échange effectués. Si au cours d'un passage aucun changement n'a été effectué : arrêt.

- **Parcours dans les deux sens :** on remarque que ce tri est particulièrement défavorable si il est dans l'ordre exactement inverse (valeurs élevées en bas du tableau). On met en place une double boucle pour :
  1. "pousser les valeurs les plus élevées vers le haut"
  2. "pousser les valeurs les plus petites vers le bas"

Tri insertion - Principe

A chaque étape i (indice courant i du tableau) :

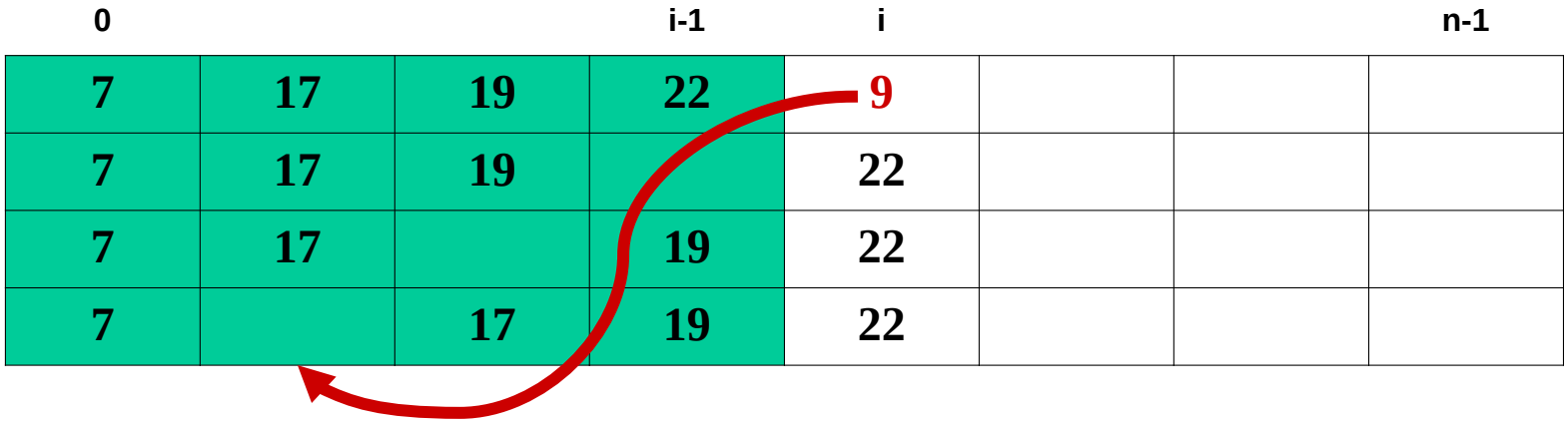
La partie du tableau de 0 à i-1 est déjà triée (les objets ne sont pas forcément à leur place finale)

1. la valeur courante en i est enlevée de sa place initiale créant un trou.
2. Les éléments de 0 à i-1 supérieurs à la valeur courante sont décalés vers la droite, déplaçant le trou (pos) vers la droite.
3. La valeur courante est insérée à la bonne place (pos) dans le trou.

Tri insertion - Exemple

Etape courante : i

temp = 9



Déplacement du trou vers la gauche tant que t[i] < t[pos]

Tri insertion Programme

```
1.  void triInsertion(int t[], int maxSaisi)
2.  {  int i, pos, temp;
3.
4.      for(i = 1; i< maxSaisi; i++)
5.      {  temp = t[i];
6.              pos= i;
7.              // DEPLACEMENT DU TROU
8.                // JUSQU'À LA BONNE POSITION
9.
10.
11.             t[pos] = temp;
12.     }
13.}
```

- les éléments de **1** à **i-1** sont triés mais pas définitivement placés
- **pos** est la position courante du trou
- recherche de la bonne place

Tri insertion Programme

```
1.  void triInsertion(int t[], int maxSaisi)
2.  {  int i, pos, temp;
3.
4.      for(i = 1; i< maxSaisi; i++)
5.      {  temp = t[i];
6.              pos= i;
7.              while((pos > 0) && (t[pos-1] > temp))
8.              {  t[pos]= t[pos-1];
9.                      pos= pos -1;
10.             }
11.             t[pos] = temp;
12.     }
13.}
```

- les éléments de **1** à **i-1** sont triés mais pas définitivement placés
- **pos** est la position courante du trou
- recherche de la bonne place

Tris - Complexité

- Le tri à bulle $O(n^2)$
- Le tri sélection $O(n^2)$
- Le tr insertion $O(n^2)$

Ces tris sont considérés comme inefficaces pour des problèmes de grande taille.

- Le tri rapide (quicksort) $O(n^2)$ au pire, $O(n.log(n))$ en moyenne.