Tris

A partir d'une liste de nombres dans le désordre, nous allons procéder à leur tri dans l'ordre croissant. Il existe plusieurs méthodes classiques, que nous allons programmer. Ce chapitre peut être vu comme une succession d'exercices d'entraînement, le problème du tri n'étant qu'un problème parmi une infinité.

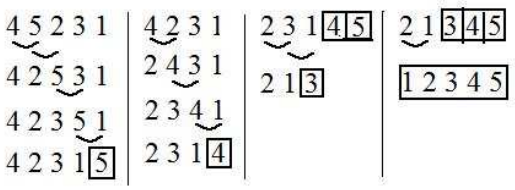
1. Le tri par sélection-échange

Il s'agit de la version officielle du tri que nous avons vu au chapitre précédent. On prend une liste de nombres, et on dégage l'élément minimal, d'où le nom de tri par sélection - du minimum. On procède alors à un échange avec celui qui est en première position. Désormais l'élément le plus petit est dans sa position définitive. Puis on recommence, en parcourant la liste à partir du deuxième élément et en dégageant le minimum. Par échange on place celui-ci en deuxième position, sa position définitive. Et on recommence à partir du troisième élément… D'où le programme :

```
remplir le tableau de nombres a[N]
for(i=0 ; i<N-1 ; i++)
{ mini=grand nombre ;
  for(j=i ; j<N ; j++) if (a[j]<mini)  { mini=a[j]; jmini=j; }
  aux=a[i]; a[i]=a[jmini]; a[jmini]=aux;
}
```

2. Le tri à bulles

Voici un exemple :



On compare le premier élément avec le deuxième, et s'ils ne sont pas dans le bon ordre, on les échange. Puis on compare le deuxième et le troisième, et éventuellement on les échange. Et ainsi de suite jusqu'au bout de la liste. A la fin de ce premier parcours, l'élément le plus grand se trouve à la fin, dans sa position définitive.

Tout se passe comme si l'on faisait coulisser un crochet sous deux éléments successifs, et l'on assiste au mouvement du plus grand élément qui est emporté jusqu'à la fin de la liste. Puis on recommence un deuxième parcours en faisant circuler le crochet du début de la liste jusqu'à l'avant-dernier élément, ce qui emporte le plus grand élément jusqu'à sa position finale, l'avant-dernière. Et ainsi de suite…

Le programme s'ensuit :

```
for(i=0 ; i< N-1 ; i++)   /* i est le numéro du parcours : parcours 0 puis parcours 1, ,..*/
for(j=0 ; j<N-1-i ; j++)  /* j est la position de l'élément de gauche du crochet */
if (a[j] > a[j+1])  { échanger a[j] et a[j+1] }
```

Ce procédé commence par placer l'élément le plus « lourd » au fond – à la fin de la liste, puis continue avec les éléments de plus en plus légers. Il s'agit en fait d'un tri par sédimentation. Si l'on veut un tri à bulles, il convient d'aller en sens inverse, de droite à gauche dans le parcours de la liste, et les éléments les plus légers arrivent en haut en premier –au début de la liste. Tout comme des bulles

qui montent dans un liquide. Mais avec la poussée d'Archimède les plus légères sont-elles aussi les plus petites ?

Amélioration, lorsque les éléments de la liste sont déjà presque triés

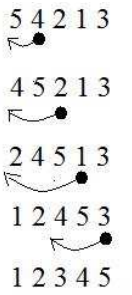
Prenons un exemple, avec au départ la liste 5 1 2 3 4. Après un premier parcours de la liste, le 5 prend sa position finale, et l'on arrivé à 1 2 3 4 5. Lorsque l'on relance le parcours sur 1 2 3 4, on s'aperçoit qu'il ne se produit aucun échange, ce qui indique que le tri est terminé. Par besoin de continuer les parcours. Il suffit d'ajouter ce test d'arrêt dans le programme.

```
for(i=0 ; i< N-1 ; i++)
 {  arret = OUI ;
    for(j=0 ;j<N-1-i ; j++)     if (a[j] >a[j+1])  { échanger a[j] et a[j+1] ; arret=NON ;}
    if  (arret= = OUI)  break ;
 }
```

Ainsi, au cas où l'on tombe sur une liste initiale qui est déjà presque triée, le tri à bulles devient le tri le plus rapide qui existe.

3. Tri par insertion

Ce procédé s'inspire de celui utilisé lorsque l'on classe des cartes que l'on a en main. Voici un exemple :



On prend le deuxième élément, ici le 5 en position 1, et on l'insère à gauche à la bonne place, d'où 4 5. Puis on prend le troisième élément, ici le 2 en position 2, et on l'insère à sa gauche à la bonne place, d'où 2 4 5. Et ainsi de suite jusqu'à au dernier élément. Tout au long du processus, à chaque étape la liste est partagée en deux parties : la partie de gauche est triée, et l'on prend le premier élément de la partie de droite, que l'on insère à gauche dans la bonne position, ce qui augmente d'une unité la longueur de la partie triée de gauche.

Programme :

```
for(i=1 ; i<N ; i++)
  { j=i;
    while (j>=1 && a[j]<a[j-1]) { échanger a[j]et a[j-1];  j--;}
  }
```

A remarquer le blocage *j*>=1. Si on ne le fait pas, la boucle *while* envoie *j* dans le négatif. Erreur fatale !

4. Performance de ce type de tri

Tous les tris précédents font intervenir une double boucle, avec des comparaisons entre deux nombres de la liste à chaque fois, par exemple entre $a[i]$ et $a[j]$ dans le tri par sélection-échange vu au chapitre précédent. Si N désigne le nombre de données, le nombre de comparaisons est de l'ordre de $N \times N = N^2$. Si l'on considère que le temps mis par le tri est la somme des temps élémentaires correspondant aux comparaisons (avec échange éventuel), cela donne un temps de l'ordre de N^2. Plus précisément, si l'on prend le tri à bulles, il y a N-1 comparaisons lors du parcours i=0 de la liste, puis N-2 lors du parcours i=1, etc., soit au total $(N\text{-}1) + (N\text{-}2) + \ldots + 2 + 1 = N(N\text{-}1)/2$, de l'ordre de $N^2/2$ pour N grand.

On dit que le comportement asymptotique (pour N très grand) de ces tris est de l'ordre de N^2 (à un facteur près). Cela veut essentiellement dire que si le nombre de données double, le temps sera quatre fois plus long.

On verra plus tard d'autres tris qui ont une performance de l'ordre de $N \log N$. Sachant que $N / \log N$ devient infiniment grand lorsque N tend vers l'infini, on peut affirmer que les tris en $N \log N$ sont beaucoup plus performants que ceux en N^2 que nous venons de voir. Mais concrètement, tant que le nombre de données ne dépasse pas quelques milliers, la différence entre les tris n'est pas sensible. C'est seulement au-delà qu'elle devient notable, et elle peut prendre des proportions gigantesques lorsque le nombre des données augmente.

Dans certains cas particuliers, certains types de tri sont mieux adaptés, comme dans l'exemple suivant. Nous verrons aussi d'autres tris dans les chapitres suivants.

5. Tri de *N* nombres tous compris entre 1 et *K*, avec *K* relativement petit par rapport à *N*

Prenons un exemple, avec le tableau $a[N+1]$, où $N = 8$ et $K = 6$, les cases étant indexées à partir de 1.

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

Première étape : Fabriquer un tableau $C[]$ de longueur k, où $C[i]$ contient le nombre de fois où le nombre i est présent dans le tableau $a[]$, soit :

```
for(i=1 ; i<=K ; i++) C[i]=0 ;
for(i=1 ; i<=N ; i++) C[a[i]]++;
```

1	2	3	4	5	6
2	0	2	3	0	1

Deuxième étape : Transformer le tableau $C[]$ de façon que $C[i]$ devienne le nombre d'éléments de $a[]$ inférieurs ou égal à i, soit :

1	2	3	4	5	6
2	2	4	7	7	8

Remarquons que le contenu de la case $C[1]$ ne change pas. Pour la suivante, on lui ajoute $C[1]$, et ainsi de suite en ajoutant dans chaque case le cumul déjà obtenu dans la case précédente.

```
for(i=2 ; i<=N ; i++) C[i]+=C[i-1];
```

Troisième étape : On va remplir un tableau $B[]$ qui contiendra à la fin la liste triée. Pour cela, on parcourt le tableau $a[]$ à partir de la fin, avec i de 8 à 1. A chaque fois, on place le contenu de la case $a[i]$ dans le tableau B, par exemple pour i=8, on prend $a[8]$=4 et on le met dans la case 7, puis on diminue le nombre $c[4]$ de 1. Et on continue de même comme indiqué ci-dessous.

	1	2	3	4	5	6	7	8								
i=8							4		B	2	2	4	6	7	8	C
i=7		1					4			1	2	4	6	7	8	
i=6		1				4	4			1	2	4	5	7	8	

……..

```
for(i=8 ; i>=1 ; i--)    {B[C[a[i]]] =a[i]; C[a[i]]--;}
```