Les tris

Les tris sont, en informatique, une opération essentielle dont les applications sont extrêmement nombreuses. Ils posent le double problème que l'on retrouve à chaque fois que l'on fait un programme informatique : le temps de calcul et l'occupation mémoire. Ces deux aspects ne sont pas étudiés ici : on se contente de voir différentes méthodes de tri

On se propose dans un premier temps de trier des nombres par ordre croissant.
Dans une amélioration possible, on pourra améliorer les algorithmes écrits :

a) en modifiant la relation d'ordre (tri par ordre croissant, par ordre décroissant)
b) en modifiant le type de données (tri de lettres, puis tri de mots par exemple)

Par ailleurs, la structure de données est la liste (comme toujours en Scheme), les algorithmes suivant doivent être adaptés quand on prend comme structure de donnée un tableau (à voir plus tard)

I Tri par insertion

Principe : C'est le plus simple des tris : il consiste à mettre un élément « à sa place » dans une liste déjà triée.

Programmation :

a) écrire une fonction *insere* qui prend en argument un nombre et une liste de nombres ***déjà triée*** et ressort la liste avec le nombre « à sa bonne place » dans la liste.
b) écrire la fonction *tri_ins* de tri par insertion
c) essayer d'écrire une version terminale de tri_insertion qu'on appellera *tri_ins_it*
d) en complément, essayer de dérécursiver la fonction *insère* en *insere_it*

II Tri par sélection

Principe :

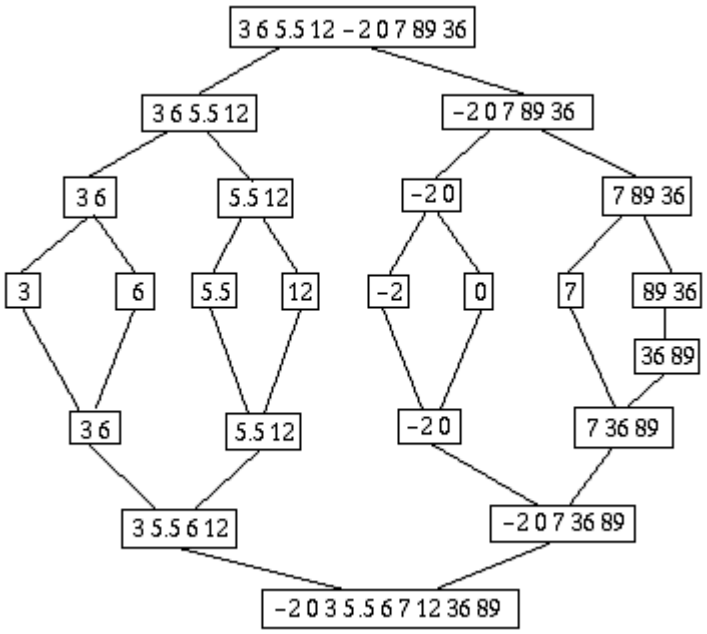
1) on recherche le minimum de la liste
2) on le retire de la liste
3) on le met en tête
4) on relance sur la queue

Programmation :

a) ecrire la fonction *minimum* qui recherche le minimum d'une liste de nombres
b) écrire la fonction *enleve* qui retire le minimum de la liste
c) écrire la fonction *en_tete* qui met un nombre en tête d'une liste
d) écrire une fonction *met_en_tete* qui fait simultanément b) et c)
e) écrire la fonction *tri_selection*

III Tri par fusion (ou merge sort en anglais)

Principe : Le tri par fusion consiste à diviser la liste en deux sous-listes de même longueur, récursivement, puis à recombiner les sous-listes :



Programmation :.

a) écrire la fonction ***fusion*** qui prend en argument deux listes ***déjà triées*** et forme une seule liste triée
b) écrire la fonction ***sous_liste*** qui prend comme argument une liste et deux nombres i et j (i <j) et qui retourne la sous liste allant du i_ème élément au j_éme ; le premier élément est l'élément 1.
c) Ecrire la fonction ***eclate*** qui prend une liste et retourne une liste dont le car est la première moitié de la liste et le cdr l'autre moitié
d) écrire la fonction ***tri_fusion*** de tri par fusion en utilisant soit *sous_liste* soit *eclate*

IV Tri rapide (ou quick sort en anglais)

Principe : Ce tri consiste à « diviser pour régner »

1) on prend dans la liste un élément appelé « pivot »
2) on réorganise la liste
   a) en plaçant à gauche du pivot, tous les éléments plus petits
   b) en plaçant à gauche du pivot, tous les éléments plus gransts

   …. Cette opération s'appelle le « partionnement » (partition opération in english)
3) … et on recommence récursivement l'opération sur les deux sous listes correspondantes

Programmation :

a) écrire la fonction ***divise*** qui prend en argument une liste, un nombre (le pivot) et deux listes qui comporteront respectivement les éléments inférieurs au pivot et supérieurs ou égal au pivot

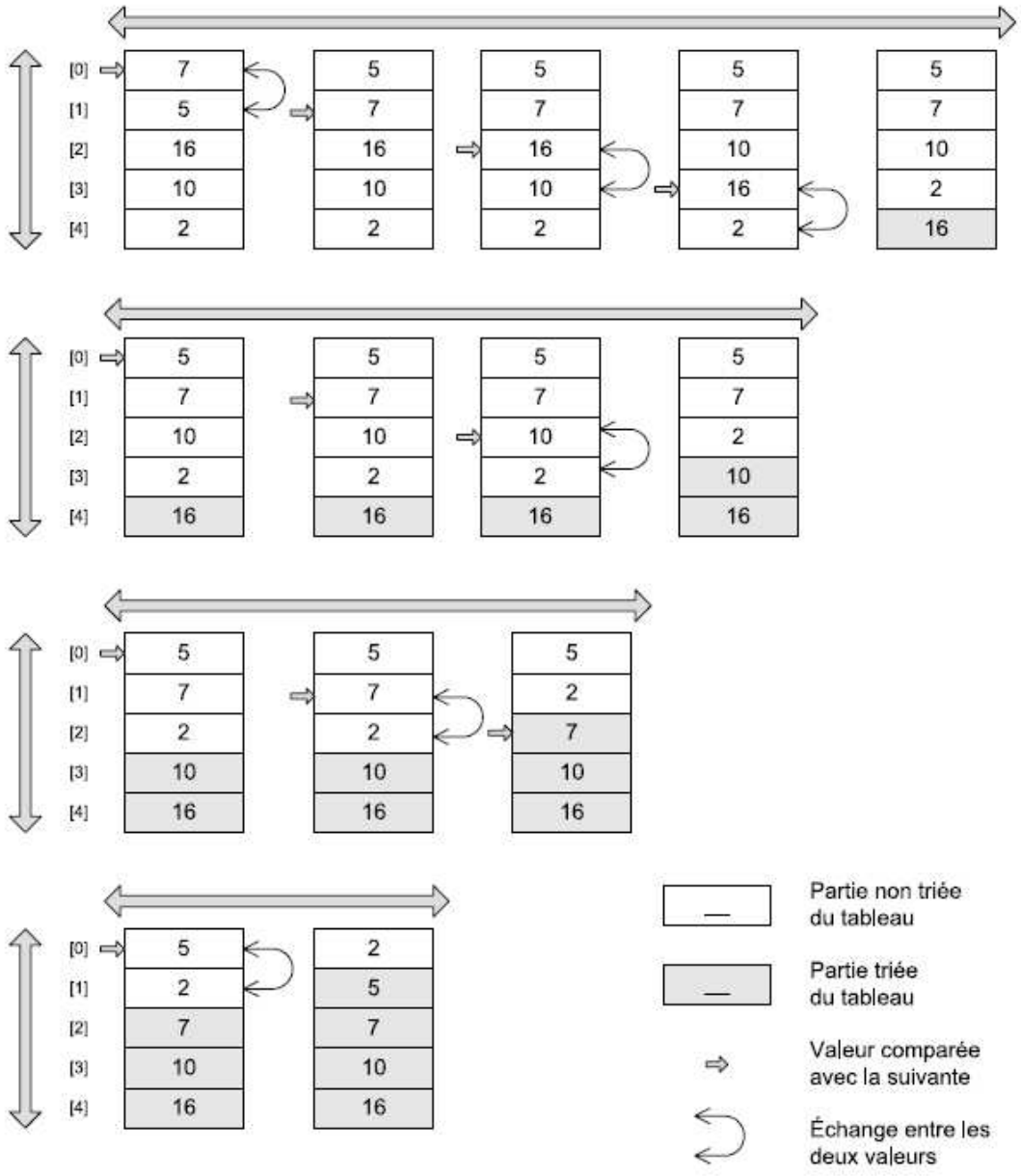
b) écrire la fonction ***tri_rapide*** en utilisant la fonction divise

Rem : on prendra comme pivot le premier élément de la liste à trier

V Tri à bulle (ou bubble sort en anglais)

Principe : L'algorithme parcourt la liste pour comparer les éléments deux à deux afin de faire « descendre » les valeurs les plus lourdes en bas de la liste *(si on dessine la liste verticalement avec la fin de la liste en position basse).*

Les éléments les plus légers (un peu comme les bulles d'air dans l'eau) vont remonter au début du tableau (à la surface). Cette méthode présente l'avantage d'être très rapide si le tableau est presque trié (sauf quelques éléments), mais elle sera lente si les éléments du tableau ne sont pas du tout triés.



Soit l'exécution sur l'exemple suivant :

```
|(tri_bulle (2 10 16 5 7))
| (tri_bulle (7 5 16 10))
| |(tri_bulle (10 16 7))
| | (tri_bulle (10 16))
| | |(tri_bulle (16))
| | | (tri_bulle ())
| | | ()
| | |(16)
| | (10 16)
| |(7 10 16)
| (5 7 10 16)
|(2 5 7 10 16)
```

Programmation :

a) écrire la fonction ***met_min_en queue*** qui prend en argument une liste et retourne la liste avec le nombre le plus petit en queue (sans utiliser la fonction minimum analogue au tri par sélection)

b) écrire la fonction ***tri_bulle***

Utilitaires pour tester vos fonctions de tri :

(random_liste n n_max)

qui retourne une liste de n entiers déterminés de façon aléatoire dans l'intervalle [0 , n_max[

```
(define (random_liste n n_max)
   (if (zero? n)
       ()
       (cons (random n_max) (random_liste (sub1 n)n_max))))

(define (test-time tri n)
    ; indique le temps d'éxécution de tri (préciser le type de tri)
    ; pour le tri de 100, 200, 300,.... n*100  entiers inférieurs à
1000
   (define (bosse n i)
   (cond ((not (zero? n))
         (newline)
         (display (* 100 i))
         (newline)
         (time (tri (random_liste (* i 100) 1000)))
         (bosse (sub1 n ) (add1 i)))
        (else false)))
   (bosse n 1))

exemple d'appel :
(test-time  tri_bulle 10)
```