.1 Tris efficaces (2)

1. On considère le tableau $[13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21]$. Donner les traces d'exécution du tri de ce tableau par l'algorithme du tri rapide.

2. **Tri fusion :** cette méthode de tri est en quelque sorte l'inverse du tri rapide. L'idée est de partager la liste à trier en deux sous-listes que l'on trie, puis on "interclasse" ces deux sous-listes. Un *interclassement* consiste à construire, à partir de deux listes triées, une liste contenant la réunion des éléments des deux listes d'origine, triée, en conservant les répétitions.
   - Donner une version (récursive ou non) en pseudo-code de cet algorithme
   - Quel inconvénient voyez-vous pour cet algorithme ?

3. On considère deux tableaux A et B de n éléments chacun. On dit que A est *contenu* dans B si tous les éléments de A figurent au moins une fois dans B.
   - Écrire une fonction qui, sans trier les tableaux, dit si A est contenu dans B. Quelle est la complexité de l'algorithme ?
   - Améliore-t-on la complexité si on trie l'un des deux tableaux et si oui, préciser lequel. Améliore-t-on à nouveau la complexité si l'on trie les deux tableaux ?

4.

.2 Listes

1. On suppose les fonctions suivantes définies pour la manipulation d'une liste de réels (`rliste`) (`l` est une rliste, `p` est une position, `e` est un réel) :
   ```
   longueur(l)
   inserer(l,e,p)
   supprimer(l,p)
   element(l,p)
   ```

   (a) Ecrire une fonction qui prend une rliste en paramètre, et trie la liste. Deux implémentations seront réalisées : l'une qui trie la liste "sur place", l'autre qui crée une nouvelle liste triée. Comparer les coûts dans les deux cas.

   (b) Ecrire une fonction qui supprime tous les doublets de la liste (sans la réordonner).

2. Pour résoudre le problème de la borne statique du tableau, ajouter à la fonction d'insertion un contrôle sur la taille, qui réalloue un tableau plus grand lorsque la taille maximale est atteinte.

3. "Padding". Modifier la fonction d'insertion de telle sorte que si on demande d'insérer un élément à une position p plus grande que la taille actuelle, la liste est complétée jusqu'à la position p avec une valeur "neutre", par exemple 0.

4. Pour trier une liste, on a souvent besoin d'une fonction d'échange qui intervertit les valeurs de deux positions dans la liste. Proposer deux versions de fonction d'échange : l'une écrite au moyen des primitives déjà écrites, l'autre écrite directement en accédant à la structure de données. Discuter la complexité.

rliste.h

```
#define RLISTE_MAX 100
struct s_rliste {
  float tab[RLISTE_MAX] ;
  int l ;
} ;
typedef struct s_rliste rliste ;
```

rliste.c

```
#include "rliste.h"

int longueur (rliste l)
{
  return l.l ;
}

float element(rliste l, int p)
{
  // Hypothèse : p in [1,longueur(l)]
  return l.tab[p-1] ;
}

void inserer(rliste *l, int p, float v)
{
  // Hypothèses : p in [1,longueur(l)+1]
  //              longueur(l) < RLISTE_MAX
  int i ;
  for (i=l->l-1 ; i>=p-1 ; i--)
    l->tab[i+1] = l->tab[i] ;
  l->tab[p-1] = v ;
  l->l++ ;
}

void supprimer(rliste *l, int p)
{
  // Hypothèse : p in [1,longueur(l)]
  int i ;
  for (i=p-1 ; i < l->l-1 ; i++)
    l->tab[i] = l->tab[i+1] ;
  l->l-- ;
}
```

.3 Correction

1. au tableau
2. Un exemplexs

```
À l'état initial on a les éléments un par un, on les fusionne deux à deux:

([6] [1]) ([2] [5]) ([4] [7]) [3]

On obtient:

([1;6] [2;5]) ([4;7] [3]) que l'on fusionne deux à deux
à nouveau et ainsi de suite:

([1;2;5;6] [3;4;7])

[1;2;3;4;5;6;7]
```

Deux codes (dont un récursif buggé) :

```
typedef int tab_entiers[MAX];

// Fusion des listes t(de1..vers1) et t(de2..vers2)
dans tmp(posInTmp..posInTmp+count-1)
void fusion(tab_entiers t, tab_entiers tmp, int de1, int vers1,
  int de2, int vers2, int count, int posInTmp)
{
     int i;
     for(i = 0 ; i < count ; i++)
     {
       if (de2 > vers2) // Si fin de la liste 2, on prend dans liste 1
          tmp[posInTmp++] = t[de1++];
       else if (de1 > vers1)   // Idem si fin de la liste 1
          tmp[posInTmp++] = t[de2++];
       else if (t[de1] <= t[de2])   // Enfin, sinon, on compare
          tmp[posInTmp++] = t[de1++];
       else
          tmp[posInTmp++] = t[de2++];
      }
}

// Tri de tout le tableau t par fusions successives
void trifusion(tab_entiers t)
{
      tab_entiers tmp;
      int sortLength = 1, de1, de2, de3, i;
      while(sortLength < MAX)
      {
              de1 = 0;
```

```
              while(de1 < MAX)
              {
               de2 = de1 + sortLength;
               de3 = de2 + sortLength;
               if(de2>MAX) de2 = MAX;
               if(de3>MAX) de3 = MAX
               fusion(t, tmp, de1, de2-1, de2, de3-1, de3-de1, de1);
               de1 = de3;
              }
              for(i = 0 ; i < MAX ; i++) t[i] = tmp[i];
              sortLength *= 2;
       }
 }


/*************************************************************/
FROIDEVAUX: buggée?
/*************************************************************/

fonction interClassement(t,s,i,j,k){
  //i:=1;j:=1;k:=1; //utile si on ne précise pas les bornes
  while(i<=n) and (j<=m) {
    if t[i] < s[j]{
      r[k]:=t[i]; i++;
    }
    else{
      r[k]:=s[j]; j++;
    }
    k++;
  }
  if(i<=n){ //on recopie la fin de t
   for j:=i to n : r[k]:=t[j]; k++;
  }
  else{ //on recopie la fin de s
   for i:=j to m : r[k]:=s[i]; k++;
  }
}

/*
triFusion trie les éléments tab[dep,i],...,tab[dep,j] et range
le résultat du tri dans les places t[arr,i],...,t[arr,j].
*/
function triFusion(tab[1..2,1..n],i,j,dep,arr){
  int a1,a2:1..2;
  if(arr==2): a1:=1; a2:=2; else  a1:=2; a2:=1;
  if (i<j){
    k:=(i+j)/2;
    triFusion(tab,i,k,dep,a1);
```

```
    triFusion(tab,k+1,j,dep,a2);
    if(a1<>a2){
     for m:=k+1 to j: tab[a1,m]:=tab[2,m];
    }
    interClassement(t,i,j,k,a2); //on range le résultat dans a2
    if(a1==1): dep:=2 else dep:=1;
  }
  else{
    if(i==j): arr=a1;
  }
}

Appel:
triFusion(tab,1,n,1,arr)
```

3. – au tableau, très facile
   – On trie le tableau B ce qui améliore la complexité en meilleur/moyen cas, au pire cas on reste à complexité égale. Trier A ne sert à rien : on doit vérifier chacun des éléments de toute façon (éventuellement ça permet de sauter qques doublons sur le dernier élément avec un test sur le dernier élément).