Analyse des performances des tris

NIDDAM Benjamin

28 Novembre 2020

En 1962, Tony Hoare inventa le tri rapide (Quicksort), qui est généralement considéré comme l'algorithme le plus utilisé dans le monde entier.

Table des matières

1 Introduction

1.1 Définition

Un algorithme de tri est, en informatique ou en mathématiques, un algorithme qui permet d'organiser une collection d'objets selon une relation d'ordre déterminée.
La collection à trier est souvent donnée sous forme de tableau, afin de permettre l'accès direct aux différents éléments de la collection, ou sous forme de liste, ce qui peut se révéler être plus adapté à certains algorithmes et à l'usage de la programmation fonctionnelle.

Aujourd'hui on retrouve de nombreux algorithmes de tri que l'on peut diviser en 3 grandes catégories :

- Les Tris par comparaison
- Les Tris utilisant la structure de données
- Les Tris externes

Les tris par comparaison peuvent encore se diviser en 4 sous catégories en fonction de leur vitesse d'exécution :

- Les algorithmes rapides avec une complexité moyenne de $n * \log(n)$
- Les algorithmes moyennement rapides (En moyenne O(n^2), O(n) dans le meilleur des cas)
- Les algorithmes lents avec une complexité de O(n^2) dans tous les cas
- Les algorithmes très lents avec une complexité moyenne moins bonne que O(n^2)

1.2 Exemples

1.2.1 Les Tris par comparaison

- Le tri fusion (dit merge sort)
- Le tri rapide (dit quicksort)
- Le tri par tas (dit heap sort)
- Le tri fusion (dit merge sort)
- Le tri par insertion
- Le tri à bulle
- Le tri par sélection
- Le tri stupide
- Le tri faire-valoir

1.2.2 Les Tris utilisant la structure de données

— Le tri comptage ou tri par dénombrement
— Le tri par base
— Le tri par paquets

1.2.3 Les Tris externes

Ces algorithmes sont souvent basés sur une approche assez voisine de celle du tri fusion. Le principe est le suivant :

— découpage du volume de données à trier en sous-ensembles de taille inférieure à la mémoire rapide disponible ;
— tri de chaque sous-ensemble en mémoire centrale pour former des « monotonies » (sous-ensembles triés) ;
— interclassement des monotonies.

2 Critères de classification

La classification des algorithmes de tri est très importante, car elle permet de choisir l'algorithme le plus adapté au problème traité, tout en tenant compte des contraintes imposées par celui-ci. Les principales caractéristiques qui permettent de différencier les algorithmes de tri, outre leur principe de fonctionnement, sont la complexité temporelle, la complexité spatiale et le caractère stable.

2.1 Principe de fonctionnement

2.2 Complexité algorithmique

Afin d'évaluer la complexité des différents algorithmes de tri présentés, on comptera le nombre de comparaisons et d'échanges de valeur entre deux éléments du tableau sans prendre en compte les affectations et comparaisons sur des variables de comptage de boucles.

Les méthodes présentées sont de deux types :

— des méthodes qui trient les éléments deux à deux, de manière plus ou moins efficace, mais qui nécessitent toujours de comparer chacun des N éléments avec chacun des N-1 autres éléments, donc le nombre de comparaisons sera de l'ordre de n^2 — on note cet ordre de grandeur $O(\mathrm{n}^2)$.

Par exemple, pour n=1000, n^2=10^6, pour n=10^6, n^2=10^{12}.

Les algorithmes de ce type sont :
- une méthode de tri élémentaire, le tri par sélection ;
- et sa variante, le tri par propagation ou tri bulle ;
- une méthode qui s'apparente à celle utilisée pour trier ses cartes dans un jeu, le tri par insertion ;

— des des méthodes qui sont plus rapides, car elles trient des sous-ensembles de ces N éléments puis regroupent les éléments triés, elles illustrent le principe « diviser pour régner ». Le nombre de comparaisons est alors de l'ordre de $n * \log(n)$.

Par exemple, pour n=1000, $n * \log(n)$=10000 environ, pour n=10^6, $n * \log(n)$=$20 * 10^6$ environ.

Les algorithmes de ce type sont :
- le fameux tri rapide ou Quicksort ;
- et enfin, le tri par fusion.

Cette liste n'est évidemment pas exhaustive. Il existe des méthodes particulièrement adaptées à certains types de données spécifiques. Le tri par base (dit radix sort) en est un exemple.

Complexité	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
$n = 30$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 min	10^{25} ans
$n = 50$	< 1 s	< 1 s	< 1 s	< 1 s	11 min	36 ans	∞
$n = 100$	< 1 s	< 1 s	< 1 s	$1s$	12, 9 ans	10^{17} ans	∞
$n = 1000$	< 1 s	< 1 s	1s	18 min	∞	∞	∞
$n = 10000$	< 1 s	< 1 s	2 min	12 jours	∞	∞	∞
$n = 100000$	< 1 s	2 s	3 heures	32 ans	∞	∞	∞
$n = 1000000$	1s	20s	12 jours	31, 710 ans	∞	∞	∞

FIGURE 1 - Temps d'éxécution en fonction de la compléxité et du nombre d'éléments

2.3 Complexité spatiale

2.4 Tri en place

2.5 Tri stable

2.6 Tri interne et externe

2.7 Tri parallèle

3 Présentation des tris

4 Comparaison des algorithmes

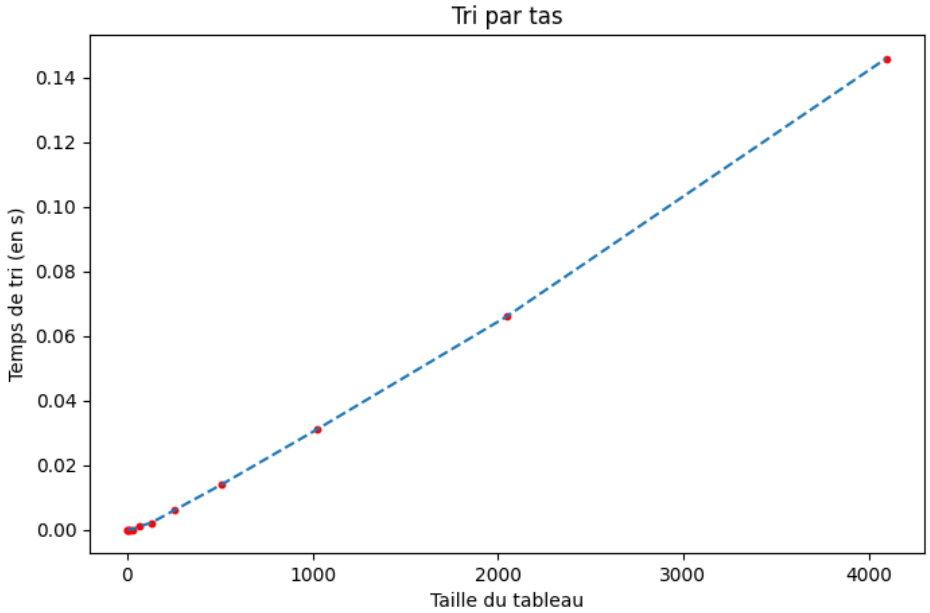
4.1 Lien avec les données

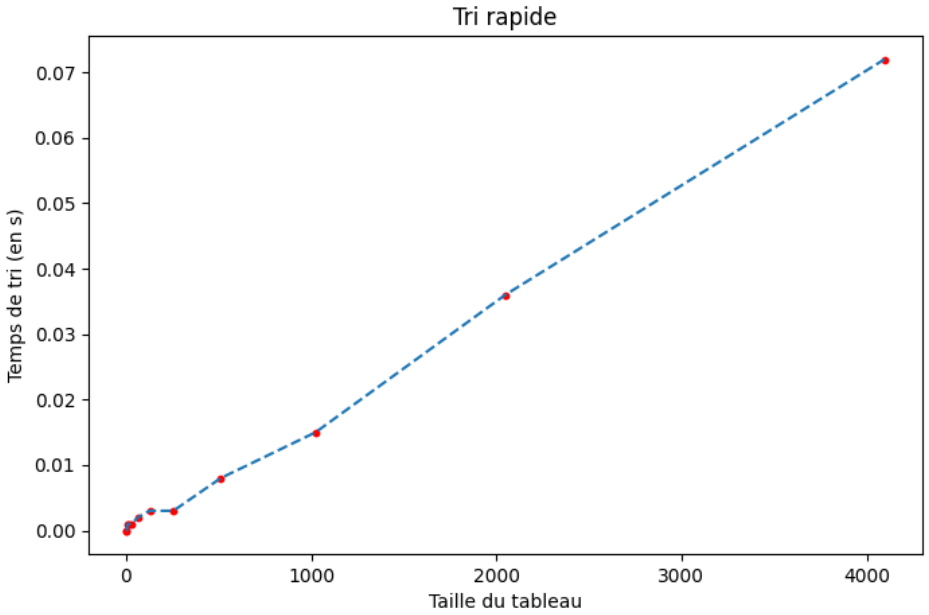
L'étude théorique des algorithmes n'est pas suffisante car l'efficacité de ces-derniers dépend aussi de l'ordre des données sur lesquelles on les utilise. Les algorithmes que nous allons considérer par la suite sont les tris : *par tas*, *comptage*, *fusion* et *rapide*. Pour les étudier et les comparer, on se propose de créer plusieurs ensembles de données. On ordonnera les données toujours par ordre croissant. De plus, on donnera aux données un ordre prédéfini (avant le tri). Les jeux de données seront ordonnés : par ordre croissant de 0 à n, par ordre décroissant de n à 0 (inversé), aléatoirement sans redondance de donnée, aléatoirement avec redondance. D'autre part, on fera varié la taille de ces jeux de données (on aura des tailles allant de n = 10 à n = 40960. Et enfin, pour chaque taille de tableau, nous trierons ces derniers à l'aide des quatre algorithmes donnés plus tôt. Précisons aussi que les algo trierons le même tableau pour conserver même conditions pour chaque test et que cette opération sera répétée quize fois et que nous ne garderons que la moyenne de ces essais.

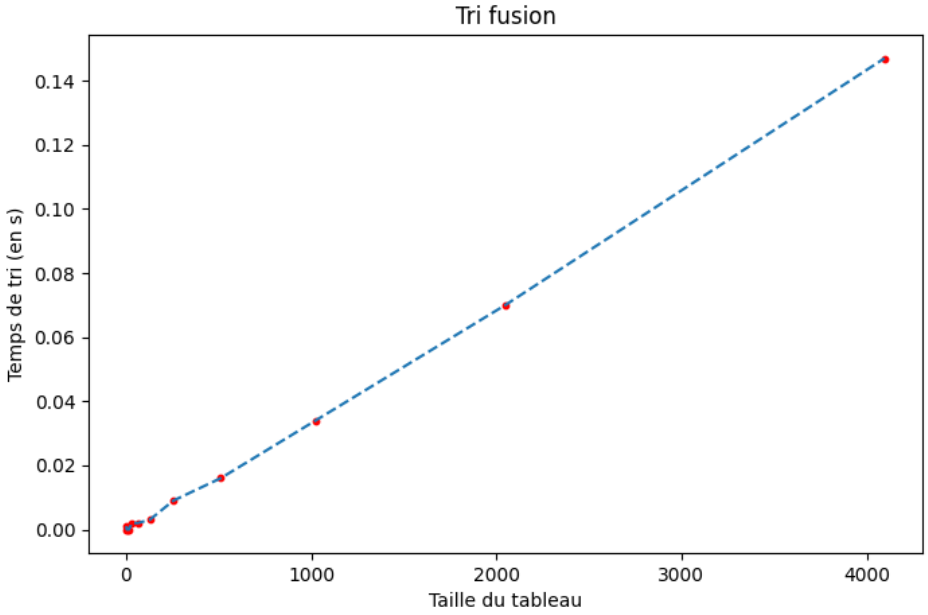
4.2 Traitement des données

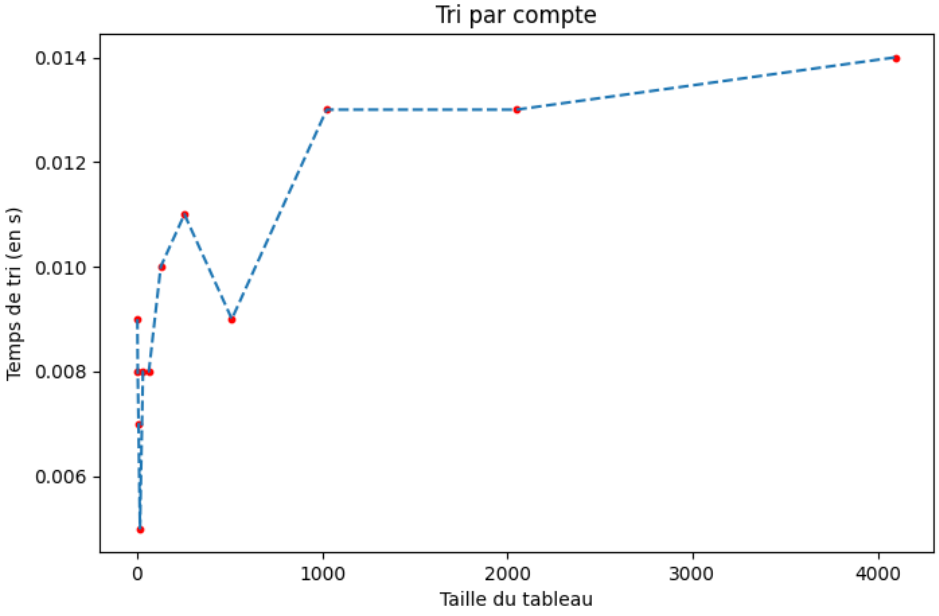
Pour observer les performances des tris sur nos jeux de données, il nous suffit d'utiliser les résultats écrits dans les fichiers créés par le programme en C. On peut traiter les données à la main ou utiliser un script Python (annexe). Ce script génère plusieurs tracés, on s'intéresse surtout aux tracés suivant :

4.2.1 Tris sur le jeu de données aléatoire avec redondance

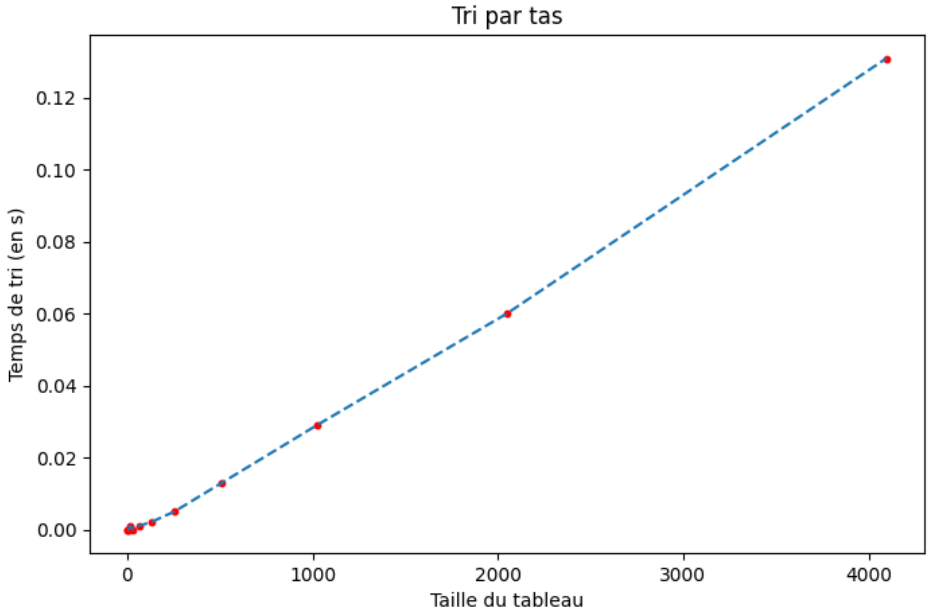


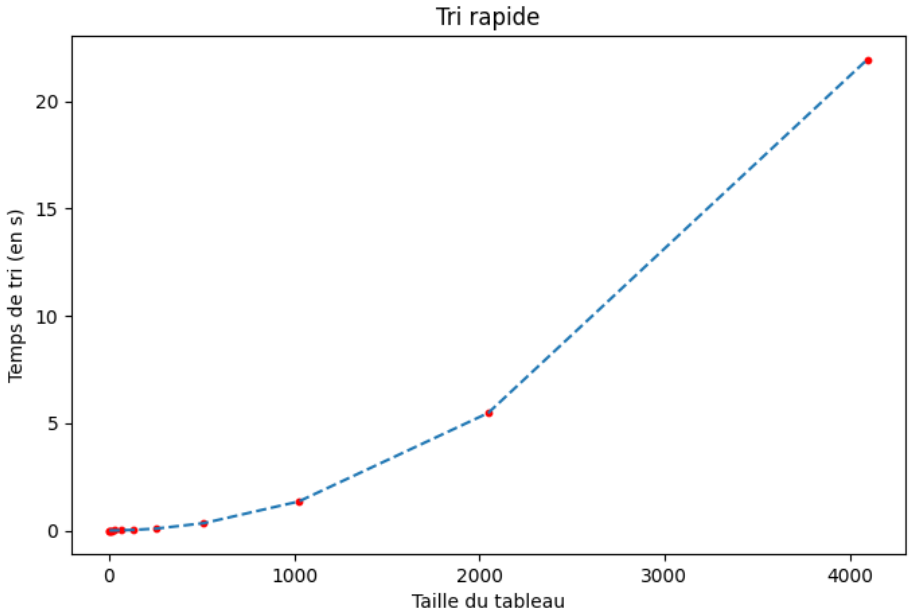


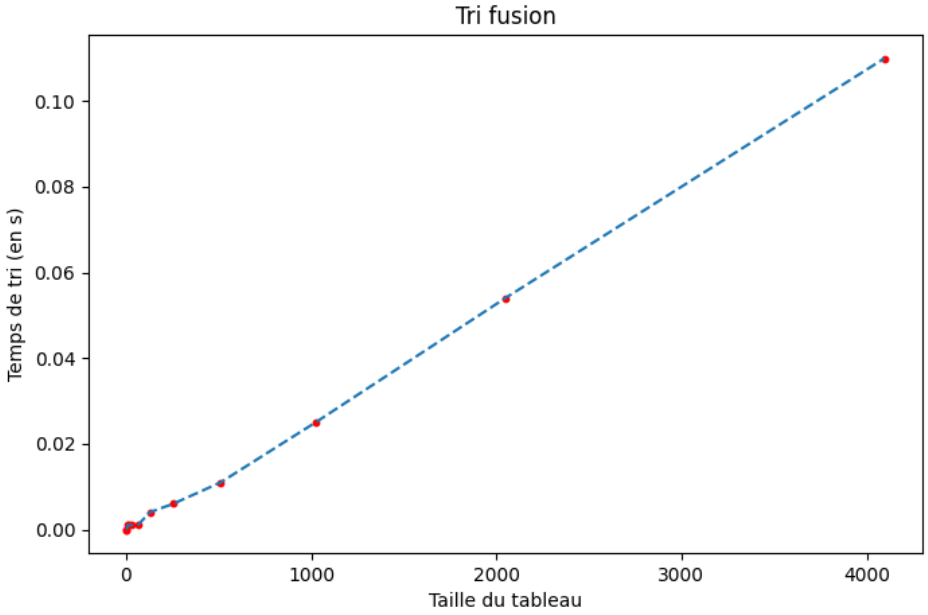

Tri fusion
Temps de tri (en s)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1000
2000
3000
4000
Taille du tableau

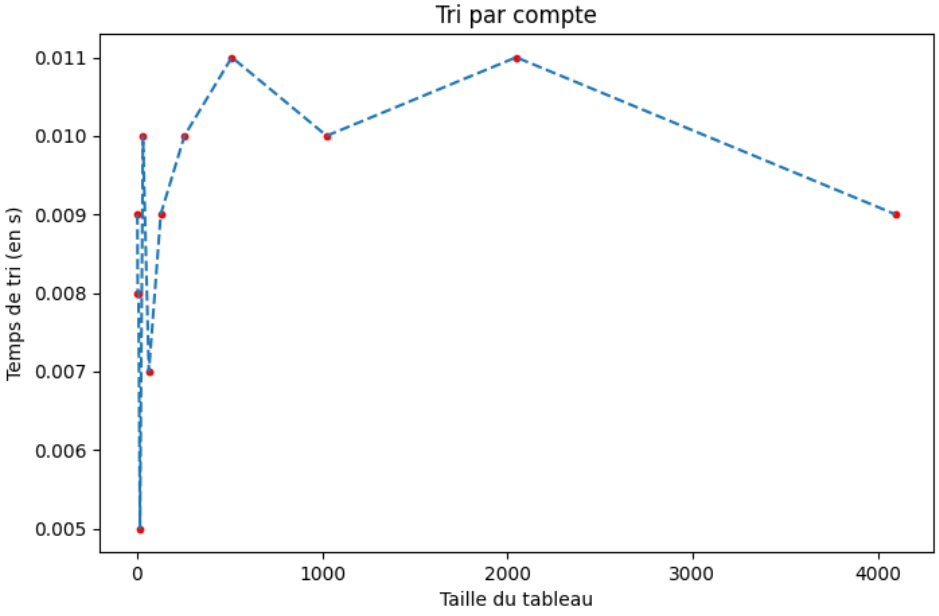

Tri par compte
Temps de tri (en s)
0.014
0.012
0.010
0.008
0.006
0
1000
2000
3000
4000
Taille du tableau

4.2.2 Tris sur le jeu de données ordonnées sans redondance

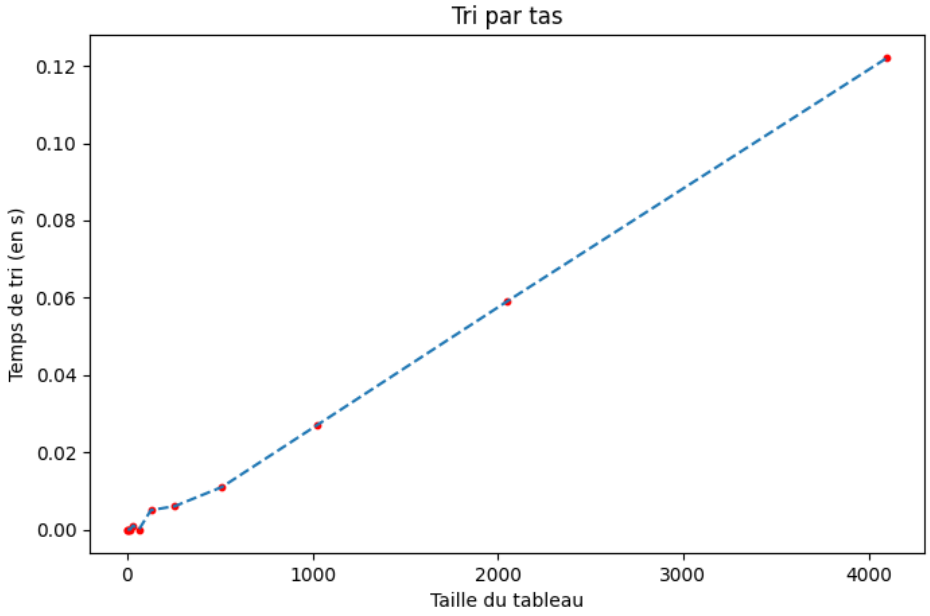


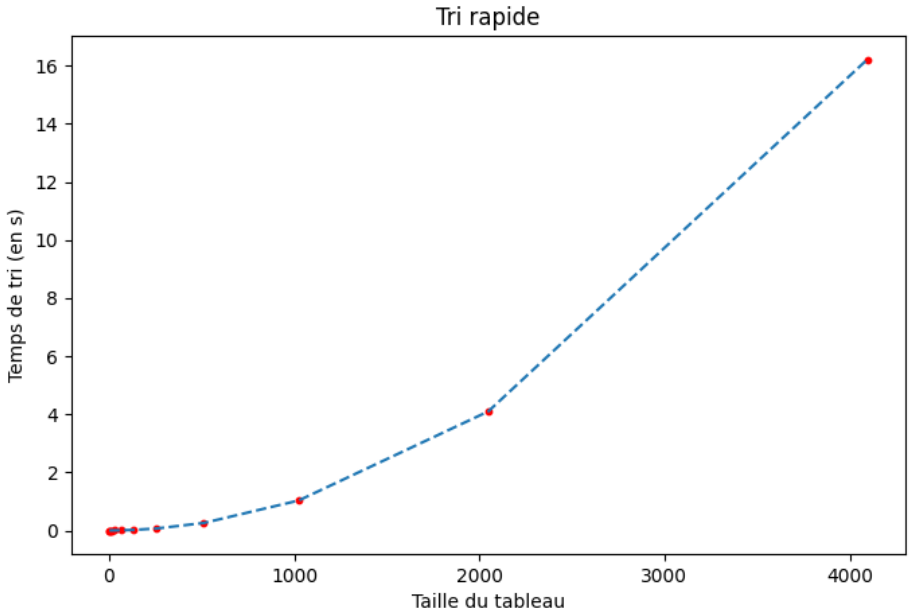


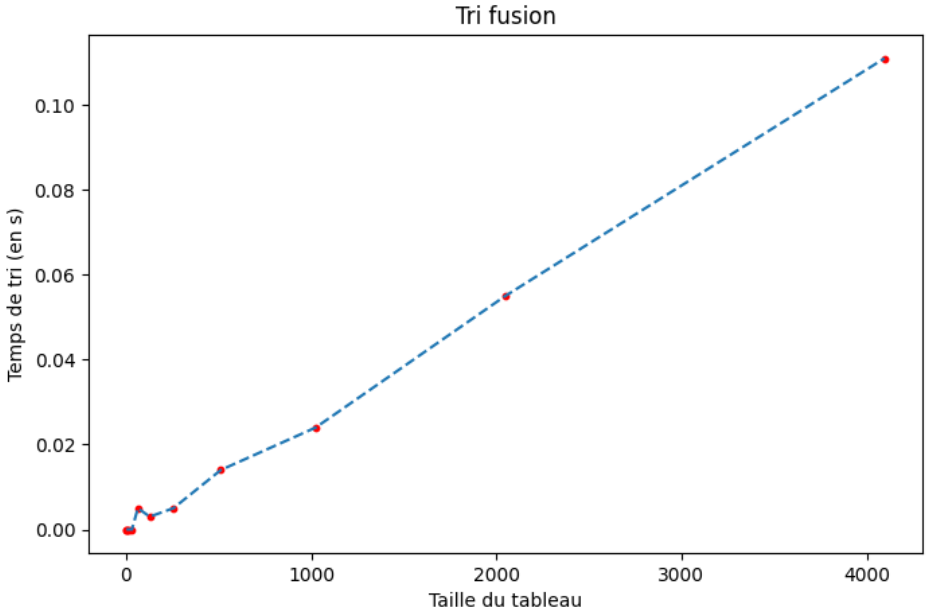

Tri fusion
Temps de tri (en s)
0.00
0.02
0.04
0.06
0.08
0.10
0
1000
2000
3000
4000
Taille du tableau

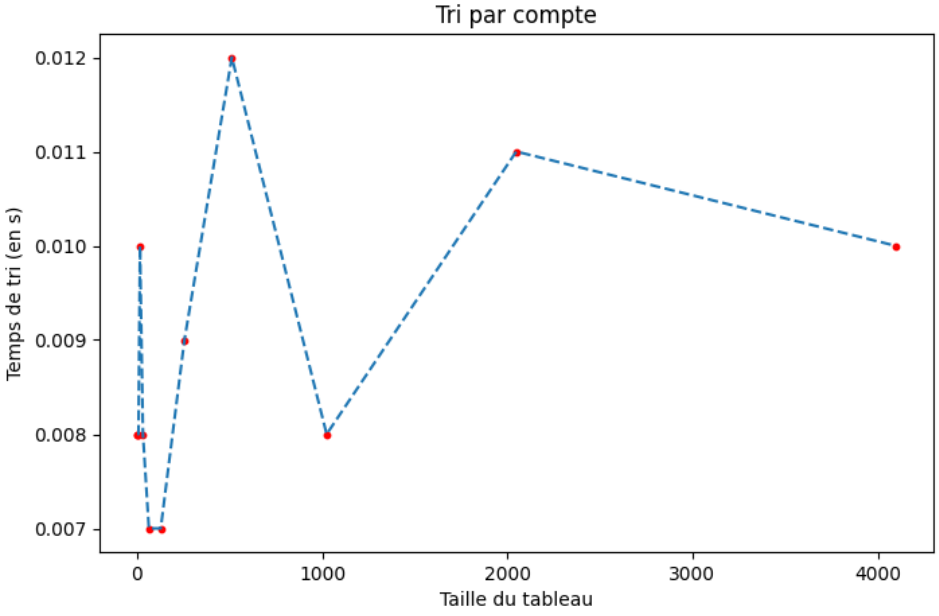

Tri par compte
Temps de tri (en s)
0.005
0.006
0.007
0.008
0.009
0.010
0.011
0
1000
2000
3000
4000
Taille du tableau

4.2.3 Tris sur le jeu de données ordonnées avec redondance

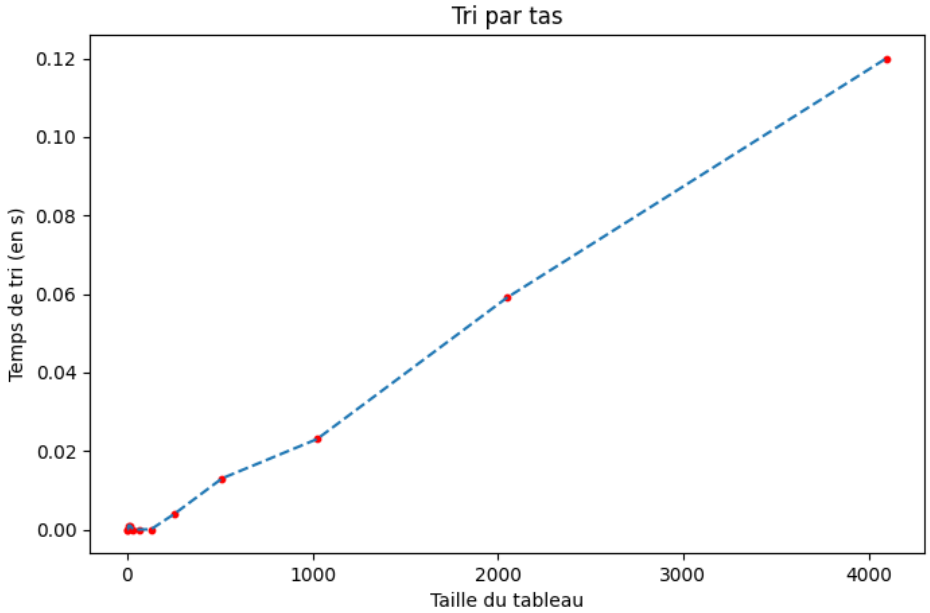


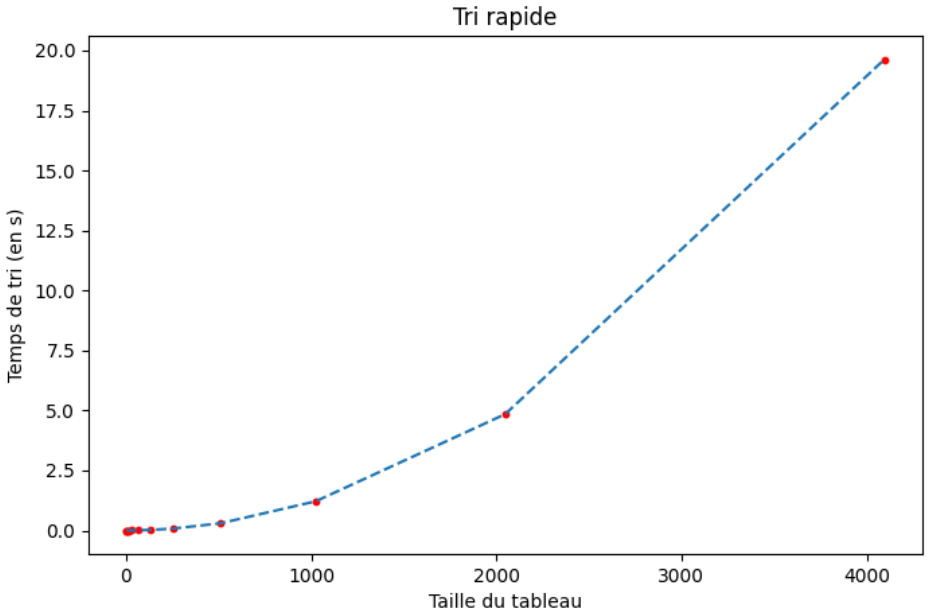


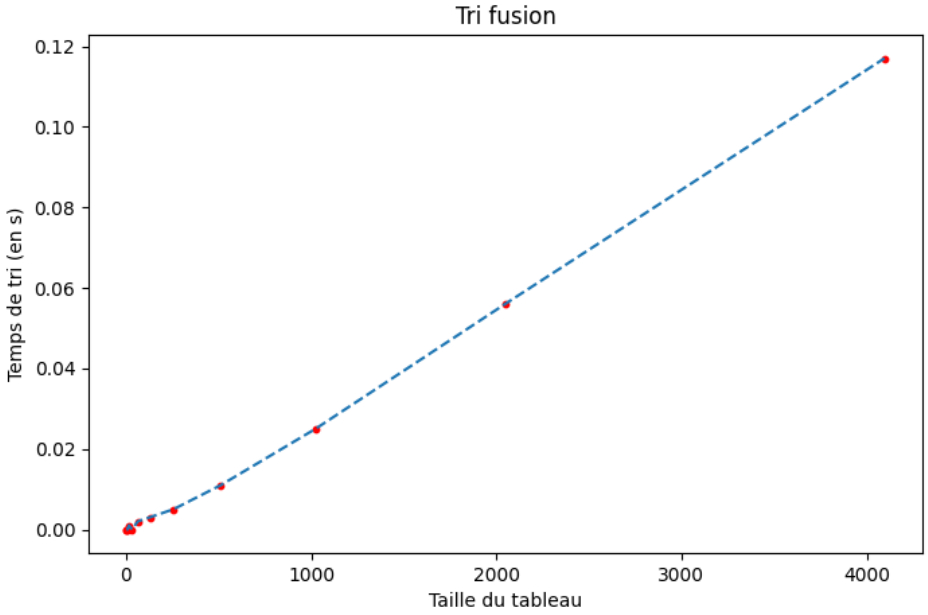

Tri fusion
Temps de tri (en s)
0.00
0.02
0.04
0.06
0.08
0.10
0
1000
2000
3000
4000
Taille du tableau

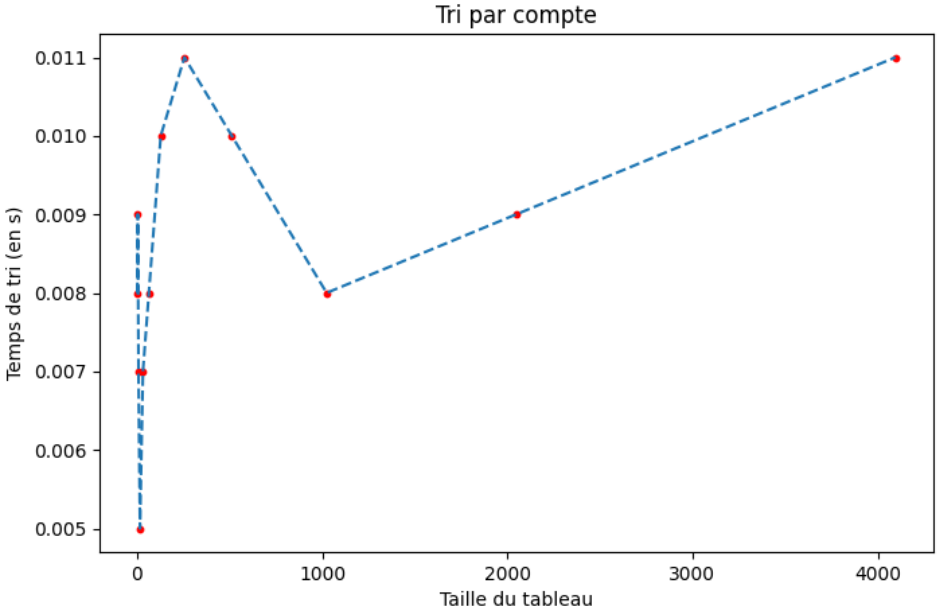

Tri par compte
Temps de tri (en s)
0.007
0.008
0.009
0.010
0.011
0.012
0
1000
2000
3000
4000
Taille du tableau

4.2.4 Tris sur le jeu de données inversées sans redondance

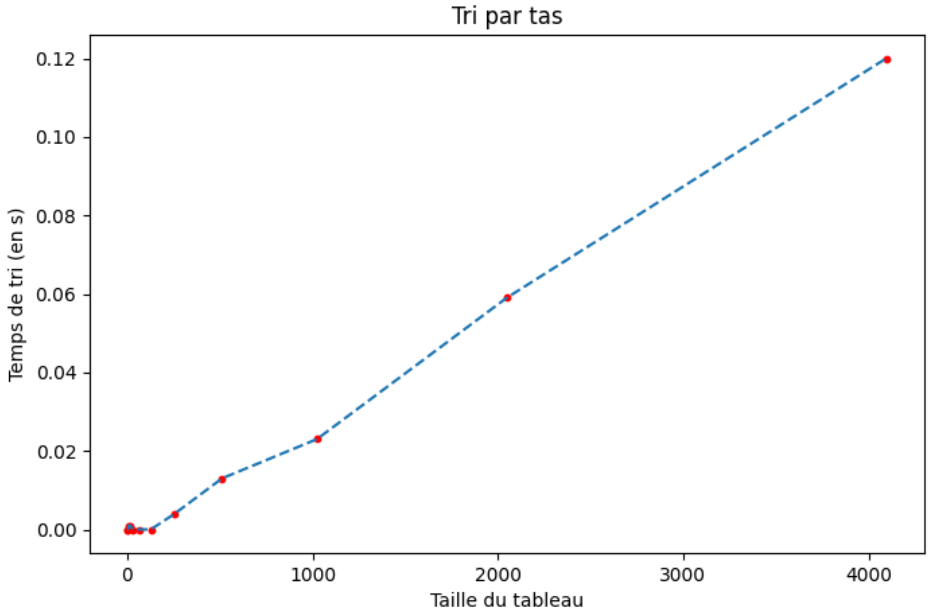


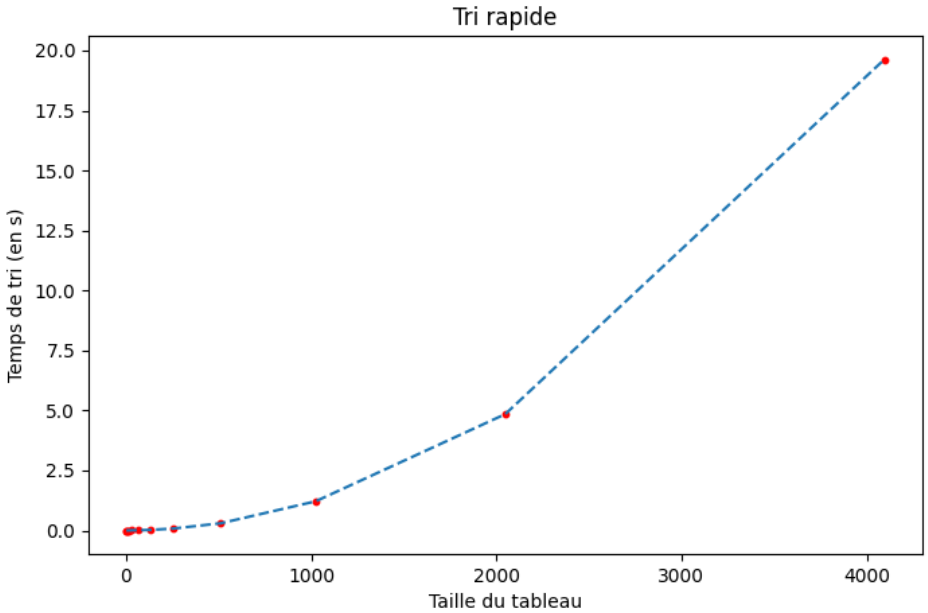



Tri fusion
Temps de tri (en s)
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1000
2000
3000
4000
Taille du tableau


Tri par compte
Temps de tri (en s)
0.011
0.010
0.009
0.008
0.007
0.006
0.005
0
1000
2000
3000
4000
Taille du tableau

4.2.5 Tris sur le jeu de données inversées avec redondance





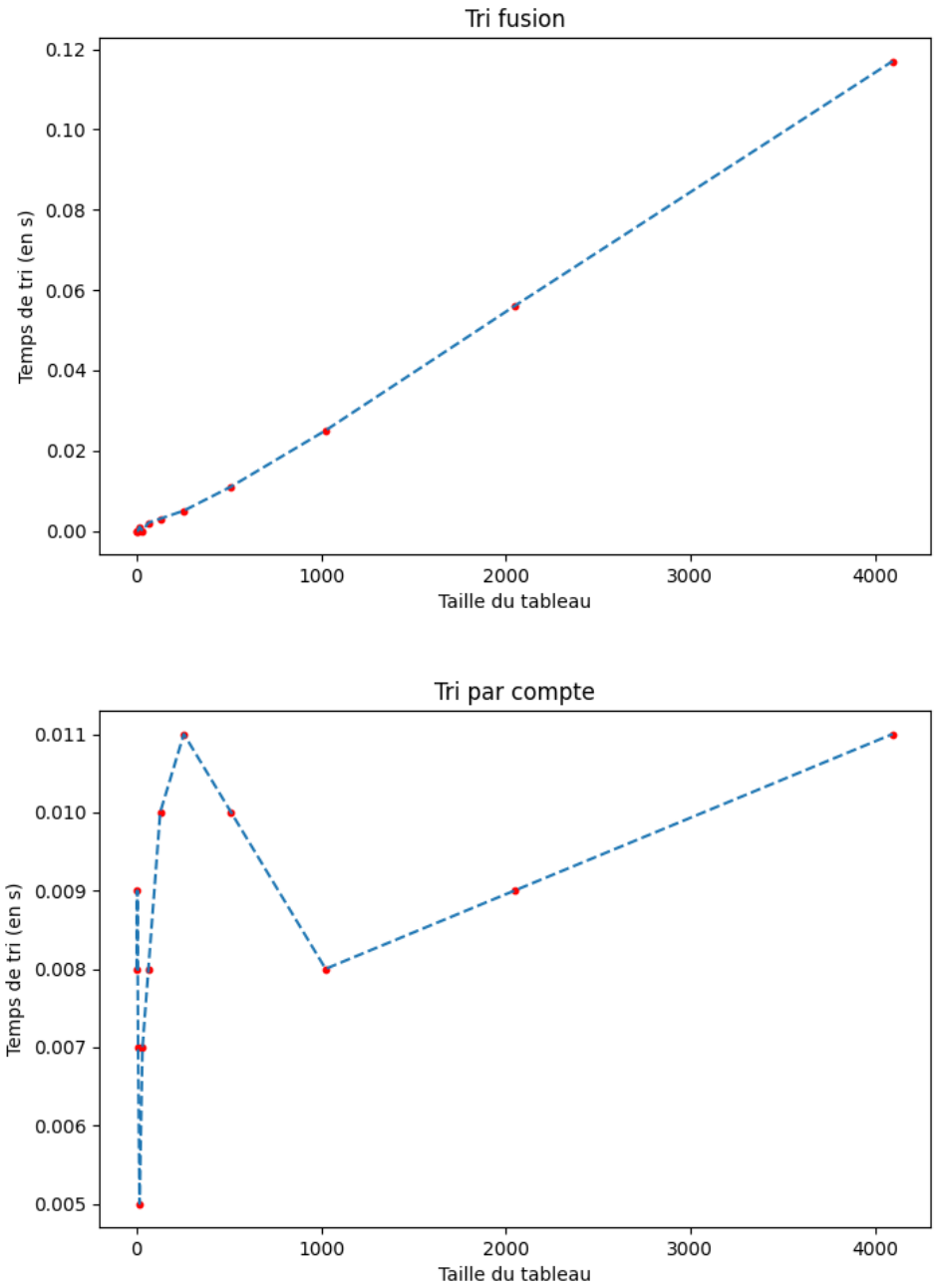


Figure 2 – Comparaisons des tris en sur les cinq jeux de données différents

On observe ici que le tri par compte est le plus efficace avec un vitesse d'exection toujours inférieure à 0.02 secondes. En effet

5 Conclusion