
**École Centrale de Lille**

# AAP - Algorithmique avancée et programmation
# S5

BARBOZA DE DEUS FONSECA Fernando

## Compte Rendu - TEA Séance 3

Responsable :

Monsieur THOMAS BOURDEAUD HUY

**Villeneuve-d'Ascq**

**Novembre 2021**

# Table des matières



# Table des figures

# 1 Introduction

L'objectif de ce TEA est de travailler avec des algorithmes de tri comment le Tri Fusion et le Tri Rapide. Enfin, une analyse de performance de chaque algorithme pour obtenir les avantages de chaque méthode.

La activité est divisé en 3 parties :

— Tri Fusion : Implémenter tri fusion d'un tableau ;
— Tri Rapide : Implémenter tri rapide d'un tableau ;
— Tri Fusion de listes : Implémenter tri fusion d'une liste ;

Le dossier S3_TEA est organisé comme suit :

— Fichier ***README*** avec des informations sur makefile e la structure ;
— Fichier ***makefile*** pour toutes les parties ;
— Dossier ***include*** avec les fichiers utilisés dans tous les exercices ;
— Dossiers ***Tri**** avec la ***main.c*** et les fichiers utilisés dans chaque exercice ;

# 2 Tri Fusion

## 2.1 Développement

Cette partie est composée de deux fonction :

— ***fusionner*** : Responsable pour comparé les élément de 2 tableau et les ordonnée ;
— ***fusionSort*** : Fonction qui applique le algorithme de Tri Fusion en utilisant lui même et la fonction fusionner ;

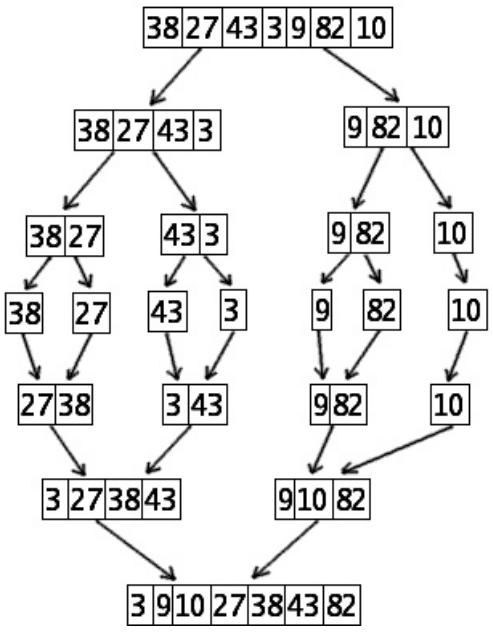


FIGURE 1 – Représentation graphique de la Tri Fusion

### 2.1.1 Code *fusion_sort.c*

```c
#include <string.h>
#include <stdio.h>
#include "test_utils.h"

#include "../include/traces.h"

// TODO : placer les compteurs aux endroits appropri s :
// stats.nbOperations ++;
// stats.nbComparisons ++;

void fusionner(T_elt t [], int d, int m, int f);


T_data fusionSort(T_data d, int n) {


    stats.nbComparisons++;
    if(n==1) return d;

    int m = n/2;
    T_elt * A = d.pElt;


    T_data leftArray = fusionSort(d, m);
    T_data rightArray = fusionSort(genData(0,&(A[m])), (n-m));

    fusionner(A, 0, m-1, n-1);

    return genData(0, A);


}

void fusionner(T_elt t [], int d, int m, int f) {
    T_elt aux[f - d + 1]; // !! Allocation dynamique sur la pile (standard C99)
    int i, j, k;


    memcpy(aux, &t[d], (f - d + 1) * sizeof(T_elt));    // Copie des donn es    fusio
    stats.nbOperations += (f - d + 1);

    i = 0; j = m - d + 1; k = 0;
    while (i <= m - d && j <= f - d) {
        stats.nbComparisons+=2;
        stats.nbOperations++;
        if (aux[i] <= aux[j])   {
            t[d + k++] = aux[i++];  // aux[i] est plus petit : on le place dans t
        }
        else {
            t[d + k++] = aux[j++];  // aux[j] est plus petit : on le place dans t
        }
    }
    stats.nbOperations += (m - d - i > 0) ? m - d - i : 0;
    for (; i <= m - d; t[d + k++] = aux[i++]); // le reste du tableau gauche
```

```
    stats.nbOperations += (m - d - i > 0) ? m - d - i : 0;
    for (; j <= f - d; t[d + k++] = aux[j++]); // le reste du tableau droit
}
```

## 2.2 Résultat

```
fernandobdf23@fernandofonseca-VM:~/AAP/S3_TEA$ ./TriFusion/TriFusion.exe
+----------------------------+
| TRI FUSION                 |
+-----------+----------------+---------------+---------------+---------------+
|   Taille  |           Mode |    Nb compar. |      Nb opér. |    Duree (ms) |
+-----------+----------------+---------------+---------------+---------------+
|     2500  |        ordonne |         32303 |         42056 |             0 |
|     5000  |        ordonne |         69607 |         91612 |             0 |
|     7500  |        ordonne |        109751 |        144184 |             0 |
|    10000  |        ordonne |        149215 |        198224 |             1 |
|    12500  |        ordonne |        191255 |        254244 |             1 |
|    15000  |        ordonne |        234503 |        310868 |             1 |
|    17500  |        ordonne |        277599 |        368532 |             1 |
|    20000  |        ordonne |        318431 |        426448 |             1 |
|    22500  |        ordonne |        361839 |        485652 |             2 |
|    25000  |        ordonne |        407511 |        545988 |             2 |
+-----------+----------------+---------------+---------------+---------------+


+----------------------------+
| TRI FUSION                 |
+-----------+----------------+---------------+---------------+---------------+
|   Taille  |           Mode |    Nb compar. |      Nb opér. |    Duree (ms) |
+-----------+----------------+---------------+---------------+---------------+
|     2500  |      aleatoire |         55167 |         53735 |             0 |
|     5000  |      aleatoire |        120501 |        117544 |             1 |
|     7500  |      aleatoire |        189177 |        184725 |             1 |
|    10000  |      aleatoire |        260891 |        255064 |             1 |
|    12500  |      aleatoire |        333979 |        326947 |             1 |
|    15000  |      aleatoire |        408461 |        399534 |             2 |
|    17500  |      aleatoire |        484779 |        473969 |             2 |
|    20000  |      aleatoire |        561707 |        550044 |             3 |
|    22500  |      aleatoire |        639931 |        626793 |             3 |
|    25000  |      aleatoire |        718395 |        704075 |             3 |
+-----------+----------------+---------------+---------------+---------------+


+----------------------------+
| TRI FUSION                 |
+-----------+----------------+---------------+---------------+---------------+
|   Taille  |           Mode |    Nb compar. |      Nb opér. |    Duree (ms) |
+-----------+----------------+---------------+---------------+---------------+
|     2500  |        inverse |         34503 |         54309 |             0 |
|     5000  |        inverse |         74007 |        118617 |             0 |
|     7500  |        inverse |        113863 |        186117 |             0 |
|    10000  |        inverse |        158015 |        257233 |             1 |
|    12500  |        inverse |        200975 |        329733 |             1 |
|    15000  |        inverse |        242727 |        402233 |             1 |
|    17500  |        inverse |        286863 |        476965 |             1 |
|    20000  |        inverse |        336031 |        554465 |             1 |
|    22500  |        inverse |        382623 |        631965 |             1 |
|    25000  |        inverse |        426951 |        709465 |             2 |
+-----------+----------------+---------------+---------------+---------------+
```

FIGURE 2 – Résultat Tri Fusion - Comparaison

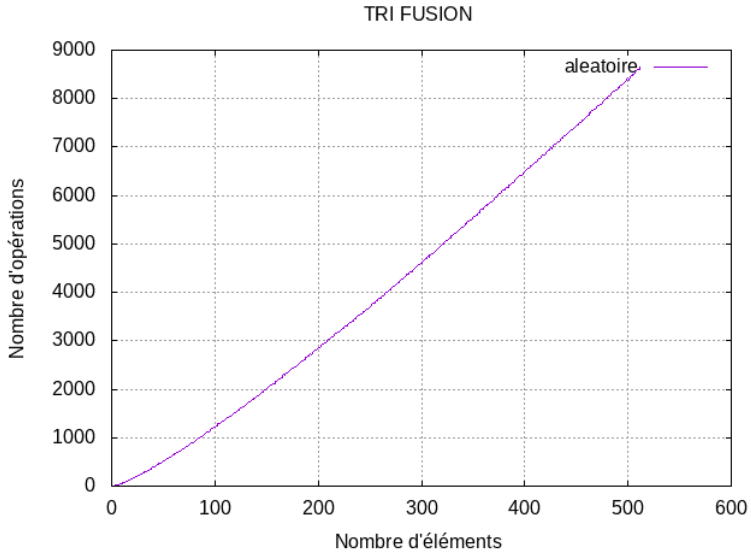


FIGURE 3 – Résultat Tri Fusion - Graphique

Avec ces résultant, on peux conclure que la Tri Fusion est très rapide et son complexité est ***O(n)***.

# 3 Tri Rapide

## 3.1 Développement

Cette partie est composée de trois fonction :

— ***Partitionnement*** : Responsable pour positionner a gauche toute les valeur inférieur a le pivot et a droit, les supérieur ;

— ***Tri_rapide*** : Fonction qui applique l'algorithme de Tri Rapide en utilisant lui même et la fonction Partitionnement ;

— ***quickSort*** : Fonction qui applique le Tri_rapide pour une T_data donnée ;

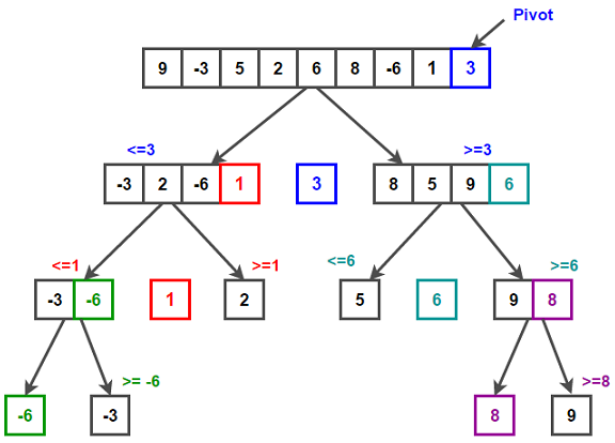


FIGURE 4 – Représentation graphique de la Tri Rapide

### 3.1.1 Code *quick_sort.c*

```
#include<stdio.h>
#include "test_utils.h"
#include "../include/traces.h"

// TODO : placer les compteurs aux endroits appropri s :
// stats.nbOperations ++;
// stats.nbComparisons ++;

int Partitionnement (T_elt t [], int g, int d);

int comparer(T_elt e1, T_elt e2) {

    stats.nbComparisons++;
    return e1-e2;
}

void echanger(T_elt t[], int i1, int i2) {

    T_elt aux = t[i1];
    t[i1] = t[i2];
    t[i2] = aux;

}


T_data quickSort(T_data d, int n){

    T_elt * A = d.pElt;

    Tri_rapide(A,0,n-1);

    return genData(0, A);
}

void Tri_rapide( T_elt t[], int debut, int fin) {

    stats.nbComparisons++;
    if(fin>debut){

        //Am lioration: Pivote    chaque it ration
        //pour  viter  le cas de la d g n rescence
        echanger(t, debut + rand() % (fin - debut), fin);

        int pivot_loc = Partitionnement(t, debut, fin);
        Tri_rapide(t, debut, pivot_loc-1);
        Tri_rapide(t, pivot_loc+1, fin);
    }
}




int Partitionnement (T_elt t [], int g, int d){
```

```
    int pg=g , pd=d-1; // On utilise g et d comme    pointeurs    qui se d placent
    // On choisit le dernier   lment   comme pivot

    while (pg<pd) {
        // On d place pg et pd  j u s q u  trouver des valeurs incoh rentes % pivot
        stats.nbComparisons++;
        while ( (pg<pd) &&  (comparer(t[pg],t[d]) <=0) ) {pg++ ;    stats.nbComparison
        while ( (pg<pd) && (comparer(t[pd],t[d])>0) ) {pd-- ; stats.nbComparisons ++;

        // Comment compter correctement ? On utilise une fonction de comparaison...
        if (pg < pd) {
            stats.nbOperations++;
            echanger(t,pg,pd);
            pg++ ; pd-- ;
        }
    }
    stats.nbComparisons ++;
    if (comparer(t[pg],t[d]) <= 0) pg++ ;

    stats.nbOperations+=3;
    echanger(t, pg, d) ;

    return pg ;
}
```

## 3.2 Résultat

```
fernandobdf23@fernandofonseca-VM:~/AAP/S3_TEA$ ./TriRapide/TriRapide.exe
+------------------------------+
| QUICK SORT                   |
+-----------+-----------------+----------------+----------------+----------------+
|   Taille  |           Mode  |    Nb compar.  |      Nb opér.  |    Duree (ms)  |
+-----------+-----------------+----------------+----------------+----------------+
|    2500   |        ordonne  |         70159  |          4680  |           0    |
|    5000   |        ordonne  |        148098  |          9564  |           0    |
|    7500   |        ordonne  |        232440  |         14235  |           0    |
|   10000   |        ordonne  |        321118  |         18972  |           1    |
|   12500   |        ordonne  |        454424  |         23688  |           1    |
|   15000   |        ordonne  |        513807  |         28446  |           1    |
|   17500   |        ordonne  |        651365  |         33219  |           1    |
|   20000   |        ordonne  |        682849  |         37854  |           1    |
|   22500   |        ordonne  |        903219  |         42717  |           2    |
|   25000   |        ordonne  |        914880  |         47286  |           2    |
+-----------+-----------------+----------------+----------------+----------------+


+------------------------------+
| QUICK SORT                   |
+-----------+-----------------+----------------+----------------+----------------+
|   Taille  |           Mode  |    Nb compar.  |      Nb opér.  |    Duree (ms)  |
+-----------+-----------------+----------------+----------------+----------------+
|    2500   |      aleatoire  |         60103  |         10151  |           0    |
|    5000   |      aleatoire  |        136164  |         21413  |           1    |
|    7500   |      aleatoire  |        224294  |         32801  |           1    |
|   10000   |      aleatoire  |        297972  |         45070  |           1    |
|   12500   |      aleatoire  |        386122  |         56958  |           1    |
|   15000   |      aleatoire  |        487982  |         69064  |           2    |
|   17500   |      aleatoire  |        560362  |         81973  |           2    |
|   20000   |      aleatoire  |        649886  |         94226  |           2    |
|   22500   |      aleatoire  |        729148  |        107486  |           3    |
|   25000   |      aleatoire  |        803063  |        120633  |           3    |
+-----------+-----------------+----------------+----------------+----------------+


+------------------------------+
| QUICK SORT                   |
+-----------+-----------------+----------------+----------------+----------------+
|   Taille  |           Mode  |    Nb compar.  |      Nb opér.  |    Duree (ms)  |
+-----------+-----------------+----------------+----------------+----------------+
|    2500   |        inverse  |         75065  |          5997  |           0    |
|    5000   |        inverse  |        139467  |         12036  |           0    |
|    7500   |        inverse  |        228436  |         18012  |           0    |
|   10000   |        inverse  |        318300  |         23964  |           1    |
|   12500   |        inverse  |        447031  |         29920  |           1    |
|   15000   |        inverse  |        514945  |         36055  |           1    |
|   17500   |        inverse  |        585585  |         42065  |           1    |
|   20000   |        inverse  |        729155  |         47873  |           2    |
|   22500   |        inverse  |        777989  |         53859  |           2    |
|   25000   |        inverse  |        864556  |         59934  |           2    |
+-----------+-----------------+----------------+----------------+----------------+
```

FIGURE 5 – Résultat Tri Rapide - Comparaison

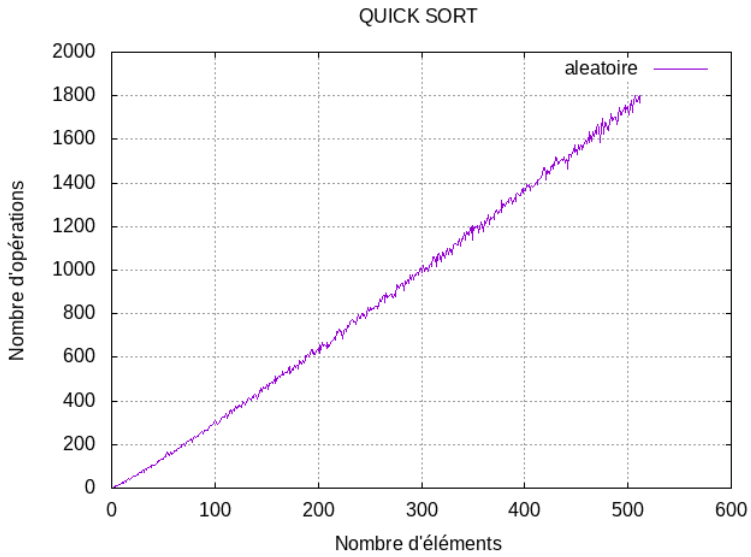


FIGURE 6 – Résultat Tri Rapide - Graphique

Avec ces résultant, on peux conclure que la Tri Rapide est très rapide et son complexité est ***O(n logn)***.

# 4 Tri Fusion de listes

## 4.1 Développement

Cette partie est composée de trois fonction :

- ***getpelt*** : Responsable pour retourner un pointeur pour le n-ime valeur de la liste ;
- ***fusionnerList*** : Responsable pour comparé les élément de 2 listes et les ordonnée ;
- ***fusionSortList*** : Fonction qui applique le algorithme de Tri Fusion en utilisant lui même et la fonction fusionner ;

### 4.1.1 Code *fusion_sort_list.c*

```
#include <string.h>
#include <stdio.h>
#include "test_utils.h"
#include "list_v2.h"

#include "../include/traces.h"

// TODO : placer les compteurs aux endroits appropri s :
// stats.nbOperations ++;
// stats.nbComparisons ++;

T_list getp_elt(T_list l, int i);
```

```c
void fusionnerList(T_list l, int d, int m, int f);


T_list fusionSortList(T_list l, int n) {


    if(n==1) return l;
    int m = n/2;

    T_list leftArray = fusionSortList(l, m);
    T_list rightArray = fusionSortList(getp_elt(l,m), (n-m));

    fusionnerList(l, 0, m-1, n-1);

    return l;
}


T_list getp_elt(T_list l, int pos) {
    int i=0;
    T_list aux = l;

    while (i < pos) {
        aux = aux->pNext;
        i++;
    }

    return aux;
}

void fusionnerList(T_list l, int d, int m, int f) {
    T_elt aux[f - d + 1]; // !! Allocation dynamique sur la pile (standard C99)
    int i, j, k=0;

    T_list listIni = getp_elt(l, d);
    T_list listAux = listIni;

    for(j=d; j<=f; j++){
        aux[k++] = listAux->data;
        listAux = listAux->pNext;
    }

    i = 0; j = m - d + 1; k = 0;
    while (i <= m - d && j <= f - d) {

        if (aux[i] <= aux[j]){
            getp_elt(listIni, k++)->data = aux[i++];    // aux[i] est plus petit : on
        }
        else {
            getp_elt(listIni, k++)->data = aux[j++];    // aux[j] est plus petit : on
        }
    }

    for (; i <= m - d; getp_elt(listIni, k++)->data = aux[i++]); // le reste du tablea
    for (; j <= f - d; getp_elt(listIni, k++)->data = aux[j++]); // le reste du tablea
}
```

Pour tester l'algorithme, nous avons faire aussi une ***main.c*** qui qui crée une liste chaînée et organise ses éléments en utilisant le fonction ***fusionSortList*** :

### 4.1.2 Code *main.c*

```
#include <stdio.h>
#include <stdlib.h>

#include "../include/traces.h"

#include "test_utils.h"
#include "list_v2.h"
#include "elt.h"

// Ajouter ici les prototypes des fonctions    tester
T_list fusionSortList(T_list l, int n);


// mode, label, x, checkOrder
T_mode m[] = {
    {MODE_TAB_ORDONNE, "ordonne", 0, 1},
    {MODE_TAB_ALEATOIRE, "aleatoire", 0, 1},
    {MODE_TAB_INVERSE, "inverse", 0, 1},
    {MODE_EVAL_X, "x=2.0", 2.0, 0},
    {MODE_TAB_ORDONNE, "ordonne (x=59)", 59, 0},
    {MODE_TAB_ORDONNE, "hanoi", 1, 0}
};

int main(int argc, char *argv[])
{

    T_list l = NULL;

    l = addNode(10, l);
    l = addNode(14, l);
    l = addNode(2, l);
    l = addNode(28, l);
    l = addNode(90, l);

    printf("Before sort: ");
    showList(l); NL();

    l = fusionSortList(l, getSizeIte(l));

    printf("After sort: ");
    showList(l); NL();

    return 0;




}
```

### 4.2 Résultat

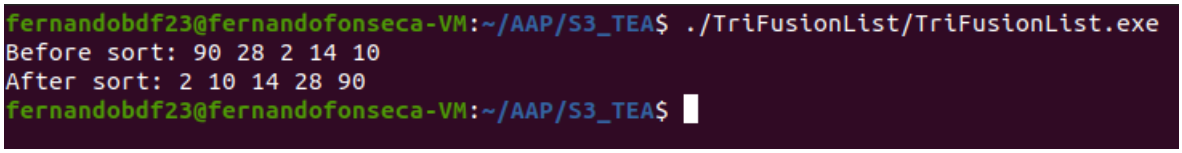


FIGURE 7 – Résultat Tri Fusion - Listes Chaînées

## 5 Conclusion

Dans ce TEA, nous avons bien travailler avec des algorithmes de tri. Aussi nous avons étudie ses avantages et performances.

## 6 Réferences


— AAP - Séance 3
— AAP - TEA 3
— Merge Sort
— QuickSort