# [Math-L312] TP 6 : Programmation récursive et tableaux

Adrien Semin
adrien.semin@math.u-psud.fr

## 1 Remarque pour ce TP

Étant donné que le mercredi 11 novembre est férié, les spécificités pour ce TP sont :
- cette feuille de TP servira également pour le TP7,
- le TP7 aura lieu (je le rapelle) le lundi 9 novembre de 15h30 à 17h15 pour le groupe K, et le jeudi 12 novembre de 13h30 à 15h15 pour le groupe H,
- le TP sera à rendre vendredi 13 novembre au plus tard (pour les deux groupes).

## 2 Travail à faire

`tri_direct.c` ***À Rendre!*** Implémenter une fonction `tri_direct` qui prend en entrée un pointeur sur un tableau dynamique de type float et un entier représentant la taille du tableau, et qui trie le tableau de la manière suivante :
- on recherche le plus grand élément du tableau, et on met cet élément en dernière position dans le tableau,
- on recherche le deuxième plus grand élément du tableau, et on met cet élement en avant-dernière position dans le tableau,
- on recherche le troisième plus grand élément du tableau, et on met cet élement en avant-avant-dernière position dans le tableau,
- ...

Tester cette fonction sur un tableau initialisé avec la fonction `fill_array`, contenue dans l'en-tête *non standarde* `fill_array.h`, téléchargeable à l'adresse `/home/doc/semin/TP/h/fill_array.h`. L'appel de cet en-tête se fait de la même manière que pour un en-tête standard. Il faut rajouter l'option `-I` suivi du nom du répertoire où se trouve le fichier en question. Exemple si le fichier `fill_array.h` se trouve dans le même répertoire que le fichier `tri_direct.c` :

```
> gcc tri_direct.c -I. -o tri_direct_compile
```

Typiquement, on devra avoir dans le programme

```
/* ... Les instructions avant le remplissage et le tri */
float* tableau; int taille;
/* ... Les instructions avant le remplissage et le tri */
fill_array(tableau,taille);
/* ... Les instructions entre le remplissage et le tri */
tri_direct(tableau,taille);
/* ... Les instructions après le remplissage et le tri */
```

Afficher également le tableau avant et après le tri.

`tri_bulle.c` ***À Rendre!*** Reprendre l'exercice précédent, mais en implémentant le tri à bulle :
1. on initialise $N$ à la taille du tableau,
2. on parcourt pour $i$ allant de 1 à $N-1$,
3. si l'élément $i$ du tableau est plus grand que l'élément $i+1$, on inverse ces deux éléments,
4. on reprend le point 2. avec $N$ diminué de 1 si au moins une inversion a été faite en 3.

`tri_fusion.c` ***À Rendre!*** Reprendre l'exercice précédent, mais en implémentant le tri-fusion : pour trier un tableau de taille $N$, on
1. trie la première moitié du tableau,
2. trie la seconde moitié du tableau,

3. recolle les deux moitiés triées du tableau.

`matrix.c` | ∗ ***À Rendre!*** ∗ | On demande un programme qui multiplie entre elles 2 matrices de coefficients aléatoires entiers compris entre 0 et 10 dont les tailles sont données par les arguments passés au programme. Par exemple, `matrix 3 5 5 7` génèrera deux matrices aléatoires de tailles respectives 3x5 et 5x7, et les multipliera entre elles. On utilisera la structure suivante

```
typedef struct {
  int N;
  int M;
  int **data;
} matrice;
```

et on définira les fonctions suivantes :

- `void creatematrice(int width, int height, matrice* mat)` : cette fonction initialise une matrice de taille `width * height` et alloue la mémoire nécessaire pour les coefficients,
- `void fillmatrice(matrice* mat)` : cette fonction remplit les coefficients d'une matrice initialisée de nombres entiers aléatoires,
- `void destroymatrice(matrice* mat)` : cette fonction détruit la mémoire précédemment allouée par une matrice,
- `void affmatrice(matrice mat)` : cette fonction affiche la matrice `mat`,
- `void multimatrice(matrice mat1, matrice mat2, matrice* matreturn)` : cette fonction multiplie entre elles les matrices `mat1` et `mat2` et stocke le résultat dans la matrice `matreturn`. On vérifiera que le nombre de colonnes de `mat1` est égal au nombre de lignes de `mat2`.