République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

TP Complexité

Spécialité
Systèmes informatique Intelligente

Titre

Rapport du TP n°02

Les Algorithmes de tri

Membres du quadrinome :

Mouaz Rayane Hamza	191931030386
Yousfi Zakaria	171732026950
ABDOU Maria Hind	161631069767
FERRADJI Tarek	181931058767

Tables des matieres

Introduction 6
Environment Experimental 7
Partie 01 : Développement de l'algorithme et du programme correspondant 7
Partie02 : Mesure du temps d'exécution. 8
1. La complexité de chaque algorithme : 8
Tri par selection: 8
Tri par insertion : 8
Tri à Bulle : 9
Tri rapide : 9
Tri par Fusion : 9
Tri par Tas : 10
3. Correspondance de meilleur, moyen et pire cas pour chaque Algorithme : 10
Tri par sélection : 10
Tri par insertion: 10
Tri à bulle 10
Tri rapide: 11
Tri fusion: 11
Tri par tas: 11
3. La mesure du temps d'exécution pour chaque algorithme : 11
Tri par sélection : 11
Tri par Insertion : 12
Tri à Bulle : 12
Tri Rapide (Pivot Médian) : 13
Tri Rapide (Pivot Droit) : 13
Tri Rapide (Pivot Gauche) : 14
Tri Fusion: 14
Tri par TAS: 15
4. Représentation des mesures par des graphes : 16
Tri par selection: 16
Tri par insertion: 16
Tri à Bulle : 17
Tri rapide (Pivot Médian) : 18
Tri rapide (Pivot Droit) : 18
Tri rapide (Pivot Gauche) : 19

Tri par Fusion : 19
Tri par Tas : 20
5. Modification des algorithmes : 20
Représentation des résultats par un tableau : 20
Représentation des résultats par un graphe : 22
Conclusion 26

Liste des figures

Figure 1 Graphe pour Tri par sélection 16
Figure 2 Graphe pour tri par insertion 16
Figure 3 Graphe pour tri à Bulle 17
Figure 4 Graphe pour tri rapide (Pivot Médian) 18
Figure 5 Graphe pour tri rapide (Pivot Droit) 18
Figure 6 Graphe pour tri rapide (Pivot Gauche) 19
Figure 7 Graphe pour tri Fusion. 19
Figure 8 Graphe pour tri par tas. 20
Figure 9 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en ordre inverse) 23
Figure 10 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en bon ordre) 24
Figure 11 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en ordre aléatoire) 25

Liste des tableaux

Tableau 1 Temps d'exécution/tri par sélection (ordre inverse)..... 11
Tableau 2 Temps d'exécution/tri par sélection (Bon ordre) 11
Tableau 3 Temps d'exécution/tri par sélection (ordre aléatoire) 12
Tableau 4 Temps d'exécution/tri par insertion (ordre inverse)..... 12
Tableau 5 Temps d'exécution/tri par insertion (Bon ordre)..... 12
Tableau 6 Temps d'exécution/tri par insertion (ordre aléatoire)..... 12
Tableau 7 Temps d'exécution/ tri à bulle (ordre inverse) 12
Tableau 8 Temps d'exécution/ tri à bulle (bon ordre)..... 12
Tableau 9 Temps d'exécution/ tri à bulle (ordre aléatoire)..... 12
Tableau 10 Temps d'exécution/ tri rapide (Pivot Médian) (ordre inverse)..... 13
Tableau 11 Temps d'exécution/ tri rapide (Pivot Médian) (bon ordre) 13
Tableau 12 Temps d'exécution/ tri rapide (Pivot Médian) (ordre aléatoire) 13
Tableau 13 Temps d'exécution/ tri rapide (Pivot Droit) (ordre inverse) 13
Tableau 14 Temps d'exécution/ tri rapide (Pivot Droit) (Bon ordre) 13
Tableau 15 Temps d'exécution/ tri rapide (Pivot Droit) (ordre aléatoire) 13
Tableau 16 Temps d'exécution/ tri rapide (Pivot Gauche) (ordre inverse)..... 14
Tableau 17 Temps d'exécution/ tri rapide (Pivot Gauche) (bon ordre) 14
Tableau 18 Temps d'exécution/ tri rapide (Pivot Gauche) (ordre aléatoire) 14
Tableau 19 Temps d'exécution/ tri Fusion (ordre inverse)..... 14
Tableau 20 Temps d'exécution/ tri Fusion (Bon ordre)..... 14
Tableau 21 Temps d'exécution/ tri Fusion (ordre aléatoire)..... 14
Tableau 22 Temps d'exécution/ tri par TAS (ordre inverse) 15
Tableau 23 Temps d'exécution/ tri par TAS (bon ordre)..... 15
Tableau 24 Temps d'exécution/ tri par TAS (ordre aléatoire)..... 15
Tableau 25 Le nombre de comparaison pour tous les algorithmes (données en ordre inversé)..... 20
Tableau 26 Le nombre de comparaison pour tous les algorithmes (données en bon ordre)..... 21
Tableau 27 Le nombre de comparaison pour tous les algorithmes (données en ordre aléatoire)..... 21

Introduction

Le tri est extrêmement important en informatique pour la même raison qu'il est important dans la vie de tous les jours. Il est plus facile et plus rapide de trouver des éléments dans une liste triée que non triée. Tri une liste d'éléments dans l'ordre croissant ou décroissant peut aider un humain ou un ordinateur trouvé rapidement des éléments sur cette liste.

La plupart des données que nous traitons, qu'il s'agisse de chiffres ou de texte, impliquent la saisie et la récupération de efficacement les informations pertinentes. Le tri devient alors crucial. Quand on a par exemple des centaines de valeurs comparables dans un tableau, il est souvent utile de les conserver dans un ordre trié pour une recherche ou une écriture ultérieure vers un fichier ou un rapport commandé. De même, la recherche ou la récupération de ces informations est réalisée d'une manière plus systématique et précise en utilisant les techniques de tri.

Plusieurs algorithmes de tri peuvent être utilisés dans un programme pour trier un tableau. Lors de la conception ou du choix un algorithme de tri, un objectif est de minimiser la quantité de travail nécessaire pour trier la liste des éléments. Là le coût du tri peut être mesuré par le nombre d'éléments du tableau qui doivent être comparés les uns aux autres, ou par le nombre de fois, deux éléments du tableau doivent être permutés. Dans certains cas, les comparaisons sont plus chères que les swaps ; dans d'autres cas, les échanges peuvent être plus coûteux que les comparaisons.

Dans ce projet, nous discuterons des algorithmes de tri standard suivants :

1. Tri par sélection.
2. Tri par insertion.
3. Tri à bulles.
4. Tri rapide.
5. Tri par fusion.
6. Tri par tas.

Environment Experimental

- *RAM: 8GO*
- *Processeur: Intel(R) CoreTM i5-3210M CPU @ 2.50 GHz.*
- *Windows 10-64 bits.*
- *Programming language: C.*
- *Environnement de développement intégré : Code-Blocks 16.04.*

Partie 01 : Développement de l'algorithme et du programme correspondant

L'implémentation de ces algorithmes ce trouve en annexes.

1. ***Tri par sélection :***

Le tri par sélection est un algorithme de tri simple qui est traité en divisant une liste en deux parties, la partie triée à l'extrémité gauche et la partie non triée à l'extrémité droite. Initialement, la partie triée est vide et la partie non triée est la liste complète. Le plus petit élément est sélectionné dans le tableau non trié et échangé avec l'élément le plus à gauche, et que l'élément devient une partie du tableau trié. Ce processus continue de déplacer la limite du tableau non trié d'un élément à droite.

2. ***Tri par Insertion :***

L'algorithme de tri par insertion est un algorithme de tri basé sur la comparaison sur place. Il maintient toujours un tri sous-liste dans les positions inférieures de la liste. Le tableau est parcouru séquentiellement et chaque nouvel élément non trié est puis réinséré à sa place dans la sous-liste précédente (dans le même tableau).

3. ***Tri à Bulle :***

L'algorithme de tri à bulles est un algorithme basé sur la comparaison qui effectue plusieurs passages dans une liste. Ce compare les éléments adjacents et échange ceux qui ne sont pas en ordre. Chaque passage dans la liste place le suivant la plus grande valeur à sa place. Essentiellement, chaque élément "bulle" jusqu'à l'endroit auquel il appartient.

4. ***Tri Rapide:***

Le tri rapide sélectionne un élément comme pivot et divise le tableau en deux tableaux. L'un d'eux contient les valeurs inférieures au pivot et l'autre tableau contient les plus grands. Il

existe de nombreuses versions différentes de tri rapide qui sélectionnent le pivot de différentes manières :

- Choisissez toujours l'élément le plus grand comme pivot.
- Choisissez toujours le plus petit élément comme pivot
- Choisissez la médiane comme pivot. (Un pivot optimal)

5. ***Tri Par Fusion :***

Le tri fusion suit la règle de diviser pour mieux régner. Cependant, il ne divise pas la liste en deux moitiés. Dans tri par fusion la liste non triée est divisée en N sous-listes, chacune ayant un élément, car une liste d'un élément est considéré comme trié. Ensuite, il fusionne à plusieurs reprises ces sous-listes, pour produire de nouvelles sous-listes triées, et enfin, une liste triée est produite.

6. ***Tri Par Tas :***

Le tri par tas est l'une des meilleures méthodes de tri en place et sans scénario quadratique du pire des cas. Tas L'algorithme de tri est divisé en deux parties fondamentales :

- Création d'un tas de la liste non triée.
- Triez le tableau en supprimant à plusieurs reprises le plus grand élément du tas et en l'insérant dans le tableau.

Le tas est reconstruit après chaque suppression.

Partie02 : Mesure du temps d'exécution.

---

1. La complexité de chaque algorithme :

Tri par selection:

L'algorithme divise le tableau en deux parties. La partie gauche est la partie triée et la partie droite est la partie non encore triée. L'algorithme va prendre le minimum de la partie non encore triée et elle le permute avec le nombre qui est le plus à gauche de cette partie, ensuite cet élément devient un élément de la partie triée et on avance vers l'élément suivant de la partie non triée. Initialement la partie trié et vide et la partie non triée contient tous les éléments du tableau donc on fait n-1 itération pour déterminer le premier minimum puis cet élément et mise dans la partie trié alors il reste n-1 élément dans la partie non triée donc il faut n-2 itération pour déterminer le minimum puis le troisième élément $(n-3)$... Jusqu'à les 2 derniers éléments ou on fait une seule itération et on termine. Donc en totale on a fait $(n-1) + (n-2) + \ldots + 2 + 1$ itérations qui est équivaut à la fameux somme des n nombres premier qui soit égale à $(n-1)(n-1+1)/2 = (n)(n-1)/2 = n\text{^}2/2 - n/2$ d'où la complexité est $O(n\text{^}2)$.

Tri par insertion :

L'algorithme passe par tous les éléments du tableau et pour chaque élément elle maintient un ordre triée des éléments précédents en faisant des permutations selon les valeurs des éléments. Donc il est évident

que le pire cas c'est quand le tableau est triée dans l'ordre inverse. Chaque élément est comparé avec tous les éléments à gauche avant l'insertion. Le 2eme élément est comparé avec le premier, le 3eme élément est comparé avec le premier et le 2eme... jusqu'au dernier élément qui est comparé avec les n-1 autres éléments d'où en tombe avec la même somme vu dans l'algorithme précédent.

Conclusion : la complexité est $O(n^2)$.

Tri à Bulle :

L'algorithme compare à chaque itération les éléments du tableau et elle fait la permutation dans le cas où l'élément droit est supérieur à l'élément gauche. Après l'arrivée à la fin du tableau on fait le traitement une autre fois mais cette fois on ignore le dernier élément c.-à-d. on considère que le tableau contient n-1 élément. Ça c'est parce que le plus grand élément a été mis dans la dernière position c'est pour ça on l'appelle tri à bulle, les grandes éléments sont comme des bulles qui flottent vers la fin du tableau. donc au pire cas si le tableau est triée dans l'ordre inverse l'algorithme va faire n-1 comparaison lors du premier passage, puis n-2 lors du deuxième...etc. d'où le nombre d'itération est égale à la somme des n-1 nombres premier vu dans les algorithmes précédents donc la complexité est $O(n^2)$.

Tri rapide :

L'algorithme prend un élément du tableau comme pivot. Après, on partitionne les éléments restants en deux tableaux disjoints tels que les éléments qui sont plus grands que le pivot sont positionnés après le pivot, et les éléments inférieur au pivot sont positionnés avant le pivot. Enfin, retourner le résultat de tri rapide du tableau des éléments inférieur au pivot, puis le pivot, puis le résultat de tri rapide du tableau des éléments supérieurs au pivot.
Le pire cas qui est très rare arrive quand la partition prend toujours le plus grand ou bien le plus petit élément des tableaux. Car si c'est le cas donc le tri est réduit à trié 2 listes, une qui contient 0 éléments et l'autre qui contient n-1 éléments, puis la liste précédente va être divisée en 2 liste une contient 0 éléments et l'autre n-2 et ainsi de suite. D'où la complexité est égale à la somme (n-1) + (n-2) + ... + 2 + 1. Comme on l'a vu dans les algorithmes précédents.
L'étape de partitionnement fait au minimum n-1 comparaison. Soit T(n) la complexité temporelle pour chaque prochaine étape pour n>=1 le nombre de comparaison est moins d'un 1 donc
$T(n) = T(n-1) + (n-1)$ avec $T(1) = 1$ alors

$$T(n) - T(n-1) = n-1$$

$$T(n) + T(n-1) + T(n-2) + \ldots + T(3) + T(2)$$

$$- T(n-1) - T(n-2) - \ldots - T(3) - T(2) - T(1)$$

$$= (n-1) + (n-2) + \ldots + 2 + 1 - 0$$ donc:

$T(n) = (n-1) + (n-2) + \ldots + 2 + 1 = (n-1)n/2$ => complexité $O(n^2)$

Tri par Fusion :

Le tri fusion divise le tableau en N tableaux différents puis elle les fusionne deux en deux jusqu'à l'arrivée à produire le tableau originale triée. C.-à-d. diviser le tableau jusqu'à qu'on a N éléments qui on peut considérer comme triée puis fusionner ses éléments en conservant le tri pour avoir le tableau finale triée. Au début on divise le tableau en deux sous tableaux de longueur n/2 puis ses tableaux sont encore divisés en des tableaux de longueur n/4 et ainsi de suite donc on peut imaginer un arbre binaire tel que le nombre de divisions est la longueur de cet arbre. Après la fin de la division on se trouve avec n éléments qui on fusionne deux a deux alors on fusionne $n * 1$ éléments puis à la prochaine itération en va fusionner $n/2 * 2$ éléments et ainsi de suite. Donc l'opération du fusionnement est linéaire avec une complexité de $O(n)$. pour chaque opération de

division il faut une opération de fusionnement et comme la hauteur d'un arbre binaire complet est égale à $\log(n)$ on déduit que la complexité de l'algorithme est égale à $O(n * \log(n))$.

Tri par Tas :

Définition d'un tas : c'est un arbre binaire équilibré stocké dans un tableau dans lequel chaque clé associée à un nœud interne est supérieure aux clés associées respectivement à ses fils s'ils existent.
On peut diviser l'algorithme en deux étapes :

1. Création d'un tas du tableau non triée
2. trier le tableau par l'enlèvement successif du plus grand élément du tas, et lui insérer dans le tableau. Le tas est reconstruit après chaque enlèvement.

On utilise trois procédures : buildHeap qui construit le tas, heapify qui arrange le tas pour maintenir sa propriété (voir définition tas) et Heapsort qui tri le tableau.
La procédure heapsort construite le tas on appelant la procédure buildHeap puis appelle n-1 fois la procédure heapify.
Dans la procédure heapify on traverse le tas du haut vers le bas, la hauteur d'un arbre binaire de taille n est égale à $O(\log n)$ donc la complexité de heapify est $O(\log n)$.
Pour construire le tas buildHeap va appeler heapify pour chaque nœud parent en débutant du dernier nœud et se terminant à la racine de l'arbre.
Un tas de taille n possède n/2 nœud parent. Comme la complexité de heapify est $O(\log(n))$ alors la complexité de buildHeap est $O(n \log(n))$.
La procédure heapify est appelée n-1 fois dans heapsort donc la complexité totale est égale $O(n \log((n)) + O(n \log(n)) \Rightarrow$ complexité $O(n \log(n))$

3. Correspondance de meilleur, moyen et pire cas pour chaque Algorithme :

Tri par sélection :

il faut traverser tous les éléments pour trouver le minimum (vu dans la question précédente) donc la complexité est toujours $O(n^2)$ pour le meilleur, moyen et pire cas.

Tri par insertion:

Le meilleur cas correspond lorsque le tableau est déjà trié. À une itération quelconque i, l'élément tab[i] est comparé avec l'élément le plus à droite de la partie trié, et comme le tableau est trié donc cet élément est inférieur à tab[i] alors on passe vers l'élément $i + 1$ d'où la complexité est égale au nombre des éléments O(n).
Le pire cas est égale à O(n^2) lorsque la liste est dans l'ordre inverse (vu dans la question précédent) donc le pire cas arrive quand pour chaque i la boucle intérieure va changer la position de tous les éléments A[1],...A[i-] (qui arrive lorsque A[i] est inferieur a tous ses éléments), ceci prend O(i-1) donc:
$T(n) = O(1) + O(2) + \ldots + O(n-1) = O(1 + 2 + 3 + \ldots + n - 1) = O(n(n-1)/2) = O(n^2)$.
Le moyen cas : pour ce cas, la boucle intérieure va insérer A[i] dans le milieu de $A[1], .A[i-1]$ et ceci prend $O(i/2)$ donc $T(n) = \sum_{i=1}^{n} O(n^2)$

Tri à bulle :

Le meilleur cas se produit lorsque le tableau est déjà trié. L'algorithme fait N comparaisons mais 0 permutations donc la complexité est O(n).
Le pire cas est lorsque le tableau sort est triée dans l'ordre inverse (vu dans la question précédente).
Le moyen cas se produit lorsqu'on fait n/2 passe et O(n) comparaison pour chaque passe donc la complexité du temps moyen pour cet algorithme est $O(n * n/2) \Rightarrow$ complexité O(n^2)

Tri rapide:

Pour le pire cas : (vu dans la question précédente)
Pour le meilleur cas : se réalise quand le pivot est l'élément médian du tableau. Les partitions nous donne un arbre binaire de hauteur log(n) et comme chaque partition prend n ou moins comparaison donc la complexité est $O(n \log(n))$ pour le cas moyen : dans le cas moyen on ne sait pas où les partitions se réalisent. Pour cette raison, on prend toutes les valeurs possibles du partitionnement et on ajoute toute leur complexité puis on divise par n pour avoir le cas complexité du cas moyen.

Tri fusion:

Comme on l'a vu dans la question précédente la complexité du tri fusion est toujours $O(n \log(n))$ car le tableau est divisé en 2 parties et ces parties sont résolues récursivement. Après avoir résolu les sous problèmes elles sont fusionnées en gardant l'ordre trié donc quel que soit l'ordre des éléments du tableau la complexité est la même.

Tri par tas:

Aussi comme on l'a vu dans la question précédente la complexité du tri par tas est toujours $O(n \log(n))$ quel que soit l'ordre des éléments du tableau.

3. La mesure du temps d'exécution pour chaque algorithme :

Tri par sélection :

i. Les données du tableau sont triées en ordre inverse.

Tableau 1 Temps d'exécution/tri par sélection (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.167	4.063	16.88	420.979	1640.003	/	/	/	/

ii. Les données du tableau sont triées en bon ordre.

Tableau 2 Temps d'exécution/tri par sélection (Bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.215	5.347	21.49	543.007996	1374.697	/	/	/	/

iii. Les données du tableau ne sont pas triées.

Tableau 3 Temps d'exécution/tri par sélection (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.142	3.317	13.432	335.939	1564.792	/	/	/	/

Tri par Insertion :

i. Les données du tableau sont triées en ordre inverse.

Tableau 4 Temps d'exécution/tri par insertion (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.138	6.736	20.947001	554.138	1640.003	/	/	/	/

ii. Les données du tableau sont triées en bon ordre.

Tableau 5 Temps d'exécution/tri par insertion (Bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.00	0.00	0.00	0.001	1374.697	0.007	0.031	/	/

iii. Les données du tableau ne sont pas triées

Tableau 6 Temps d'exécution/tri par insertion (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.198	2.267	9.877	264.88	1564.792	/	/	/	/

Tri à Bulle :

i. Les données du tableau sont triées en ordre inverse.

Tableau 7 Temps d'exécution/ tri à bulle (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.462	9.792	37.174	/	/	/	/	/	/

ii. Les données du tableau sont triées en bon ordre.

Tableau 8 Temps d'exécution/ tri à bulle (bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.00	0.00	0.00	0.0001	0.004	0.004	0.007	0.008	/

iii. Les données du tableau ne sont pas triées.

Tableau 9 Temps d'exécution/ tri à bulle (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8

Temps d'exécution	0.405	12.703	58.788	/	/	/	/	/	/

Tri Rapide (Pivot Médian) :

i. Les données du tableau sont triées en ordre inverse.

Tableau 10 Temps d'exécution/ tri rapide (Pivot Médian) (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.001	0.002	0.005	0.026	0.055	0.299	0.621	3.521	7.203

ii. Les données du tableau sont triées en bon ordre.

Tableau 11 Temps d'exécution/ tri rapide (Pivot Médian) (bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.00	0.002	0.007	0.026	0.058	0.318	0.67	3.624	7.347

iii. Les données du tableau ne sont pas triées

Tableau 12 Temps d'exécution/ tri rapide (Pivot Médian) (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.0	0.006	0.013	0.078	0.158	0.858	1.977	11.009	36.558

Tri Rapide (Pivot Droit) :

i. Les données du tableau sont triées en ordre inverse.

Tableau 13 Temps d'exécution/ tri rapide (Pivot Droit) (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.139								

ii. Les données du tableau sont triées en bon ordre.

Tableau 14 Temps d'exécution/ tri rapide (Pivot Droit) (Bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.142								

iii. Les données du tableau ne sont pas triées.

Tableau 15 Temps d'exécution/ tri rapide (Pivot Droit) (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8

Temps d'exécution	0.242								

Tri Rapide (Pivot Gauche) :

i. Les données du tableau sont triées en ordre inverse.

Tableau 16 Temps d'exécution/ tri rapide (Pivot Gauche) (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.133								

ii. Les données du tableau sont triées en bon ordre.

Tableau 17 Temps d'exécution/ tri rapide (Pivot Gauche) (bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.057								

iii. Les données du tableau ne sont pas triées

Tableau 18 Temps d'exécution/ tri rapide (Pivot Gauche) (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.001								

Tri Fusion:

i. Les données du tableau sont triées en ordre inverse.

Tableau 19 Temps d'exécution/ tri Fusion (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.001	0.008	0.088	0.105					

ii. Les données du tableau sont triées en bon ordre.

Tableau 20 Temps d'exécution/ tri Fusion (Bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.002	0.029	0.024	0.104					

iii. Les données du tableau ne sont pas triées.

Tableau 21 Temps d'exécution/ tri Fusion (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8

Temps d'exécution	0.003	0.026	0.034	0.145					

Tri par TAS:

i. Les données du tableau sont triées en ordre inverse.

Tableau 22 Temps d'exécution/ tri par TAS (ordre inverse)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.003	0.035	0.068	0.301	0.441	1.887	4.098	19.935	44.03

ii. Les données du tableau sont triées en bon ordre.

Tableau 23 Temps d'exécution/ tri par TAS (bon ordre)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.002	0.052	0.102	0.224	0.366	1.932	3.791	21.018	43.788

iii. Les données du tableau ne sont pas triées.

Tableau 24 Temps d'exécution/ tri par TAS (ordre aléatoire)

Taille	10^4	5*10^4	10^5	5*10^5	10^6	5*10^6	10^7	5*10^7	10^8
Temps d'exécution	0.003	0.096	0.057	0.374	0.718	3.544	8.240	48.294	108.433

4. Représentation des mesures par des graphes :

Tri par selection:

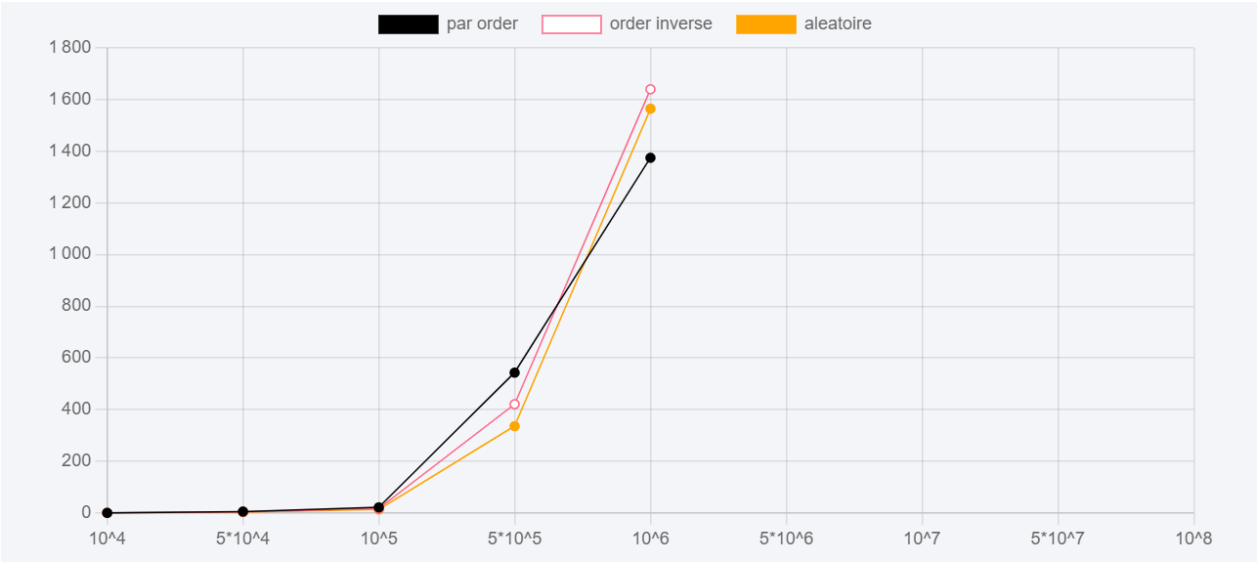

Figure 1 Graphe pour Tri par sélection

Analyse :

Le graphe montre clairement que le tri par sélection est affecté par la taille du tableau. Nous constatons que cela prend trop de temps pour trier les tableaux lorsque la taille dépasse **10^6** peu importe l'ordre des données du tableau(en bon ordre, en ordre inverse, en ordre aléatoire). Ces résultats sont en accord total avec la théorie et confirment que cet algorithme n'est pas adapté aux données volumineuses.

Tri par insertion:

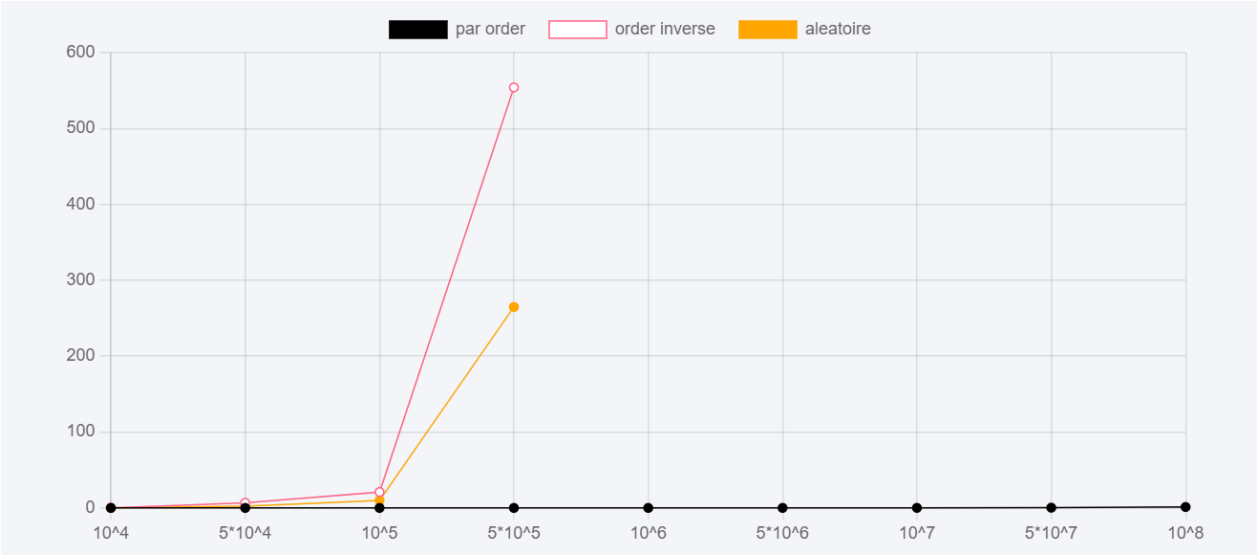

Figure 2 Graphe pour tri par insertion

Analyse :

Les résultats sont proches de ceux du tri par sélection. Le graphe obtenu révèle que l'algorithme de tri par insertion n'est pas efficace pour les grands ensembles de données car le temps d'exécution augmentait rapidement lorsque la taille du tableau atteignait **5*10^5** pour les cas où les données du tableau sont triées en ordre inverse ou en ordre aléatoire. Mais dans le cas où les données sont triées en bon ordre en remarque que le temps d'exécution est toujours à **0 seconde** quel que soit la taille du tableau.

Ceci est cohérent avec la complexité quadratique de ce type dans le pire des cas.

Tri à Bulle :

Figure 3 Graphe pour tri à Bulle

Analyse :

Le graphe montre que le temps d'exécution a augmenté très rapidement lorsque la taille du tableau a augmenté dans les deux cas (données du tableau sont en ordre inverse ou en ordre aléatoire), et par contre si les données sont en bon ordre l'exécution ne prendre aucun temps et il prend **0 seconde** pour n'importe quelle taille du tableau.

Donc il est clair que le tri à bulles est assez inefficace pour trier des tableaux volumineux.

Tri rapide (Pivot Médian) :

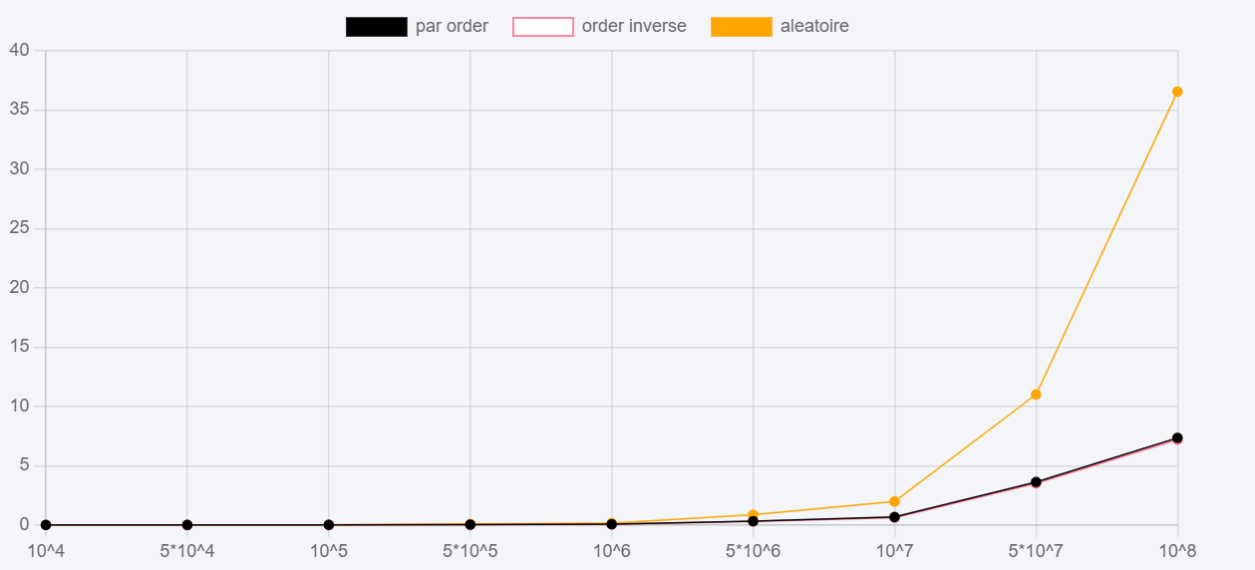

Figure 4 Graphe pour tri rapide (Pivot Médian)

Analyse :

En analysant le graphe, on voit clairement que le temps d'exécution est très faible même pour des tableaux de grandes tailles. Donc le tri rapide lorsque le pivot est le médian est très efficace pour trier les tableaux même pour de grande taille.

Tri rapide (Pivot Droit) :

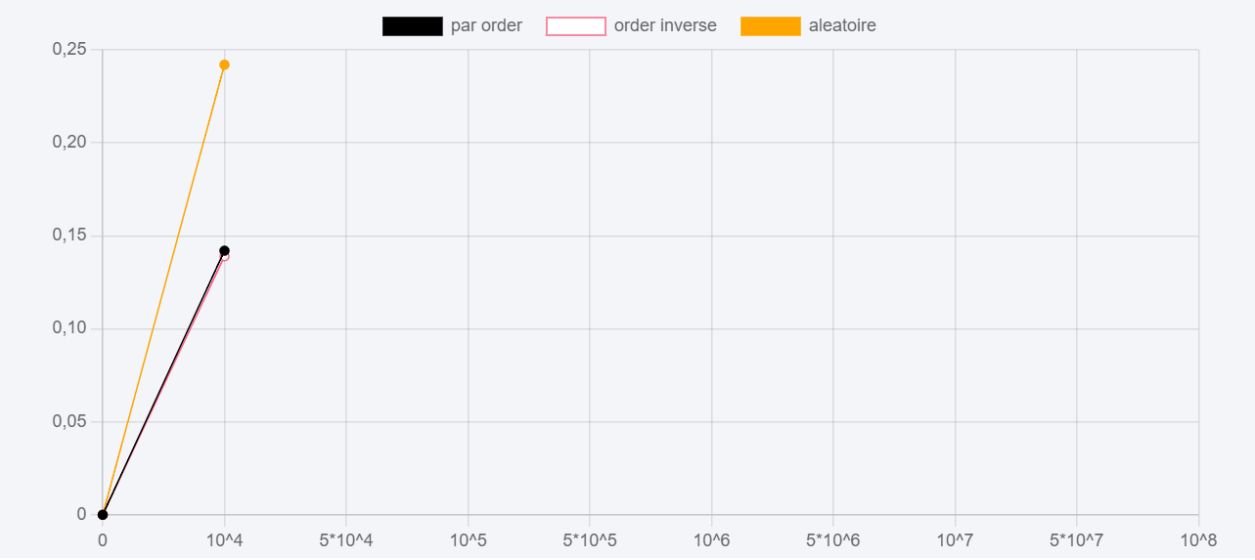

Figure 5 Graphe pour tri rapide (Pivot Droit)

Tri rapide (Pivot Gauche) :

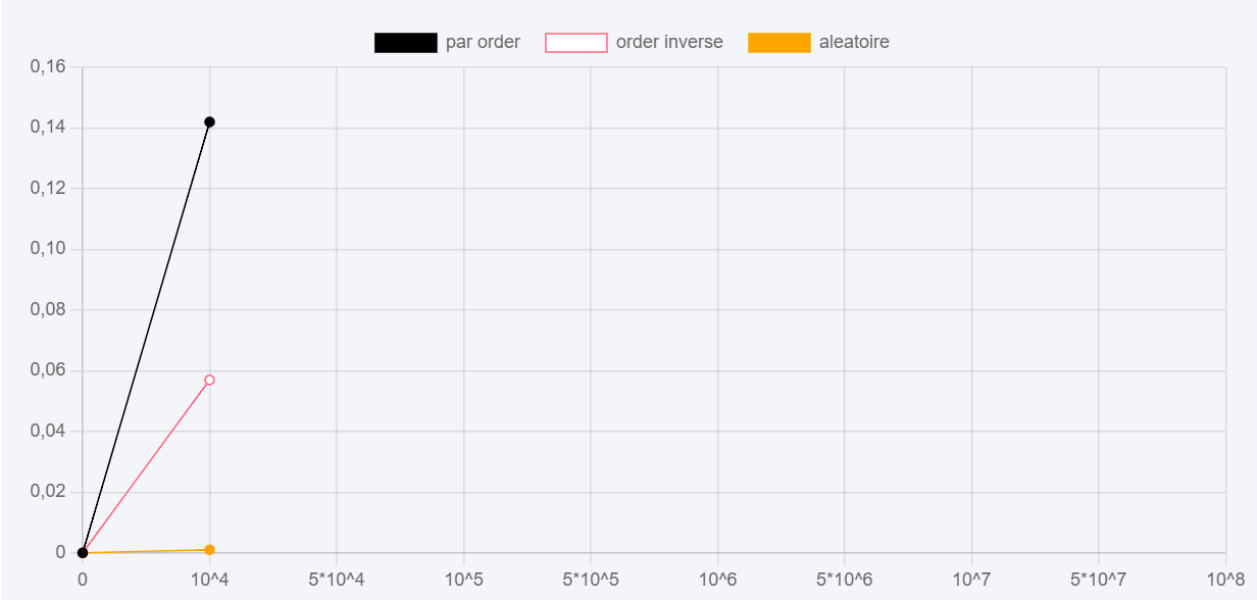

Figure 6 Graphe pour tri rapide (Pivot Gauche)

Tri par Fusion :

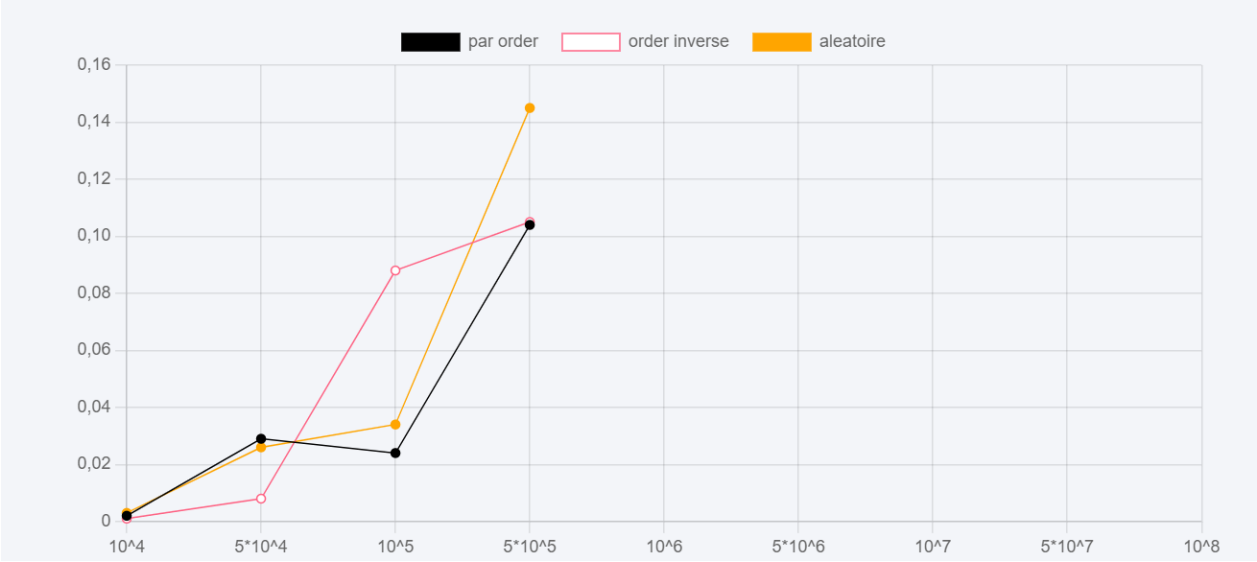

Figure 7 Graphe pour tri Fusion.

Analyse :

Comme le montre le graphe, le temps d'exécution reste très faible quel que soit la manière ou les données du tableau sont triées (bon ordre, ordre inverse, ordre aléatoire), ce qui confirme la logarithmique complexité du tri par fusion. Cette méthode de tri est très utile, car elle est efficace en termes de temps.

Tri par Tas :

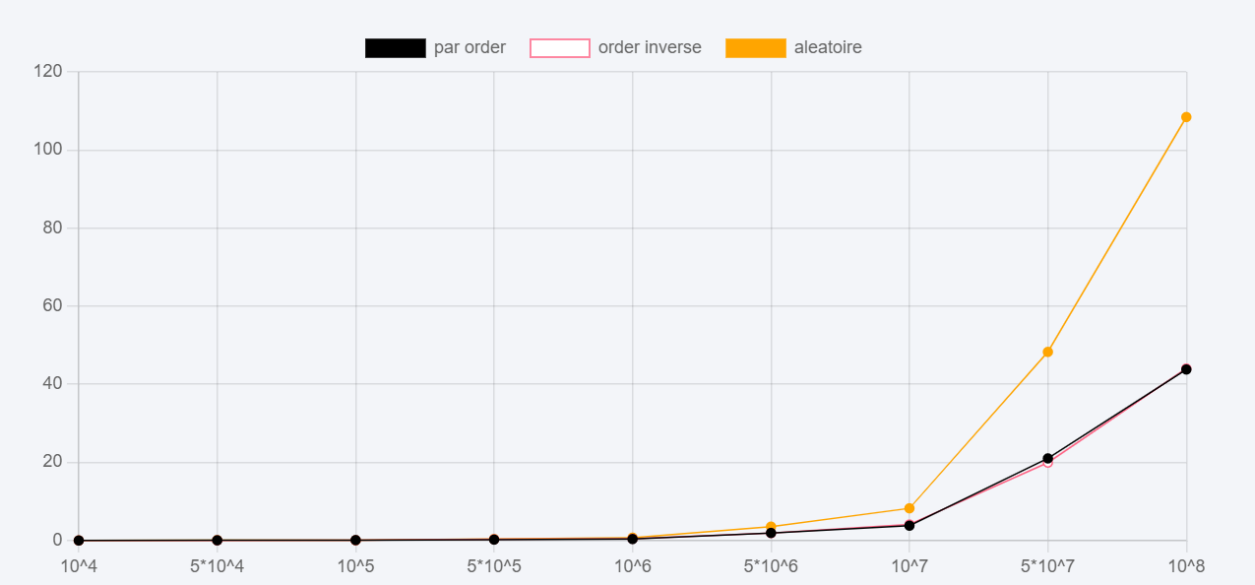

Figure 8 Graphe pour tri par tas.

Analyse :

Le temps d'exécution, pour des ensembles de données moyens (moins de **5*10^6**), est presque **zéro seconde**. L'augmentation de la taille des tableaux (plus que **10^7**) a conduit à une augmentation du temps d'exécution, mais ce dernier est faible et négligeable dans le cas où les données tu tableau sont en bon ordre ou en ordre inverse, et beaucoup plus augmenté dans le cas où ils sont aléatoires.

Ce résultat cadre parfaitement avec les estimations et correspond à la complexité **n Log(n).**

5. Modification des algorithmes :

-Se trouve dans un fichier en annexe.

Représentation des résultats par un tableau :

i. Les données du tableau sont triées en ordre inverse.

Tableau 25 Le nombre de comparaison pour tous les algorithmes (données en ordre inversé)

Taille du fichier	10^4	5*10^4	10^5	5*10^5
Tri par sélection	49995000	1249975000	704982704	445698416
Tri par insertion	49995000	1249975000	704982704	445698416
Tri à Bulle	49995000	1249975000	704982704	445698416
Tri rapide	50019997	/	/	/

(pivot gauche)				
Tri rapide (pivot médian)	141341	832780	1765549	9762159
Tri rapide (pivot droit)	50024998	/	/	/
Tri par Fusion	64608	382512	815024	4692496
Tri par tas	246706	1464781	3131749	17966008

ii. Les données du tableau sont triées en bon ordre.

Tableau 26 Le nombre de comparaison pour tous les algorithmes (données en bon ordre)

Taille du fichier	10^4	5*10^4	10^5	5*10^5
Tri par sélection	0	0	0	0
Tri par insertion	0	0	0	0
Tri à Bulle	0	0	0	0
Tri rapide (pivot gauche)	50019997	/	/	/
Tri rapide (pivot médian)	131343	782782	1665551	9262161
Tri rapide (pivot droit)	50024998	/	/	/
Tri par Fusion	69008	401952	853904	4783216
Tri par tas	264514	1556241	3313399	18845977

iii. Les données du tableau sont triées en ordre aléatoire.

Tableau 27 Le nombre de comparaison pour tous les algorithmes (données en ordre aléatoire)

Taille du fichier	10^4	5*10^4	10^5

Tri par sélection	25095926	624356333	1796255215
Tri par insertion	24995736	625100751	1797818005
Tri à Bulle	25077861	623358006	1794362743
Tri rapide (pivot gauche)	217805	/	/
Tri rapide (pivot médian)	199060	1199735	2663376
Tri rapide (pivot droit)	65093956	/	/
Tri par Fusion	120412	718249	1536282
Tri par tas	255208	1509511	3218972——

Représentation des résultats par un graphe :

i. Les données du tableau sont triées en ordre inverse.

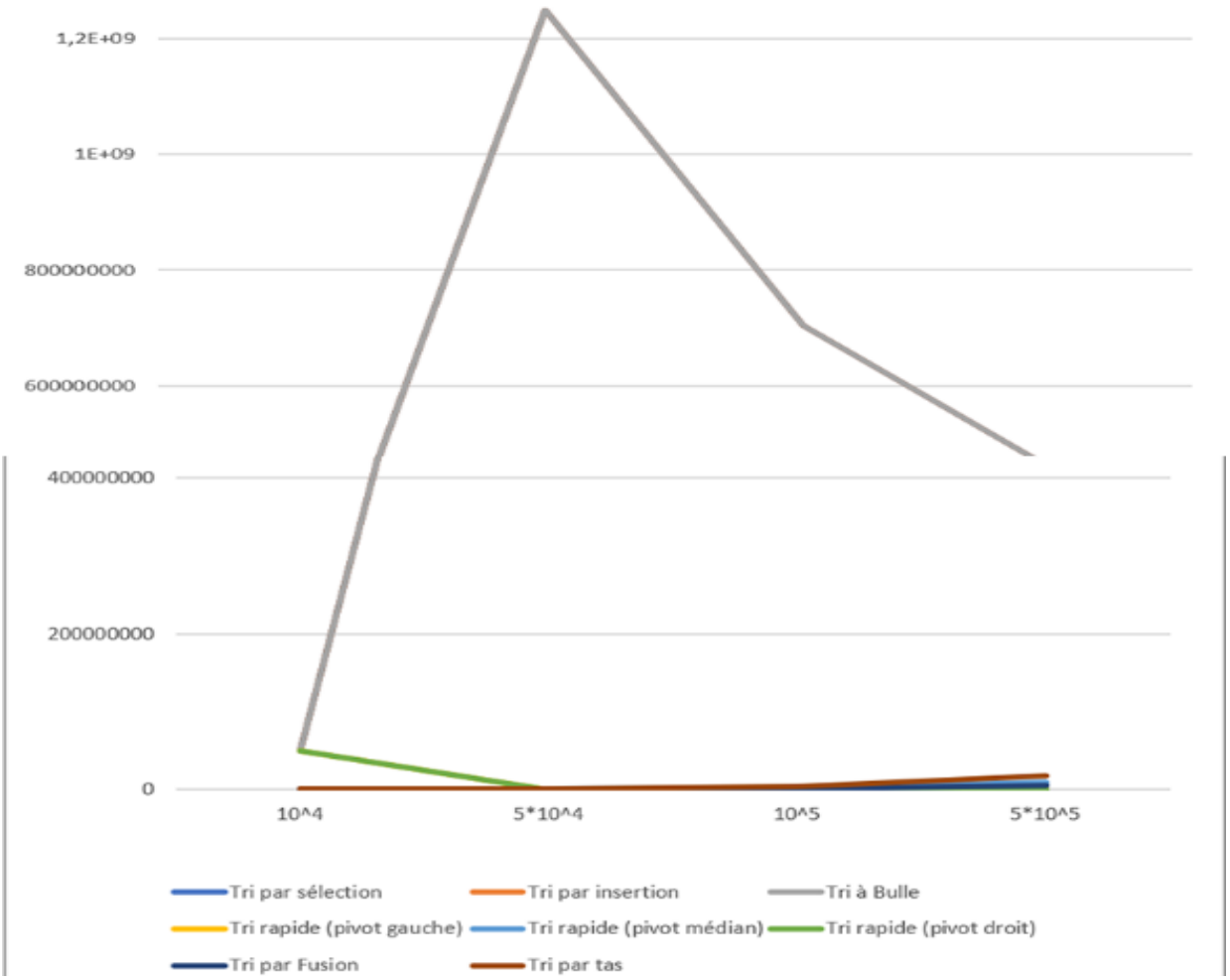

Figure 9 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en ordre inverse)

Analyse :

On remarque que tous les algorithmes de tri font un nombre de comparaison proche de zéro, sauf le tri à bulle qui fait un nombre de comparaison très grand.

ii. Les données du tableau sont triées en bon ordre.

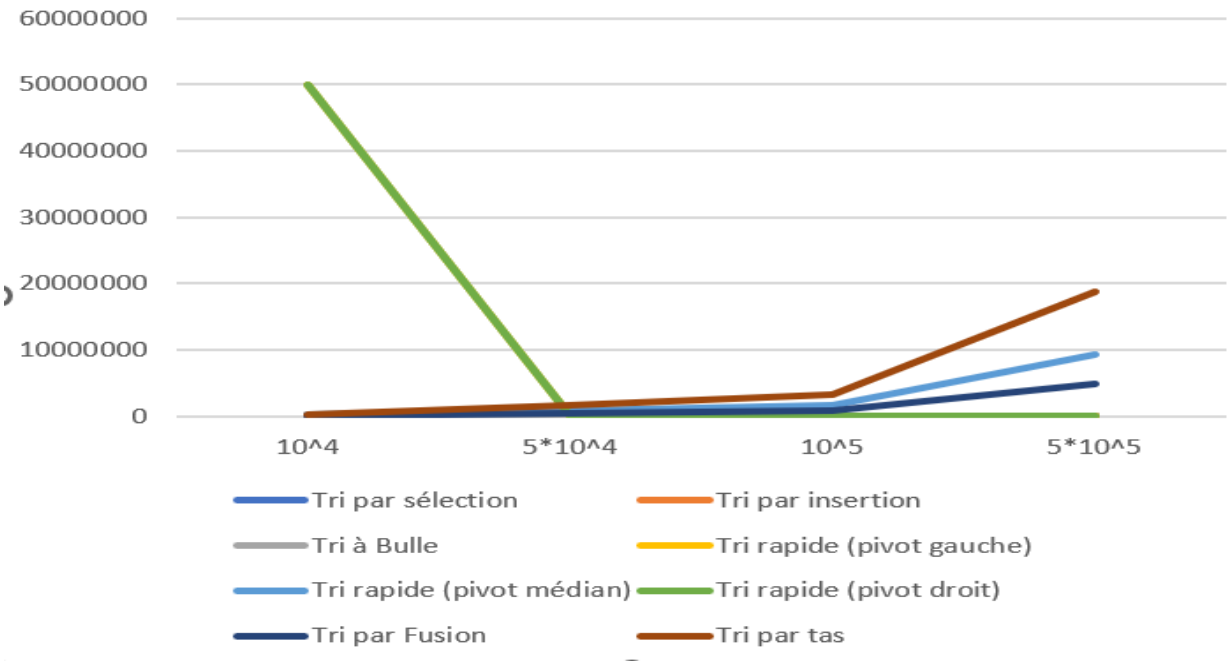

Figure 10 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en bon ordre)

Analyse :

On remarque que les trois algorithmes de tri (tri par sélection, tri par insertion, tri à bulle), et les trois algorithmes de tri (tri par tas, tri rapide (pivot médian), tri par fusion) font un nombre de comparaison croissant par rapport la taille de fichier des données dans le cas où les données sont en bon ordre, et pour le tri rapide pivot (droit) le nombre de comparaison et décroissant jusqu'à ce qu'il devient zéro quand la taille de fichiers dépasse 5*10^4.

iii. Les données du tableau sont triées en ordre aléatoire.

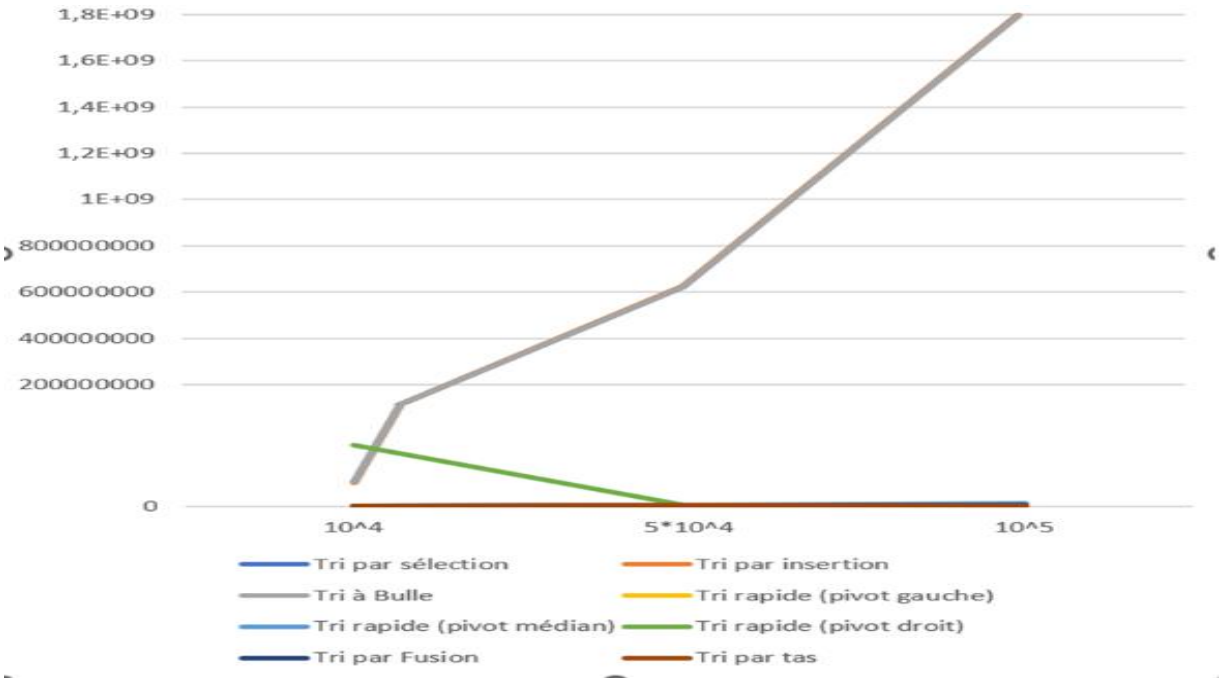

Figure 11 Graphe qui montre le nombre de comparaison pour chaque algorithme(cas ou les données sont en ordre aléatoire)

Analyse :

Dans ce cas on remarque que le tri à bulle fait un nombre de comparaison très grand quand la taille de fichier augmente, et les autres algorithmes font un nombre de comparaison proche de zéro même si la taille de fichier augmente.

Conclusion

La plupart des algorithmes de tri fonctionnent en comparant les données triées. Ils sont généralement jugés par leur efficacité. Dans ce cas, l'efficacité fait référence à l'efficacité algorithmique comme la taille de là l'entrée devient volumineuse et est généralement basée sur le nombre d'éléments à trier. La plupart les algorithmes utilisés ont une efficacité algorithmique de $O(n\text{^}2)$ ou $O(n \log(n))$.

De nombreux algorithmes qui ont la même efficacité n'ont pas la même vitesse sur la même entrée. Tout d'abord, les algorithmes doivent être jugés en fonction de leur efficacité moyenne, dans le meilleur des cas et dans le pire des cas. Certains algorithmes, tels que tri rapide, fonctionne exceptionnellement bien pour certaines entrées, mais horriblement pour d'autres. D'autres algorithmes, tels que la fusion sort, ne sont pas affectés par l'ordre des données d'entrée.

Par conséquent, pour trier une liste d'éléments, nous avons d'abord analysé le problème donné : le problème est de quel type (Petits nombres, grandes valeurs, grande taille...). Après cela, nous appliquons l'algorithme de tri en gardant à l'esprit le Complexité minimale, comparaison minimale et vitesse maximale.

Il est très important de connaître les avantages et les inconvénients des techniques de tri pour choisir le meilleur tri algorithme pour le problème donné.

Enfin, il est intéressant de noter que la complexité temporelle n'est pas le seul critère d'efficacité. Espace requis et la stabilité ont également un rôle important à jouer dans le choix de la meilleure méthode de tri.