Chapitre 11 – Les algorithmes de tri

Un algorithme de tri permet d'organiser une collection d'objets selon un ordre déterminé. Cette collection doit être munie d'une relation d'ordre. Il est possible de définir un algorithme de tri indépendamment de la fonction d'ordre utilisée.

Classification des algorithmes de tri

- Permet de choisir l'algorithme le plus adapté au problème, tout en tenant compte des contraintes. Principales caractéristiques pour choisir :
  - *Complexité algorithmique :*

    Pour les tris les plus simples : $T(n) = O(n^2)$

    Pour les tris plus élaborés : $T(n) = O(n\log(n))$, complexité optimale.
  - *Ressources nécessaires*

    Un algorithme est **en place** s'il utilise un nombre limité de variables et s'il modifie directement la structure à trier (sans autre structure). Nécessite de l'utilisation d'une structure de donnée adaptée (par ex, un tableau).
  - *Caractère stable*

    Un algorithme est **stable** s'il garde l'ordre relatif des éléments (pour des éléments égaux). Ces algorithmes peuvent être retravaillés afin de les rendre stables, cependant cela peut être aux dépens de la rapidité et/ou de l'espace mémoire.

Tris par comparaison

- *Algorithmes lents*

  Algorithmes lents si > 20 élts et sont en $O(n^2)$. (Grande collections si > 30 000 élts).

  - Tri à bulles : $T(n) = O(n^2)$, en moyenne et dans le pire des cas, stable et en place ; amusant mais non efficace.
  - Tri par sélection : $T(n) = O(n^2)$, en moyenne et dans le pire des cas, pas stable si tri sur place ; rapide lorsque l'on a moins de 7 éléments.
  - Tri par insertion : $T(n) = O(n^2)$, en moyenne et dans le pire des cas, stable et en place. Le plus rapide et le plus utilisé pour moins de 15 éléments.

- *Algorithmes plus rapides (liste non exhaustive)*

  - Tri fusion (*merge sort*) : O(nlog n) en moyenne et dans le pire des cas, stable mais pas en place.
  - Tri rapide (*quick sort*) : O(nlog n) en moyenne, mais en $O(n^2)$ au pire cas, en place mais pas stable. Tri en place le plus rapide
  - Tri par tas (*heap sort*) : O(nlog n) en moyenne et dans le pire des cas, en place mais pas stable. Environ deux fois plus lent que le tri rapide. Intéressant à utiliser si les données à trier sont des cas quadratiques pour le tri rapide.

Tri fusion (mergesort)

- C'est un tri récursif qui est une application de « diviser pour régner ».
- Le principe récursif est le suivant :
  - on divise le tableau à trier en deux sous-tableaux de même longueur (±1),
  - on trie les deux sous-tableaux indépendamment,
  - et on fusionne les deux sous-tableaux triés.

Algorithme tri-fusion

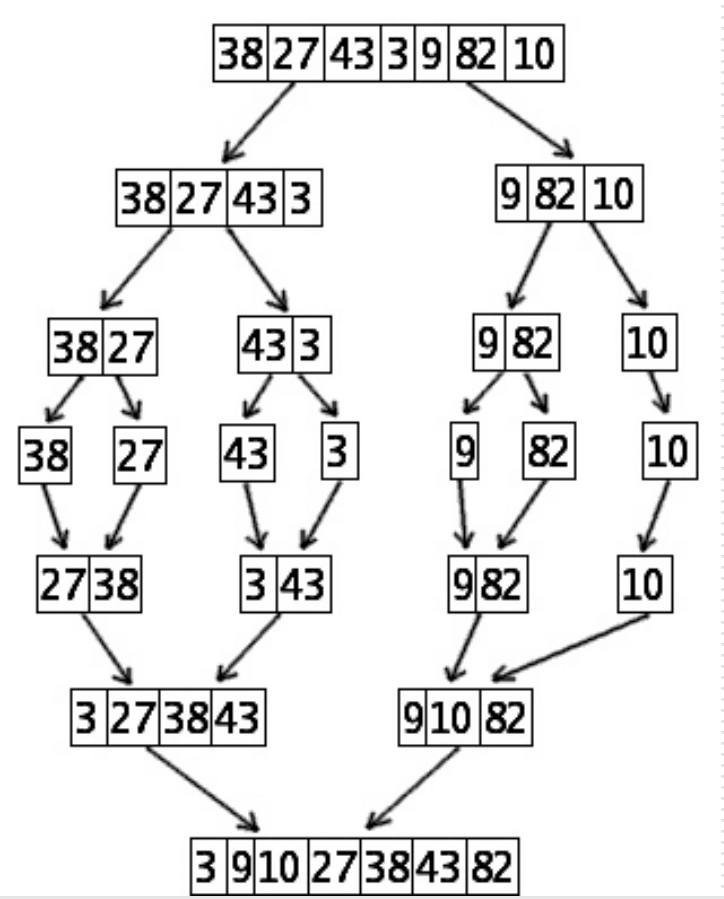
```
Const inf = 1
      sup = 100
Type tab = tableau [inf,sup] d'entier
Procédure tri-fusion (E/S t : tab ; E inf,sup : entier) {inf≤sup}
Var m : entier
Début
  si inf=sup
    alors écrire(''le tableau est trié'')
    sinon
      m←(inf+sup) div 2
      tri-fusion(t,inf,m)
      tri-fusion(t,m+1,sup)
      fusionner(t,inf,m,sup)
  finSi
Fin
```

Algorithme fusionner

(fusionne 2 parties triées de t dans ce même tableau en utilisant temp. Fusionner(t,inf,m,sup) fusionne les 2 parties de t qui vont de inf à m et de m+1 à sup (inf≤m≤sup)).

```
Procédure fusionner (E/S t : tab,
                     E d,m,f : entier)
Var i,j,k : entier
    temp : tab
Début
  i←d
  j←m+1
  Pour k←1 à f-d+1 inc +1 Faire
    Si i≤m et j≤f
       Alors
         Si t[i]≤t[j]
           Alors
             temp[k]←t[i]
             i←i+1
           Sinon
             temp[k]←t[j]
             j←j+1
    FinSi
       Sinon
         Si i≤m
           Alors
             temp[k]←t[i]
             i←i+1
           Sinon
             temp[k] ←t[j]
             j←j+1
         FinSi
    FinSi
  FinPour
  Pour k←1 à f-d+1 inc +1 Faire
     t[d+k-1]←temp[k]
  FinPour
Fin
```

Exemple


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 rapide (quicksort)

- C'est aussi un tri récursif qui est une application de « diviser pour régner ».

- Le principe récursif est le suivant :

  - on partitionne le tableau en deux sous-tableaux afin que les éléments du sous-tableau gauche (resp. droit) soient plus petits (resp. grands) ou égaux à un élément appelé pivot,
  - et on trie les deux sous-tableaux indépendamment.

Algorithme tri-rapide

```
Procédure tri-rapide(E/S t : tab, E inf,sup : entier)
Var p : entier
Début
  si inf<sup
    Alors
      partionner(t,inf,sup,p)
      tri-rapide(t,inf,p-1)
      tri-rapide(t,p+1,sup)
  Finsi
Fin
```

Algorithme partionner

on cherche le plus petit i tel que T[i] soit supérieur ou égal au pivot,
on cherche le plus grand j tel que T[j] soit inférieur ou égal au pivot,
si i<j, on échange T[i] et T[j] puis on recommence.

```
Procédure partitionner (E/S t : tab,
            E d,f : entier, S p : entier )
Var i,j,pivot : entier
Début
  pivot←t[d]
  i←d
  j←f
  TantQue i≤j Faire
    TantQue t[i]≤pivot et i≤j Faire
      i←i+1
    FinTantQue
    TantQue t[j]>pivot et i≤j Faire
      j←j-1
    FinTantQue
    Si i≤j alors
      échanger(t[i],t[j])
    FinSi
  FinTantQue
  p←j
  échanger(t[d],t[j])
Fin
```

Exemple

6 3 0 9 1 7 8 2 5 4

6 est le pivot !

6 3 0 9 1 7 8 2 5 4

échange de 9 et 4

6 3 0 4 1 7 8 2 5 9

échange de 7 et 5

6 3 0 4 1 5 8 2 7 9

échange de 8 et 2

6 3 0 4 1 5 2 8 7 9

échange de 6 et 2

2 3 0 4 1 5 6 8 7 9

Et on recommence sur les deux sous-tableaux …