Chapitre 3: Algorithmes de tri

Qu’est ce qu’un tri ?

En entrée, on dispose d’une séquence (a1, …, aN) de N données que l’on désire classer par ordre croissant ou décroissant s’il s’agit de valeurs numériques ou par **ordre alphabétique** pour des chaînes de caractères.

Exemple

On suppose qu'on se donne une suite de N nombres entiers (ai), et on veut les ranger en ordre croissant au sens large. Ainsi, pour N = 10, la suite

(18;3;10;25;9;3;11;13;23;8)

devient

(3;3;8;9;10;11;13;18;23;25)

Quoi trier ?

En général, les données à trier se composent :

- d'une **clé** c'est-à-dire les valeurs à trier (les clés appartiennent à un ensemble totalement ordonné) ;

Quoi trier ? (2)

- Pour être triées, les données doivent toutes être mémorisées. On suppose ici qu'il s'agit de données numériques à classer par ordre croissant et placées en mémoire centrale dans un tableau T, de taille N.
- On parle alors de **tri interne,** par opposition au **tri externe** qui s'applique à des données situées en mémoires auxiliaires (sous forme de fichiers).
- Très utilisées dans les applications tant scientifiques que de gestion, les méthodes de tri sont très importantes du point de vue méthodologique et constituent l'un des problèmes les plus fréquents à résoudre en informatique.

Pourquoi trier ?

Par exemple, il faut établir le classement des étudiants par ordre de mérite, mettre en ordre un dictionnaire, trier l'index d'un livre, . . .

Taille des données à trier (1)

Il faudra bien faire la distinction entre le tri d'un grand nombre d'éléments (plusieurs centaines), et le tri de quelques éléments (un paquet de cartes). Dans ce dernier cas, la méthode importe peu. Un algorithme amusant, **bogo-tri**, consiste a regarder si le paquet de cartes est déjà ordonne. Sinon, on le jette par terre. Et on recommence. Au bout d'un certain temps, on risque d'avoir les cartes ordonnées. Bien sûr, le **bogo-tri** peut ne pas se terminer. Une autre technique fréquemment utilisée avec un jeu de cartes consiste a regarder s'il n'y a pas une transposition à exécuter. Des qu'on en voit une à faire, on la fait et on recommence. Cette méthode marche très bien sur une bonne distribution de cartes.

Taille des données à trier (2)

Plus sérieusement, il faudra toujours avoir a l'esprit que le nombre d'objets à trier est important. Ce n'est pas la peine de trouver une méthode sophistiquée pour trier 10 éléments.

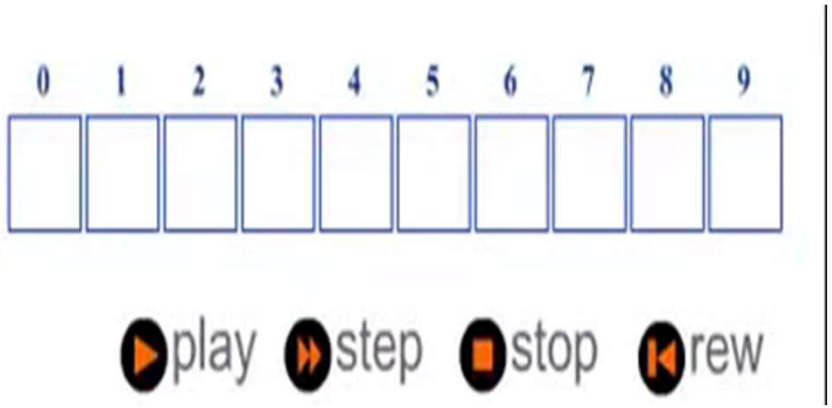
Contraintes lors d’un tri

En général, on exigera que le tri se fasse in situ, c'est-a-dire que le résultat soit au même endroit que la suite initiale. 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** (voir chapitre suivant) selon un certain champ.

TRI A BULLES

- Le principe du **tri à bulles** consiste à comparer deux éléments consécutifs et à les échanger s'ils ne sont pas dans le bon ordre. A l'étape i, on parcours la séquence à partir de la fin de manière à « faire remonter » le plus petit élément en T[i].
- **Métaphore des bulles :** Comme des bulles qui remontent à la surface d'un liquide, le plus grand élément, plus « léger » que les autres, gagne de proche en proche la « surface » (l'extrémité droite du tableau). Ce principe est répété pour chacun des éléments.

ETAPE DU TRI A BULLES



TRI A BULLES

Procedures tri_bulles(n :entier;variable t: tableau d'entiers)

Variables i, dernier:entier

Début

dernier←n-2;

Tant que (dernier >=0) faire

Debut

Pour i allant de 0 à dernier Faire

Si (t[i] > t[i+1]) **alors**

permuter(t[i], t[i+1]);

dernier←dernier – 1;

Fin

Fin

TRI A BULLES EN C

```
void tri_bulles(int *t, int n){
    int dernier=n-1;
    while(dernier >=0)
    {
      for(int i=0 ; i < n-1 ; i++)
        {
            if(t[i] > t[i+1])
                permuter(&t[i],&t[i+1]);
          }
        dernier--;
    }
}
```

TRI PAR SELECTION

- On parcourt le tableau pour chercher **le plus petit élément** qui est alors **permuté avec le premier élément du tableau**.
- On fait la même chose avec les **n-1** éléments restants du tableau sans le premier élément, puis avec les **n-2** éléments, ... et jusqu'à ce qu'il ne reste plus qu'un seul élément (le dernier, lequel est alors le plus grand élément).
  - Trouver le plus petit élément (**ppe**) et le mettre au début de la liste
  - Trouver le 2e ppe et le mettre en seconde position
  - Trouver le 3e ppe et le mettre à la 3e place, etc

ALGORITHME DU TRI PAR SELECTION

```
Procedures tri_sel(n :entier;variable t: tableau d'entier)
Variables i, j,imin:entiers
Début
Pour i allant de 0 à n-2 faire
    Début
        { trouver l'indice du min }
        imin ← i;
        Pour j allant de i +1 à n-1 faire
            Si t[imin]>t[j] alors imin ←j ;
                { permuter le min avec le ppe}
        permuter (t[i], t[imin]);
    Fin(pour i)
Fin
```

TRI PAR SELECTION EN C

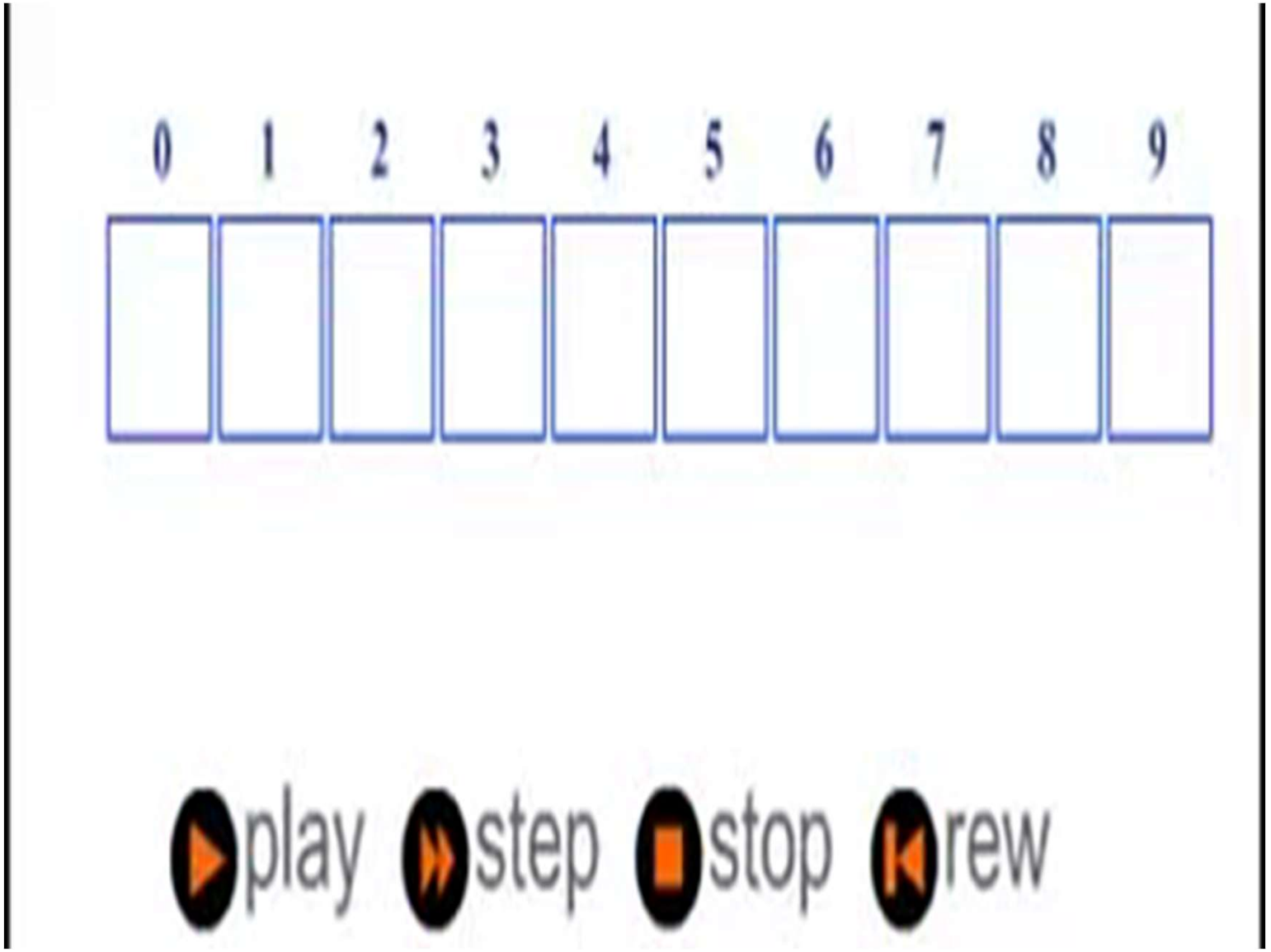
```
void tri_selection(int *t, int n)
{
    int i, imin, j, t;

    for (i = 0; i < n - 1; i++)
    {
        imin = i;
        for (j = i+1; j < n; j++)
            if (tableau[j] < t[imin])
                imin = j;
        permuter(&t[i],&t[imin]);

    }
}
```

TRI PAR INSERTION

- Insérer à la nième itération le nième élément à la bonne place.
- On choisit un élément du tableau,
- on trie les autres et on insère l'élément initialement choisi à la bonne place en parcourant le tableau.
- A la fin, chaque élément sera inséré à sa bonne place.


0 1 2 3 4 5 6 7 8 9
play
step
stop
rew

TRI PAR INSERTION

```
Procedures tri_inser(n :entier,variable t: tableau d'entier)
Variables : i, j, x : entiers;
Début
Pour i allant de 1 à n-1 faire
      Début
      x←t[i]
      j ←i
      Tant que (j>0) et (x<t[j-1]) faire
      Début
            t[j]←t[j-1]
            j←j-1
      Fin(tant que)
     t[j]←x
Fin(pour)
Fin
```

TRI PAR INSERTION EN C

```
void tri_insertion(int n,  int* t)
{
    int i, j;
    int x;

    for (i = 1; i < n; i++) {
        x = t[i];
        for (j = i; j > 0 && t[j - 1] > x; j--) {
            t[j] = t[j - 1];
        }
        t[j] = x;
    }
}
```

EXERCICES D'APPLICATION (1)

1. **Ecrire une fonction retournant un booléen permettant de tester si un tableau est trié.**
2. **Ecrire une fonction qui compte le nombre de fois où deux entiers successifs dans le tableaux ne sont pas ordonnés.**
3. **Ecrivez un algorithme qui permette de saisir un nombre quelconque de valeurs, et qui les range au fur et à mesure dans un tableau.**

Le programme, une fois la saisie terminée, doit dire si les éléments du tableau sont tous consécutifs ou non.

Par exemple, si le tableau est :

11	12	13	14	15	16	17

éléments sont tous consécutifs. En revanche, si le tableau est :

11	10	13	14	15	16	9

ses éléments ne sont pas tous consécutifs.

EXERCICES D'APPLICATION (2)

On considère l'algorithme suivant :

```
i ← 1
répéter
        indice_max ← i
        pour j ← i+1 à n faire
          si T[j] > T[indice_max] alors
                temp ← T[indice_max]
                T[indice_max] ← T[i]
                T[i] ← temp
                i ← i + 1
          fsi
        fpour j
jusqu'à i = n-1
```

1) Appliquez-le pour le tableau T ci-dessous :

1	3	1	9	2

2) De quel algorithme s'agit-il ?

EXERCICES D'APPLICATION (3)

Soit le tableau suivant:

1	2	3	4	5	6
41	56	20	31	59	15

Quel sera le contenu du tableau après :

a. la troisième itération en utilisant la méthode du tri par insertion séquentielle ?
b. la quatrième itération en utilisant la méthode du tri par bulles ?
c. La troisième itération en utilisant la méthode du tri par sélection ?

EXERCICES D'APPLICATION (3)

Soit le tableau suivant:

1	2	3	4	5	6
41	56	20	31	59	15

Quel sera le contenu du tableau après :

a. la troisième itération en utilisant la méthode du tri par insertion séquentielle ?
b. la quatrième itération en utilisant la méthode du tri par bulles ?
c. La troisième itération en utilisant la méthode du tri par sélection ?

TRI PAR FUSION

- On appelle **fusion** l'opération qui, à partir de deux suites classées de x et y valeurs, associe une seule suite classée de x+y valeurs.
- Supposons que l'on dispose de N valeurs à trier. On les sépare en deux sous-suites de N/2 valeurs que l'on trie chacune séparément puis que l'on fusionnera ultérieurement. Sur chacune des deux sous-suites, on recommence l'opération effectuée sur la suite initiale, jusqu'à n'obtenir que des suites de un ou deux éléments, donc triées ou faciles à trier, que l'on fusionne jusqu'à réobtenir la suite initiale cette fois-ci triée.

6 5 3 1 8 7 2 4

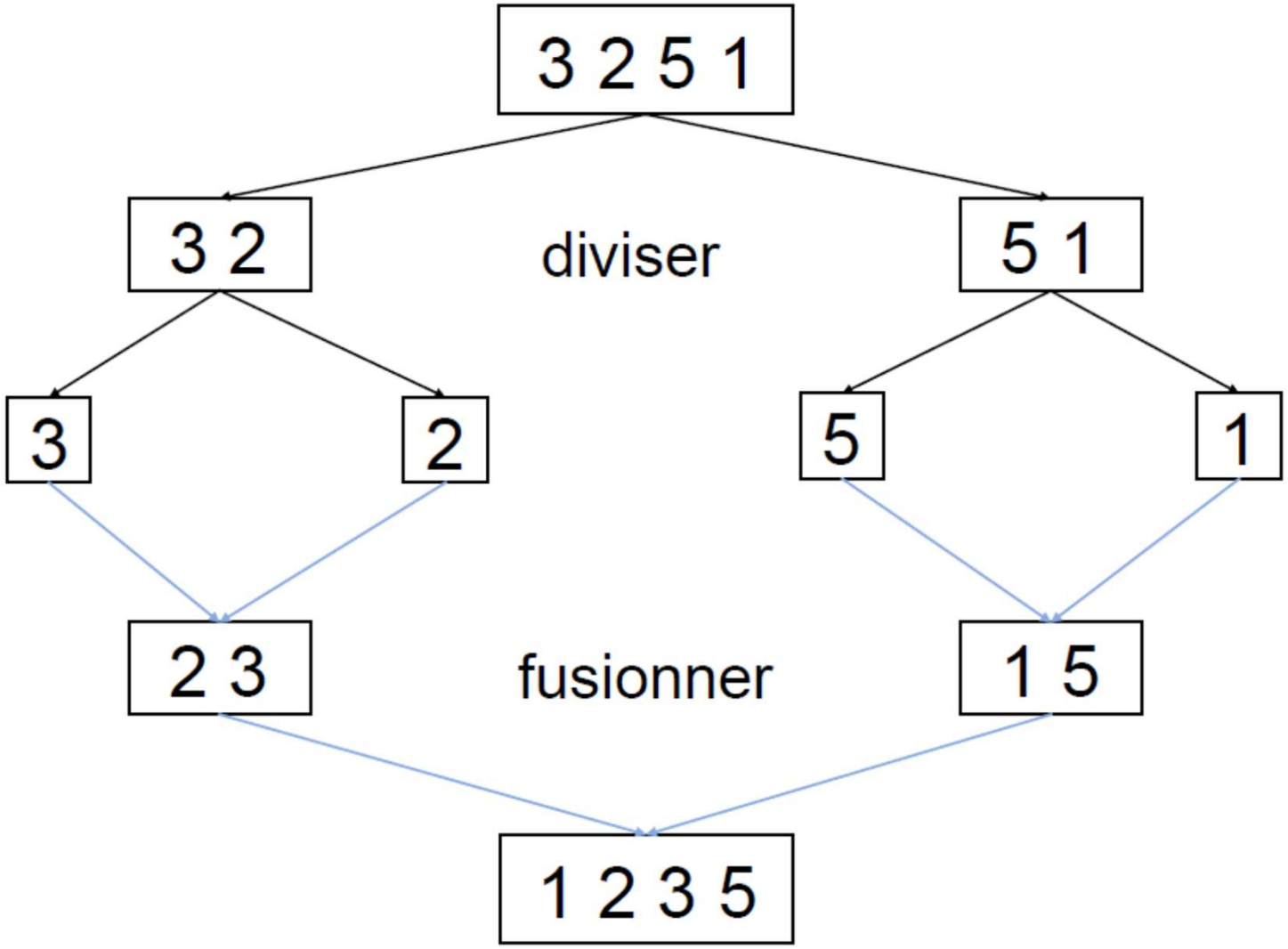
TRI PAR FUSION

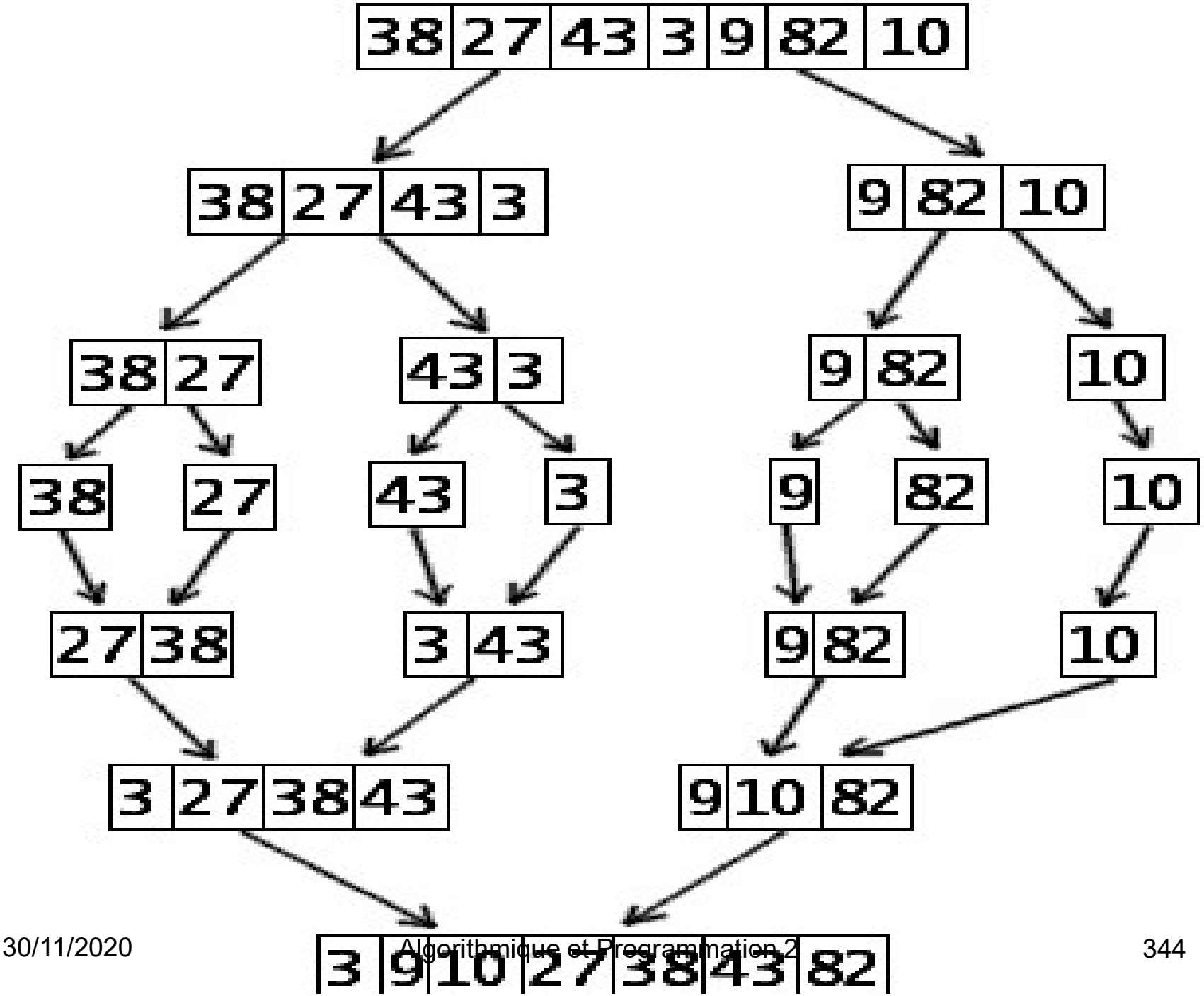
L'algorithme du **tri par fusion** peut aussi se décrire dans le paradigme « diviser pour régner » de la manière suivante :

1. Diviser : Diviser la séquence de N éléments à trier en deux sous-séquences de N/2 éléments.

2. Régner : Trier les deux sous-séquences récursivement.

3. Combiner : Fusionner les deux sous-séquences triées.


3 2 5 1
3 2
diviser
5 1
3
2
5
1
2 3
fusionner
1 5
1 2 3 5


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

TRI PAR FUSION

procedure triFusion (p, r : entiers, var t : tableau[0 à n-1] d'entiers)

Debut

si p < r **alors**

debut

    q ← (p + r) / 2

    triFusion(p, q, T)

    triFusion(q+1, r, T)

    fusionner(T, p, q, r)

fin

Fin

La procédure fusionner « fusionne » deux séquences triées dans l'ordre croissant.

```
i ← 1                    // indice de l'élément courant dans t1
j ← 1                    // indice de l'élément courant dans t2
k ← 1                    // indice de la position d'insertion dans t3
N ← longueur(t1)
P ← longueur(t2)
tant que  i ≤ N ou  j ≤ P faire
      si t1[i] ≤ t2[j] alors
            t3[k] ← t1[i]
            i ← i + 1
      sinon
            t3[k] ← t2[j]
            j ← j + 1
      fsi
      k ← k + 1
ftq                                   // i > N ou j > P
si  i > N alors
      tant que  j ≤ P  faire
            t3[k] ← t2[j]
            j ← j + 1
            k ← k + 1
      ftq
 sinon                                // i ≤ N
      tant que i ≤ N faire
            t3[k] ← t1[i]
            i ← i + 1
            k ← k + 1
       ftq
 fsi
```

TRI RAPIDE OU QUICKSORT

- On divise le tableau en deux (2) parties :
  - ***première partie*** : valeurs du tableau **inférieures** à la valeur de comparaison (appelée aussi valeur de partage),
  - ***deuxième partie*** : valeurs du tableau **supérieures** à la valeur de comparaison .
- **Pour effectuer le partage :**
  - on se déplace vers la droite du tableau tant qu'on reste inférieur à la valeur de comparaison et vers la gauche tant qu'on reste supérieur vers à la valeur de comparaison.
  - On permute les deux éléments mal placés, et on réitère ce processus jusqu'à ce que l'indice de parcours partant de la gauche dépasse celui partant de la droite.

Tri Rapide ou Quick Sort

```
Procedure tri_rapide (variable t : tableau d'entiers ; D, F : entiers)
Variables i, j, vp : entiers;
Debut
i := D ; j := F ; vp := t[(D+F) div 2] ;
faire
      Tant que t[i] < vp faire i := i+1 ;
      Tant que t[j] > vp faire j := j-1 ;
      Si i <= j alors
      debut
            permuter (t[i], t[j]) ;
            i :=i+1 ;
            j := j-1 ;
      fin
Tant que(i < j);
si D < j alors tri_rapide (t, D, j) ;
si i < F alors tri_rapide (t, i, F) ;
Fin
```