- Fiche de TP3 : Diviser pour régner -

Une trame de code en C(++) est disponible pour les exercices des deux premières parties.

- Tri fusion -

Le but de cette partie est d'implémenter un tri fusion, comme étudié en cours. Le fichier *FonctionsTableau.cc* contient des fonctions de bases d'intialisation et de gestion des tableaux.
Le fichier *tp3ex1.cc* contient les fonctions à compléter. Il faut seulement compiler ce fichier-là (avec **g++**) en ayant inclus *FonctionsTableau.cc* dans le même répertoire.

- Exercice 1 - Fusion ! -

Compléter la fonction `trifusion(int n, int T[])` en lui faisant tout d'abord uniquement remplir les tableaux `T1` et `T2`.
Compléter ensuite la fonction `fusion(int n1,int n2,int T1[],int T2[],int T[])` qui fusionne les tableaux `T1` et `T2` de tailles respectives `n1` et `n2` dans le tableau `T`.
Compléter `trifusion` en ajoutant les appels récursifs.
À chaque étape, testervos programmes sur l'exemple fixe proposé avant de les faire tourner sur des tableaux de tailles plus importantes.

- Exercice 2 - Comparaison avec un tri à bulles -

Implémenter la fonction `void tribulles(int n, int T[])`, qui trie le tableau `T` à l'aide d'un tri à bulles. On pourra faire appel à la fonction *swap(a,b)* de la libraire standard qui échange le contenu des variables a et b en temps $O(1)$.
Comparer les temps mis par le tri fusion et le tri par tas. Augmenter la taille des tableaux à traiter pour observer une différence importante.

- Exercice 3 - Pour aller plus loin : tri fusion presque sur place -

Programmer une nouvelle version de l'algorithme TRI-FUSION qui permet de trier sans garder en mémoire des tableaux (à part le tableau de départ) à chaque appel récursif. Pour cela il faut préciser aux différentes fonctions sur quel intervalle du tableau T elle doivent travailler. Ainsi, par exemple la signature de la fonction de tri pourra être : `void trifusion(int n, int a, int b, int T[])` signifiant que lors de l'appel de cette fonction, on cherchera à trier les éléments rangés entre les indice a et b (compris) du tableau T. L'appel intial sera `trifusion(n,0,n-1,T);`. Il faudra modifier de même la fonction `fusion`, en s'autorisant ici à créer temporairement un tableau intermédiaire permettant d'effectuer la fusion. Ce tableau devant être libérer à la fin de la fonction `fusion`.

- Calcul de rang -

Le but de cette partie est d'implémenter un calcul de rang linéaire, comme étudié en cours. Le fichier *FonctionsTableau.cc* contient des fonctions de bases d'intialisation et de gestion des tableaux.
Le fichier *tp3ex2.cc* contient les fonctions à compléter. Il faut seulement compiler ce fichier-là (avec **g++**) en ayant inclus *FonctionsTableau.cc* dans le même répertoire.

- Exercice 4 - Calcul de rang -

La tâche principale est l'implémentation de la fonction `int rang(int k, int n, int T[])` permettant de trouver le `k` ième rang du tableau `T` de taille `n`.

Compléter le calcul de l'indice du pivot `p` selon le choix aléatoire vu en cours. Compléter ensuite les différents cas à traiter. On prêtera attention aux remplissages des tableaux `Ti` et `Ts`.
Tester vos programmes sur l'exemple fixe proposé avant de les faire tourner sur des tableaux de tailles plus importantes.

- Exercice 5 - Nombre de tirages -

Lors de la recherche d'un calcul de rang, déterminer le nombre moyen de tirages de pivot aléatoire par appel à la fonction `rang`. Pour cela, on se servira des variables statiques `ntirage` et `npivot`. On incrémentera `ntirage` à chaque nouveau tirage aléatoire de pivot et `npivot` à chaque pivot valide trouvé (c-à-d à chaque entrée dans la fonction `rang` sur un tableau de taille au moins 2).

- Exercice 6 - Pour aller plus loin : comparaison entre différents calculs de rang -

Écrire une fonction `void rang-par-tri(int k, int n, int T[])` qui calcule le k ième rang du tableau `T` en le triant (avec `tri-fusion` par exemple) et affiche sa valeur.
En s'inspirant de la méthode utilisée pour TRI-FUSION, comparer les temps d'exécution de `rang` et de `rang-par-tri`.

- Multiplication de polynômes -

- Exercice 7 - Encore plus loin : algorithmes classique et de Karatsuba -

Implémenter les deux algorithmes de multiplication de polynômes vu en cours, l'algorithme classique et l'algorithme de Karatsuba.