Compte-rendu du TEA n°3

Valentin GAUTHIER
Alexandre TORRES–LEGUET

December 8, 2022

1 Introduction

Ce compte-rendu présente le travail réalisé dans le cadre du TEA 3 de l'enseignement d'AAP, qui se concentre sur le tri rapide et le tri fusion en C.

Nous avons vu en TD le principe de fonctionnement et le code de ces tris en C, notre tâche a alors été dans un premier temps d'adapter ce travail afin de le rendre compatible avec les outils de création de graphiques, i.e. avec la structure de données `T_data`.

Dans un second temps, nous avons implémenté les fonctions `fusionsort` et `quicksort` qui respectent le même prototype que la fonction `qsort` disponible nativement dans C. (sous réserve d'inclure `stdlib.h`)

Enfin, nous avons implémenter le tri fusion sur des listes chaînées.

2 Développement

2.1 Tri fusion

Le dossier `tri_fusion` recense notre travail relatif au tri fusion sur un tableau.

Pour cette partie, la première partie du travail a été de transformer la fonction d'en-tête

```
void triFusion(T_elt t [], int debut, int fin)
```

en la fonction

```
void triFusion(T_data d, int n)
```

où `n` désigne le nombre d'éléments à considérer à partir du début du tableau passé. Il faut donc, au moment d'appeler cette deuxième fonction, envoyer la bonne partie du tableau (et non pas le tableau entier comme c'est le cas pour la première fonction).

Nous avons effectué ensemble la fonction `fusionsort` une fois la fonction `quicksort` implémentée car cette dernière nous a paru plus abordable pour gérer le type `void *` que l'on récupère en entrée. Le développement de `fusionsort` a requis celui de la fonction `merge` qui permet de fusionner deux tableaux triés. Pour faire cela, nous avons été obligé de copier dans des tableaux temporaires (`L` et `R`) les données à fusionner. Nous aidant des considérations faites pendant le développement de `quicksort`, nous nous sommes beaucoup basés sur la fonction `memcpy`.

2.2 Tri rapide

Le dossier `tri_rapide` recense notre travail relatif au tri fusion sur un tableau.

Le même travail de conversion a été effectué afin de rendre compatible la fonction vue en TD au type `T_data`.

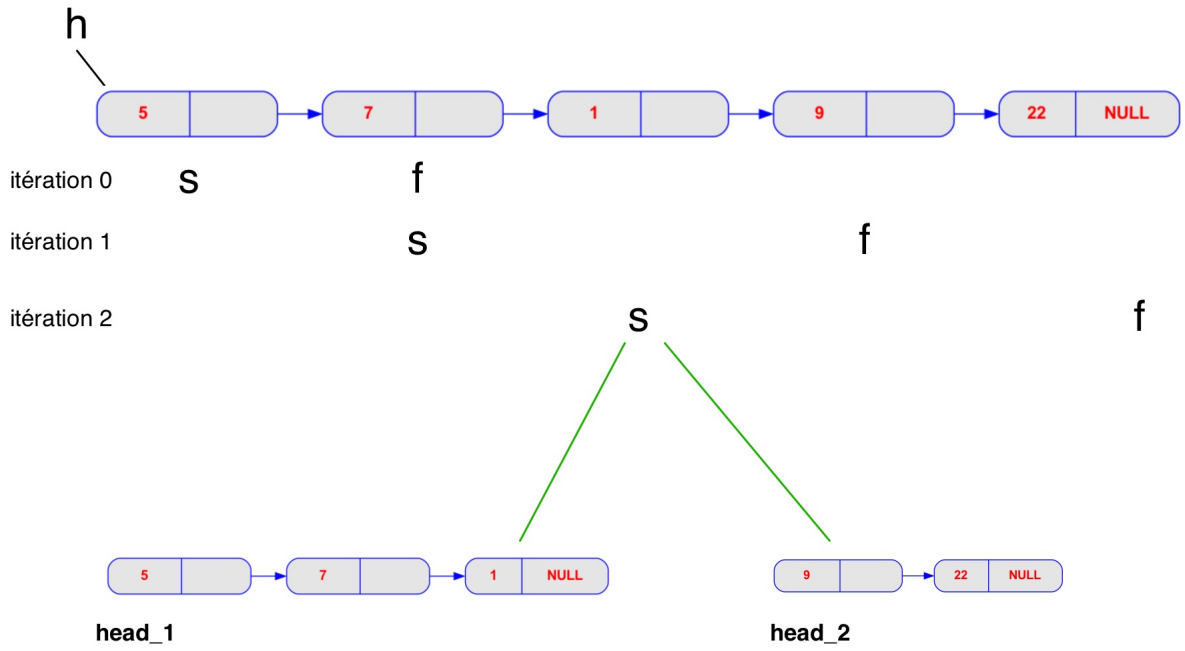
La fonction `quicksort` reprend les mêmes idées que `triRapide`. Cependant, l'implémentation concrète du tri a été amené à changer. La plus grosse difficulté rencontré a été que le tableau passé en entrée est de type `void *` (i.e. type non déterminé). Cela a notamment posé un problème dans la fonction annexe `echanger` qui a pour but d'échanger 2 éléments d'un tableau. En effet, le compilateur nous empêche d'assigner une valeur à type `void *` : la solution pour pallier à cela a été d'utiliser la fonction native `memcpy` afin de ne pas avoir à assigner une valeur.

2.3 Tri rapide sur une liste chaînée

Le dossier `tri_fusion_listes` recense notre travail relatif au tri fusion sur une liste chaînée.

Nous nous sommes d'abord concentré sur la fonction `couper`, qui a pour but de diviser en 2 moitié une liste chaînée. Cette fonction, au lieu de retourner 2 valeurs qui sont les têtes des 2 listes, modifie 2 pointeurs (`T_node **head_1` et `T_node **head_2`) qui, une fois la fonction terminée, pointeront sur les 2 listes divisées.

Voici un schéma explicatif qui montre le déroulement de cette division :

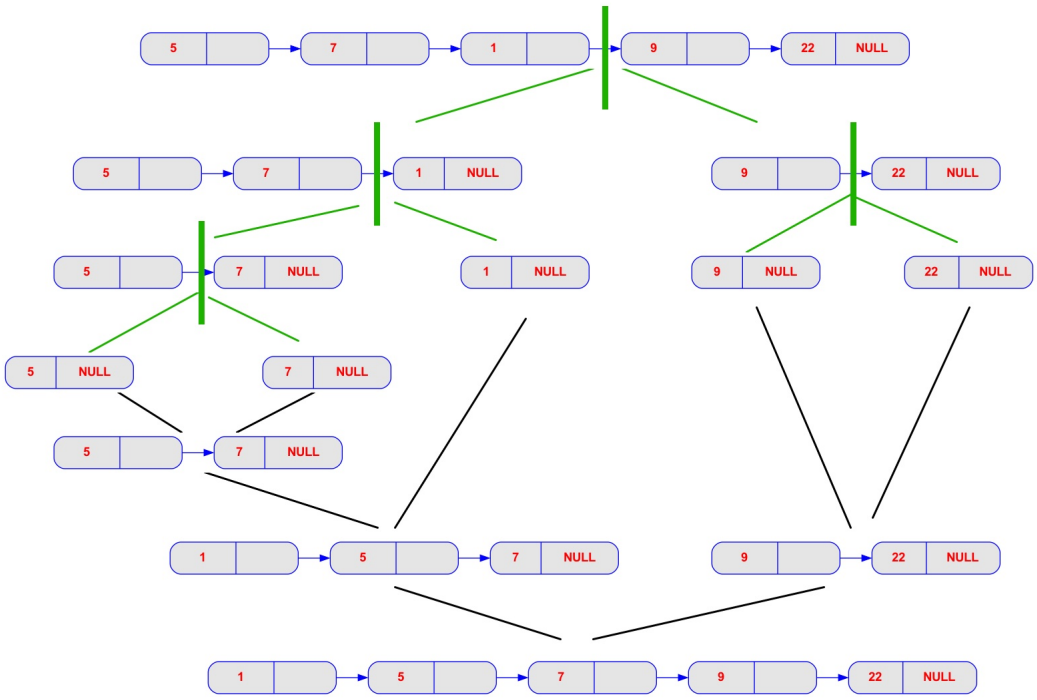


`s` et `f` correspondent aux variables `slow` et `fast` de notre code : ces pointeurs parcourent la liste, mais un le fait de manière "lente" (une maille à la fois) et l'autre le fait plus rapidement (deux mailles à la fois). Il s'en suit que, lorsqu'on les fait parcourir la liste, à la fin, lorsque `f` atteint le bout de la liste, `s` se trouve au milieu de la liste. On s'en sert alors pour découper la liste en deux. On peut comparer cette situation à la suivante : deux personnes montent le même escalier, et, quand la première personne monte une marche, la seconde en monte deux. Alors, lorsque la seconde personne sera arrivée en haut de l'escalier, la première en sera à la moitié.

La fonction `fusion` fusionne 2 listes chaînées supposées triées. Le principe est le même qu'avec le tableau : on compare les premiers éléments de chacun des deux listes à fusionner, on sélectionne le petit, puis on recommence... jusqu'à épuisement des deux listes.

La fonction `triFusionListes` se tâche simplement de diviser la liste d'entrée, trier récursivement les deux listes obtenues, et de les fusionner.

Voici un schéma permettant d'illustrer le tri fusion sur une liste chaînée :



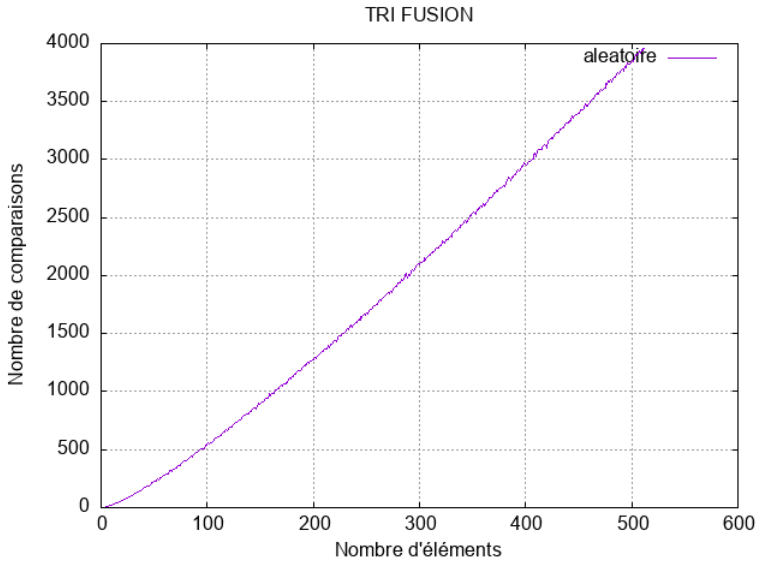
2.4 Organisation - Gestion de projet

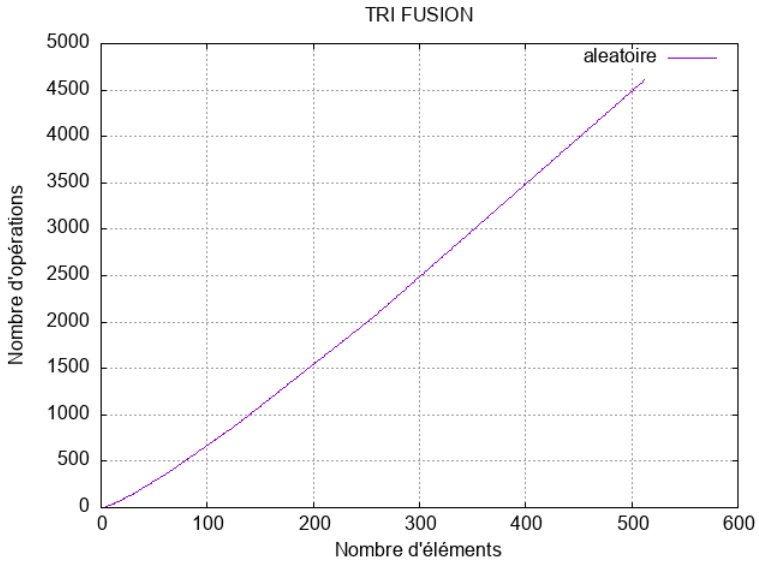
Concernant l'organisation des tâches au sein du groupe, Valentin s'est chargé de la partie concernant le tri fusion et Alexandre celle du tri rapide. Ce travail a été opéré le week-end qui a suivi notre séance de TP du vendredi.

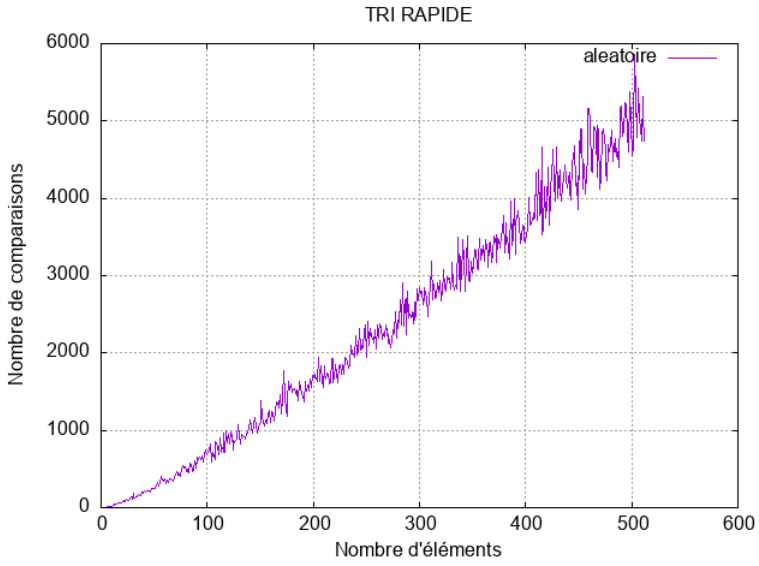
Enfin, au cours du début de la semaine suivante, nous avons pu nous réunir afin de réfléchir à adapter et implémenter l'algorithme de tri fusion sur une liste chaînée. Pour cela, nous sommes revenus sur la séance de TP antécédente pendant laquelle nous avions implémenté le type de liste chaînée (fichiers `list.h` et `list.c`).

3 Résultats

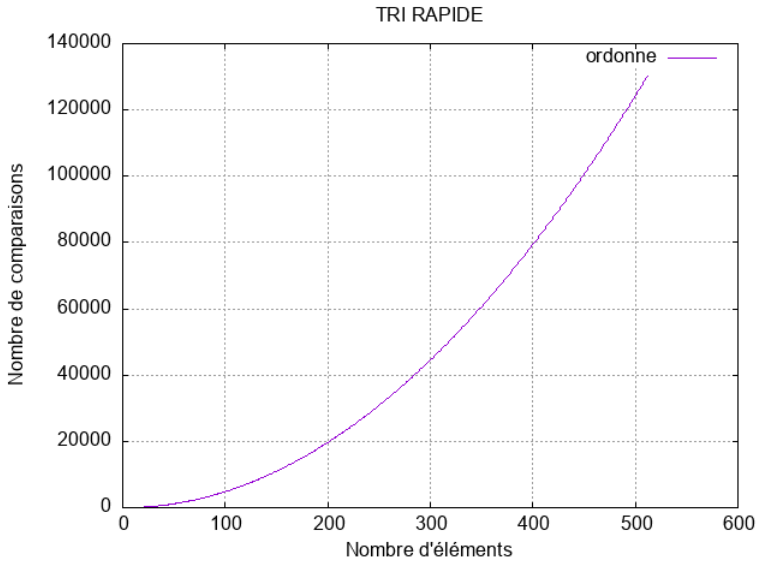
Voici le comportement de nos algorithmes sur des tableaux de grande taille. On observe bien un comportement quasi-linéaire pour les tris fusion et rapide sur des tableaux de nombre aléatoires.


TRI FUSION
aleatoire
Nombre de comparaisons
Nombre d'éléments
0
500
1000
1500
2000
2500
3000
3500
4000
0
100
200
300
400
500
600

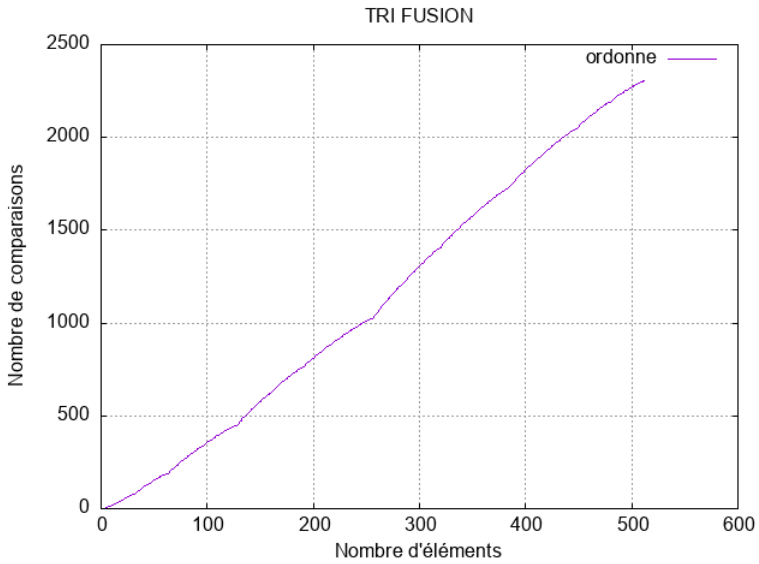

TRI FUSION
aleatoire
Nombre d'opérations
Nombre d'éléments
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
0
100
200
300
400
500
600



Toutefois, le tri rapide d'un tableau ordonné laisse place à un comportement en $\Theta(n^2)$: c'est la dégénérescence du tri rapide. En effet, le tri rapide comme nous l'avons implémenté dans un premier temps choisit pour pivot le dernier élément du tableau. Ainsi, si le tableau est déjà trié, le pivot est le nombre le plus grand du tableau : les deux sous-tableaux construits sont donc le tableau vide et le tableau contenant tous les éléments sauf le pivot : on perd donc en efficacité.



L'idée pour pallier à ce problème est de choisir le pivot tel que les deux sous-tableaux soient équilibrés. Une solution simple est de prendre le pivot comme étant l'élément du milieu. C'est ce que nous avons fait et voici le résultat :



Une autre méthode consiste à prendre 3 éléments du tableau au hasard, et à choisir l'élément médian parmi ces 3. Nous n'avons pas testé cette technique.

Nous avons enfin comparé nos implémentations des tris rapide et fusion avec la fonction `qsort` native de C. Pour cela, nous avons utilisé la bibliothèque `time` afin de mesurer le temps d'exécution de nos programmes. Les temps que l'on voit apparaître dans le tableau correspondent au tri de 100 tableaux de taille `MAX_ELT = 250 000`.

`quicksort`	`fusionsort`	`qsort`
11.2s	12.1s	4.1s

On remarque que l'implémentation native de `qsort` bat largement nos deux implémentations.

4 Conclusion

Nous avons pu, au cours de ce TEA, nous familiariser grandement avec le tri fusion et le tri rapide, ainsi que leurs performances globales. L'utilisation du tri fusion sur les listes chaînes nous a permis de nous rendre compte de l'avantage d'utiliser des listes afin de réduire l'empreinte mémoire de nos programmes.