Algorithmique et programmation 4

RAPPORT SUR LA COMPLEXITE DES ALGORITHMES DE TRI

Université du Havre Normandie

2023/2024

Réalisé par: BENGRAB MAHA
HALMI ILIAS

Licence 2 : Maths-Info

La complexité des algorithmes de tri

BENGRAB MAHA
HALMI ILIAS

TABLE DES MATIERES

BENGRAB MAHA
HALMI ILIAS

Présentation du rapport :

Dans le vaste domaine de l'algorithmique et de la programmation, les algorithmes de tri sont des opérations qu'on utilise très souvent pour faciliter plusieurs taches. Que ce soit pour organiser des données, accélérer les recherches ou faciliter le traitement de l'information.

Ce rapport s'attache donc à explorer en profondeur cette notion fondamentale, en examinant 3 différents types de tris : le tri par insertion, le tri fusion et le tri par dénombrement. Nous ferons une étude de leurs mécanismes sous-jacents, leurs complexités qui mettront en évidence leurs avantages et leurs limitations.

En déployant une approche analytique et pratique, nous plongerons dans l'univers fascinant des algorithmes de tri, en mettant en lumière leur pertinence et leur utilité dans le contexte de la programmation moderne et en illustrant les différents points importants qui marquent la différence entre les 3 types de tris.

Plan du rapport :

⇒ Tri par insertion / Tri fusion / Tri par dénombrement

- Définition Générale et principe
- Algorithme
- Complexité
- Avantages et limitations en fonction de la complexité du tri
- Analyse du graphe représentatif
- Bilan général

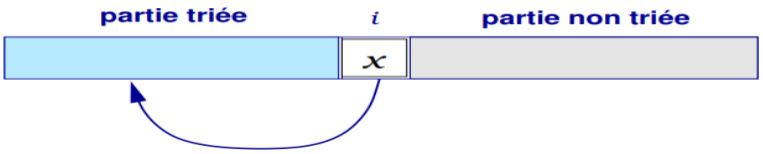
TRI PAR INSERTION

1. DEFINITION GENERALE :

Le tri par insertion est un algorithme de tri simple et efficace où chaque élément de la liste est inséré à sa place correcte parmi les éléments déjà triés, en comparant et en déplaçant les éléments au besoin. C'est un processus itératif qui fonctionne progressivement en construisant une liste triée élément par élément.

2. PRINCIPE :

L'idée est de trier successivement les premiers éléments du tableau. À chaque étape, l'algorithme prend un élément non trié de la liste et le place à sa position appropriée dans la partie triée de la liste c'est-à-dire on insère le **i-ème** élément à traiter à son rang parmi les **i-1** éléments précédents qui ont déjà été traités et triés entre eux.



L'algorithme répète ce processus jusqu'à ce que tous les éléments soient triés en suivant les étapes suivantes :

- Parcourir le tableau jusqu'au dernier élément en commençant par le **2ème** élément du tableau. (L'élément à l'indice **1** mais pas **0**)

- Comparer pour chaque élément, sa clé avec les clés des éléments précédents dans la partie triée de la liste, si la clé courante est plus petite que les précédentes, on effectue un changement pour insérer l'élément à sa bonne position dans la partie triée, en déplaçant les éléments plus grands d'une position vers la droite.

- Répéter les mêmes étapes tant que l'indice de l'élément courant est supérieur à **0** et que sa clé est inférieure à celle de l'élément précédent. Dans ce cas, l'élément est déplacé vers la gauche dans la liste triée. Sinon, l'élément reste à sa place actuelle.

3. ALGORITHME :

Entrée : n : entier **; T** un tableau de **n** éléments

Sortie : T le tableau trié des éléments

Variables locales : i, pos : entier ; **temp :** élément

```
début
    pour i = 2 jusqu'à n faire
        temp ← T[i] ;
        pos ← chercherPosition(T, 1, i-1, temp);
        pour j = i → pos + 1 faire T[j]← T[j-1]
        T[pos]← temp ;
        fin faire
    fin faire
fin
```

4. COMPLEXITE DU TRI PAR INSERTION :

- ANALYSE SUCCINCTE DE LA COMPLEXITE TEMPORELLE :
  - **Meilleur cas : O(n)**
  - **Cas moyen : O(n^2)**
  - **Pire cas : O(n^2)**

  Où **n** représente le nombre d'éléments dans la liste à trier.

- ÉTUDE THEORIQUE DE LA COMPLEXITE TEMPORELLE :
  - **Complexité au meilleur :**

Dans le meilleur cas du tri par insertion, la complexité reste linéaire. Lorsque le tableau est déjà parfaitement trié ou presque trié, il y aura toujours **n - 1** comparaisons à effectuer et au plus **n** affectations. Cela est dû au fait que chaque nouvel élément doit être comparé avec les éléments précédents pour déterminer sa position correcte dans la séquence triée. Puisque chaque comparaison est suivie d'une insertion dans la séquence triée, le nombre total d'opérations pour trier le tableau est proportionnel au nombre d'éléments, donnant une complexité temporelle au mieux de ***O(n)*.**

  - **Complexité en moyenne :**

Lors de l'insertion d'un élément dans un tableau trié, on doit le comparer à chaque élément jusqu'à trouver la position appropriée. Ainsi en moyenne, pour insérer un élément on doit le comparer à environ la moitié des éléments déjà triés.

Alors lorsqu'on insère le **i -ème** élément, il y a **i-1** éléments déjà triés. Chacun de ses éléments a une probabilité de ½ d'être plus grand ou plus petit que le **i -ème** élément, donc le nombre d'inversions en moyen est $\boldsymbol{i - \frac{1}{2}}$.

En sommant sur i on obtient :

$$\sum_{i=2}^{n} \frac{i-1}{2} = \frac{1}{2}\sum_{i=2}^{n}(i-1) = \frac{1}{2}\left(1 + 2 + \cdots + (n-1)\right) = \frac{1}{2}\frac{(n-1)n}{2} = \frac{n^2-n}{4}$$

Ainsi la complexité en moyenne du tri par insertion est $\frac{n^2-n}{4}$, qui est équivalent à

$\frac{n^2}{4} + O(n)$.

Pour conclure la complexité du tri par insertion en moyenne est $O(n^2)$.

- **Complexité au pire :**

Le tri par insertion a une complexité temporelle de $O(n^2)$ dans le pire des cas. Cela est dû aux conditions de la liste à trier, qui est forcément trié à l'envers, c'est à dire elle est classée de manière décroissante alors qu'on souhaite la trier d'une façon croissante ou vice versa. Pour réaliser cette opération dans ce cas, il faut donc décaler les **(p-1)** éléments qui précédent le **p-ème** élément qu'on souhaite insérer.

L'algorithme effectue de l'ordre de $n^2/2$ affectations et comparaisons.

Soit une complexité totale, dans le pire des cas, donnée par la formule suivante :

$$T(n) = \sum_{i=1}^{n-1} i = 1 + 2 + 3 + \cdots + (n-1)$$

Où **T(n)** représente le nombre total d'opérations effectuées pour trier une liste de taille **n** dans le pire des cas.

5. La complexite sous les projecteurs : Devoilement des Avantages et Limitations du tri

- Avantage :
  - **Complexité linéaire dans le meilleur des cas** : Lorsque la taille des données est petite ou lorsque la liste est déjà partiellement triée, le tri par insertion peut être plus efficace que d'autres algorithmes de tri plus complexes car sa complexité temporelle moyenne qui est plus proche de **O(n)** dans ce genre de cas lui permet d'effectuer un nombre minimal de comparaisons et de déplacements.

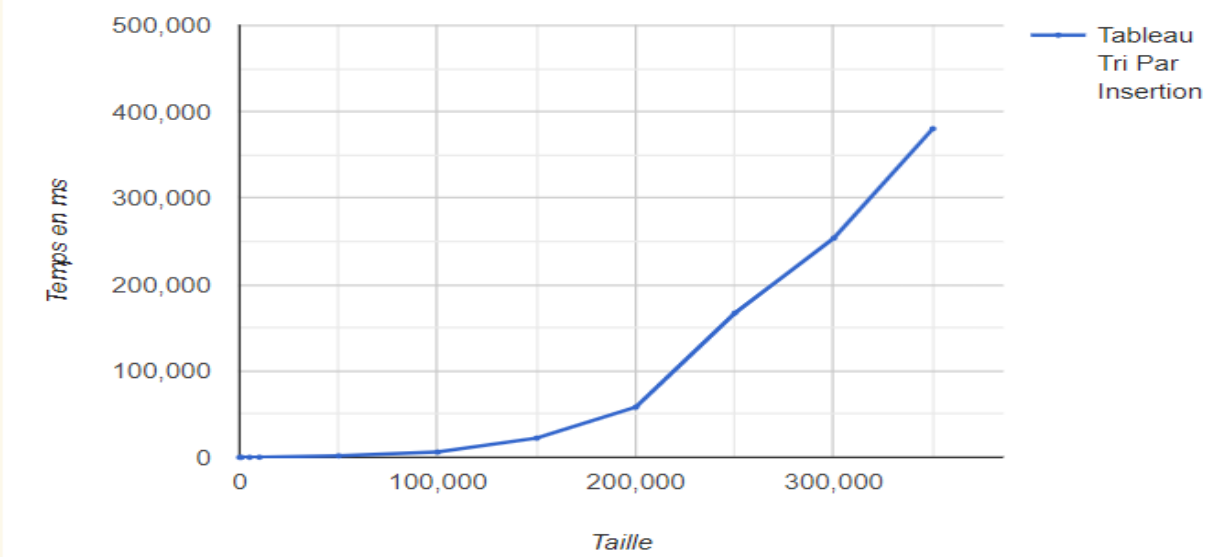
- Limitation :
  - **Complexité quadratique dans le pire des cas** : La complexité temporelle du tri par insertion est quadratique dans le pire des cas. Cela signifie que le temps d'exécution peut être très long pour de grandes listes non triées et fait preuve de l'inefficacité de ce tri pour de très grandes quantités.

6. ANALYSE DU GRAPHE REPRESENTATIF

- TABLEAU REPRESENTANT LA TAILLE DU TABLEAU ET LE TEMPS D'EXECUTION :

Taille	1000	5000	10000	50000	100000	150000	200000	250000	300000	350000
Temps en ms	7	24	149	1742	6189	22305	58149	167087	253784	380471

- LA COURBE :



- INTERPRETATION DE LA COURBE :

D'après ce graphique, on peut clairement remarquer que plus la taille du tableau augmente, plus le temps d'exécution est long. La courbe a subi une augmentation très intéressante allant d'une valeur de 7 ms pour une taille de 1000 à une valeur de 380471 ms pour une taille de 350000. Cette courbe observée confirme donc de manière tangible les propriétés mentionnées précédemment concernant le tri par insertion.

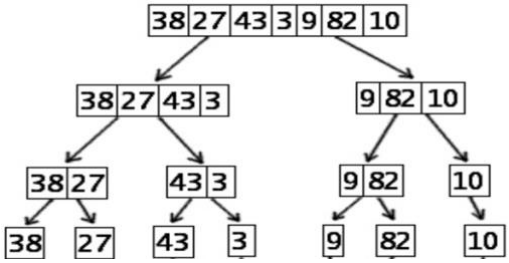
TRI FUSION

1. DEFINITION ET PRINCIPE :

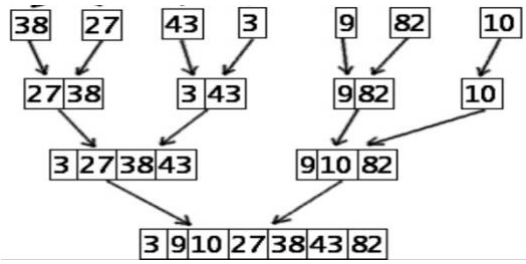
Le tri fusion est un algorithme de tri récursif basé sur la technique de diviser pour régner. Il divise une liste en deux moitiés, trie chaque moitié individuellement, puis fusionne les deux parties triées pour obtenir une liste complètement triée.

Le principe de base des étapes stratégiques du tri fusion est que pour résoudre un gros problème, il est préférable de le diviser en petits problèmes élémentaires. Une fois chaque petit problème est résolu, il n'y a plus qu'à combiner sa solution aux autres pour résoudre le problème global.

- **La première étape (La division) :** elle s'agit de diviser le tableau en deux, cette étape consiste à diviser récursivement le tableau initial en sous-tableaux plus petits jusqu'à ce que chaque sous-tableau contienne un seul élément. Cela se fait en continuant à diviser chaque sous-tableau en deux parties égales jusqu'à ce qu'ils ne puissent plus être divisés comme il est clarifié dans l'image ci-dessous.



- **La deuxième étape (La fusion) :** elle compte à opérer des fusions entre les tableaux qui ont été divisés de telle façon qu'on obtient un tableau trié au final. Une fois que chaque sous-tableau contient un seul élément, on passe à fusionner ces sous-tableaux en paires, puis fusionner les paires pour former des tableaux triés plus grands. Cette fusion se fait de manière récursive, en comparant les éléments des sous-tableaux et en les fusionnant dans l'ordre croissant pour former un tableau trié. Ce processus de fusion récursive se poursuit jusqu'à ce qu'on puisse arriver au résultat attendu qui s'agit d'un tableau trié contenant tous les éléments du tableau initial. Tel que mis en évidence dans l'image suivante.



2. ALGORITHME :

Entrée : Tableau de N entiers non triés

Sortie : Tableau trié par ordre croissant

Variables locales : i, j, elementActuel : entiers

```
début
    pour i de 1 à N - 1
        elementActuel ⇐ liste[i]
        j ⇐ i - 1
        tant que j ≥ 0 et liste[j] > elementActuel
            liste[j + 1] ⇐ liste[j]
            j ⇐ j - 1
        fin tant que
        liste[j + 1] ⇐ elementActuel
    fin pour
fin
```

3. COMPLEXITE DU TRI FUSION :

- ANALYSE SUCCINCTE DE LA COMPLEXITE TEMPORELLE :
  - **Complexité au meilleur : O (n log n)**
  - **Complexité en moyenne : O (n log n)**
  - **Complexité au pire : O (n log n)**

  Où **n** représente le nombre d'éléments dans la liste à trier et ***log(n)*** est le logarithme en base **2**.

- ÉTUDE THEORIQUE DE LA COMPLEXITE TEMPORELLE :

Le fonctionnement du tri fusion l'oblige à obtenir la même complexité dans les différents états du tableau à trier car ce tri consiste à diviser le tableau à des partitions et les fusionner après peu importe le cas du tableau.

On peut trouver le tableau à trier soit en :

- **Meilleur cas :** Tableau déjà trié.
- **Moyen cas :** Tableau avec une partie triée et une autre non.
- **Pire cas :** Tableau totalement inversé.

On pourra donc simplifier la complexité des trois cas par :

- La première étape qui est la division d'un tableau de **n** élément, où on effectue **n-1** divisions et aucune comparaison. Ainsi la complexité est **O(n)**.

- La deuxième étape qui est la fusion des sous tableaux jusqu'à l'obtention du tableau trié de taille **n**, la complexité est équivalente à **O (n log(n))**.

Preuve du résultat par le principe de récurrence :

Soit T un tableau de taille ***n*** et ***O (n log(n))*** la complexité de ce tableau dans tous les cas.

Initialisation :

Pour un tableau de taille ***n = 1***, aucun tri n'est nécessaire car il n'y a qu'un seul élément. Donc, la complexité est ***o (1)***, ainsi la propriété est vérifiée pour ***n = 1***.

Hérédité :

Supposons que pour tout tableau de taille ***n***, l'algorithme de tri fusion a une complexité de ***O (n log(n))***. Montrons que l'algorithme fonctionne également pour un tableau de taille **n+1** et que sa complexité pour cette taille est ***O (n+1 log(n+1))***.

On a :

L'étape de division : On partitionne le tableau initial T en deux tableaux T1 et T2. Chacun des deux a donc une taille qui vaut $\frac{n+1}{2}$.

L'étape du tri : Les deux moitiés sont triées récursivement à l'aide de l'algorithme de tri. Par hypothèse de récurrence, la complexité de chaque moitié est $O\left(\frac{n+1}{2}\ log\ \left(\frac{n+1}{2}\right)\right)$.

L'étape de fusion : On fusionne les deux sous tableaux triés. Le processus de fusion a une complexité linéaire ***O(n).***

La complexité totale de l'algorithme de tri fusion pour éléments est la somme des complexités des étapes de division, de tri et de fusion :

$$O\left(\frac{n+1}{2}\ log\ \left(\frac{n+1}{2}\right)\right) + O\left(\frac{n+1}{2}\ log\ \left(\frac{n+1}{2}\right)\right) + O\ (n)$$

En simplifiant, On obtient :

$O\left(\frac{n+1}{2}\ log\ \left(\frac{n+1}{2}\right)\right) + O\ (n)$ (car c'est le terme dominant en termes de croissance)

$= O\left(n\left(log\ (n+1) - log(2)\right)\right) + O\ (n)$ (car $\frac{n+1}{2} \approx n$ pour des valeurs ***n*** suffisamment grandes)

$= O\left(n + 1\ log\ (n+1)\right) + O\ (n)$ (***log(2)*** est une constante, donc nous pouvons l'ignorer dans la notation)

$= O\left(n + 1\ log\ (n+1)\right)$

Ainsi, la complexité d'un tableau de taille ***n+1*** est $O\left(n + 1\ log\ (n+1)\right)$ donc si l'algorithme fonctionne pour ***n+1***, alors il fonctionne aussi pour ***n***.

Conclusion :

Par le principe de récurrence, nous concluons que l'algorithme de tri fusion a une complexité de ***O (n log(n))*** pour tout tableau de taille ***n*** dans le meilleur cas, le moyen et le pire.

4. Complexite sous les projecteurs : Devoilement des Avantages et Limitations du tri

- ### Avantages :
  - **Complexité temporelle efficace** : Le tri fusion est l'un des algorithmes de tri les plus efficaces car il garantit des performances optimales pour de grandes quantités de données grâce à sa complexité temporelle de ***O (n log n)*** même dans le pire des cas. Cette complexité permet une exécution rapide du tri, ce qui est crucial dans de nombreux contextes d'application où le temps de traitement est critique.
  - **Facilité d'analyse** : La complexité du tri fusion est relativement facile à analyser et à comprendre, ce qui facilite l'évaluation de ses performances et son utilisation dans divers contextes.

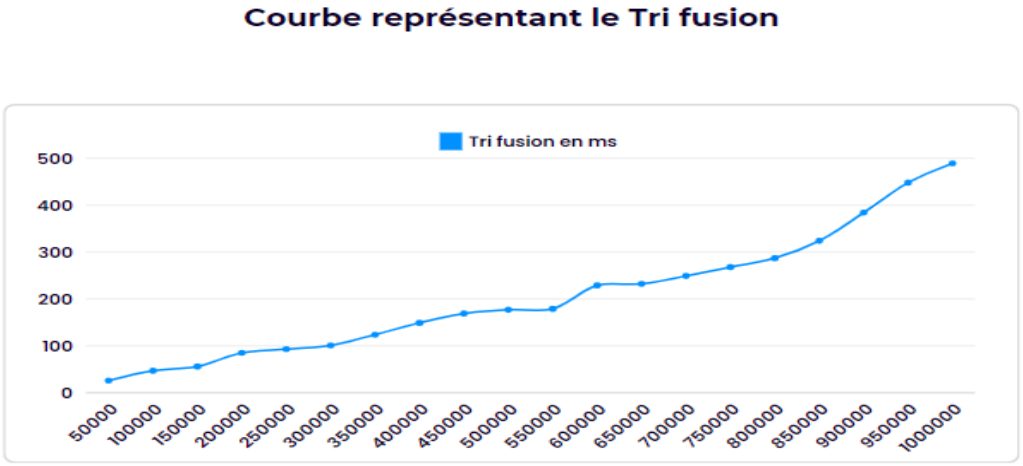
- ### Limitations :
  - **Utilisation de la mémoire supplémentaire :** La complexité spatiale du tri fusion peut le rendre moins adapté à certains scénarios d'utilisation à cause de son besoin de mémoire supplémentaire pour stocker les sous-listes temporaires lors de la fusion, surtout dans des environnements où les ressources sont limitées.
  - **Over Head récursif** : L'approche récursive du tri fusion peut entraîner un certain over Head en termes de temps d'exécution et de consommation de la pile d'appels, en particulier pour de très grandes listes.

5. Analyse du graphe representatif

- ### Tableau representant la taille du tableau et le temps d'execution :

Taille	Temps en ms
0	0ms
50000	26ms
100000	47ms
150000	56ms
200000	85ms
250000	93ms
300000	101ms
350000	124ms
400000	149ms
450000	169ms
500000	177ms
550000	179ms
600000	229ms
650000	232ms
700000	249ms
750000	268ms
800000	287ms
850000	324ms
900000	384ms
950000	448ms
1000000	489ms

- LA COURBE :



- ANALYSE DE LA COURBE :

En étudiant le graphe ci-dessus, on constate que la courbe a commencé une vraie évolution jusqu'après atteindre une taille très importante qui s'agit de 200000. Après la taille 1000000, l'évolution du temps d'exécution reste remarquable et très importante. Cela prouve donc que le temps d'exécution du tri fusion devient significativement perceptible uniquement pour des tableaux de très grande taille, comme le montre l'évolution de la courbe du temps d'exécution en fonction de la taille du tableau.

TRI PAR DENOMBREMENT

1. DEFINITION ET PRINCIPE :

-Le tri par dénombrement est l'un des algorithmes de tri les plus rapides en référence à sa classe de complexité en temps. Il s'agit d'un algorithme de tri stable utilisé uniquement sur des listes de valeurs entières qui se caractérisent par une quantité d'une faible dispersion définie par :

$$\Delta = \frac{\{v_i | v_i \in L\}}{max(L) - min(L)}$$

Où le numérateur représente le nombre de valeurs distinctes dans la liste L, et le dénominateur désigne la plage totale des valeurs dans L qui est la différence entre la plus grande et la plus petite valeur.

-Le principe de ce tri est de parcourir la totalité du tableau ***T*** en suivant les étapes :

1ère étape : compter le nombre de fois que chaque élément apparaît et les placer dans le tableau d'effectif ***E*** dont la taille est déterminée par la valeur maximale dans le tableau ***T*** (avec ***E[j]*** représentant le nombre d'occurrences de ***j*** dans le tableau). Ensuite, calculer la somme des valeurs obtenues pour chaque case ***E[i],*** indiquant ainsi le nombre de clés inférieures ou égales à ***i***.

2ème étape : Elle consiste à parcourir le tableau ***E*** pour déterminer l'indice final de chaque élément de ***T***, puis les placer dans le tableau ***B***. Enfin, copier les éléments du tableau ***B*** dans le tableau ***T*** pour obtenir le résultat final du tri.

2. ALGORITHME :

Entrée : T un tableau de n éléments

Sortie : R le tableau trié des éléments de T

Résultat : Récupération du tableau trié des éléments de T dans R

Variables locales : i : variable de boucle utilisée pour parcourir les éléments du tableau T.

nb : tableau de compteurs pour enregistrer le nombre d'apparitions de chaque élément dans T

pos : tableau pour enregistrer la position de départ de chaque élément dans R

```
début
    pour i := 0 à k-1 faire                                   initialisations
        nb[i] := 0
    pour i := 1 à n faire                                     calcul des nombres d'apparitions
        nb[T[i]] := nb[T[i]] + 1,
        pos[0] := 0
    pour i := 1 à k-1 faire          calcul des indices du premier élément de chaque catégorie
        pos[i] := pos[i-1] + nb[i-1]
    pour i := 1 à n faire                    recopie des éléments originaux du tableau T dans R
        R[pos[T[i]]] := T[i],
        pos[T[i]] := pos[T[i]] + 1
    renvoyer R
fin procédure
```

3. COMPLEXITE DU TRI PAR DENOMBREMENT :

- ANALYSE SUCCINCTE DE LA COMPLEXITE TEMPORELLE :

- **Meilleur cas : O (n + k)**
- **Cas moyen : O (n + k)**
- **Pire cas : O (n + k)**

Où n est le nombre d'éléments dans la liste à trier et k est la plage de valeurs possibles dans cette liste.

- ÉTUDE THEORIQUE DE LA COMPLEXITE TEMPORELLE :

Le tri par dénombrement a une complexité linéaire dans les différents cas, cela est dû au fait qu'il opère sans faire de comparaisons sur les termes de la liste. C'est un tri qui consiste à compter le nombres occurrences dans un premier temps et à reconstruire le tableau trié.

- Au début on commence par le parcours du tableau en comptant le nombre d'occurrence de chaque élément et pour un tableau de ***n*** éléments on obtient une complexité de ***O(n)*** puisqu'on visite tous les éléments du tableau.

- Après on reconstruit le tableau en se basant sur le comptage des occurrences de chaque élément et cela nécessite une complexité de ***O(k).***

Preuve du résultat par le principe de récurrence :

Soit T un tableau de taille ***n*** et $\boldsymbol{O}\ (\boldsymbol{k}+\boldsymbol{n})$ la complexité de ce tableau dans tous les cas.

Initialisation :

Pour un tableau de taille ***n = 1***, le tri par dénombrement nécessite simplement de compter les occurrences de chaque élément, ce qui prend **O(k)** opérations. Ainsi, la complexité est ***O(1),*** ainsi la propriété est vérifiée pour ***n=1***.

Hérédité :

Supposons que pour tout tableau de taille ***n***, l'algorithme de tri par dénombrement a une complexité de $\boldsymbol{O}\ (\boldsymbol{k}+\boldsymbol{n})$**.** Montrons que la complexité pour un tableau de taille ***n+1*** est $\boldsymbol{O}(\boldsymbol{k}+\boldsymbol{n}+\mathbf{1})$.

On a :

L'énumération des occurrences : nous comptons les occurrences de chaque élément dans le tableau, ce qui prend ***O(k)*** opérations.

La construction du tableau trié : En utilisant les compteurs d'occurrences, nous construisons le tableau trié, ce qui prend ***O(n+1)*** opérations.

La complexité totale de l'algorithme de tri par dénombrement est la somme des complexités des deux étapes citées précédemment :

$\boldsymbol{O}(\boldsymbol{k})+\boldsymbol{O}\ (\boldsymbol{n}+\mathbf{1})$

En utilisant les propriétés des notations de Landau ***"O"*** nous pouvons dire que ***O(k)+O(n+1) = O(n+k+1)***

Ainsi, la complexité d'un tableau de taille ***n+1*** est $\boldsymbol{O}\ (\boldsymbol{k}+\boldsymbol{n}+\mathbf{1})$ donc si l'algorithme fonctionne pour ***n+1***, alors il fonctionne aussi pour ***n***.

Conclusion :

Par le principe de récurrence, nous concluons que l'algorithme de tri par dénombrement a une complexité de $\boldsymbol{O}\ (\boldsymbol{k}+\boldsymbol{n})$ pour tout tableau de taille ***n*** dans le meilleur cas, le moyen et le pire.

4. Complexite sous les projecteurs : Devoilement des Avantages et Limitations du tri

- Avantages :
  - **Efficacité en temps :** l'efficacité du temps d'exécution du tri par dénombrement est très importante comparé aux autres tris car elle est influencée par la complexité de

l'algorithme, qui est linéaire par rapport à la taille du tableau et dépend de la plage de valeurs, ce qui en fait une option efficace pour les ensembles de données avec une plage limitée de valeurs.

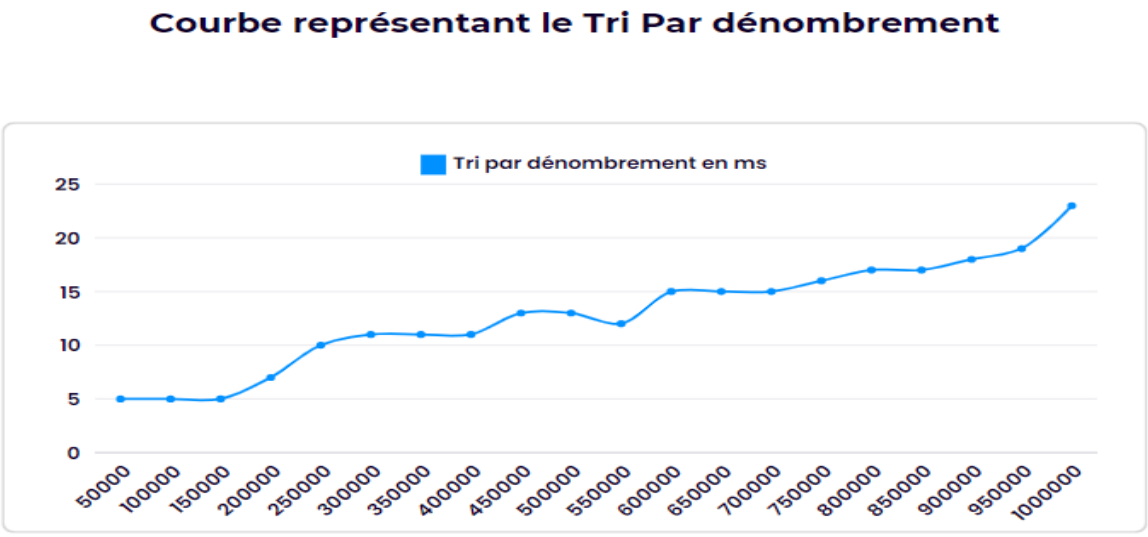
- LIMITATIONS :
  - **Utilisation de la mémoire :** Le tri par dénombrement nécessite une quantité de mémoire supplémentaire proportionnelle à la plage de valeurs possibles dans le tableau. Si la plage de valeurs est très grande, cela peut entraîner une consommation élevée de mémoire, rendant l'algorithme peu pratique pour les ensembles de données avec une plage de valeurs très étendue.
  - **Dépendance à la plage de valeurs :** La performance du tri par dénombrement est influencée par la variabilité des valeurs présentes dans le tableau. Si la plage de valeurs est très grande par rapport au nombre d'éléments à trier, la complexité peut augmenter et rendre le tri moins efficace que d'autres algorithmes de tri pour des ensembles de données de grande taille.

5. ANALYSE DU GRAPHE REPRESENTATIF

- TABLEAU REPRESENTANT LA TAILLE DU TABLEAU ET LE TEMPS D'EXECUTION :

Taille	Temps en ms
0	0ms
50000	5ms
100000	5ms
150000	5ms
200000	7ms
250000	10ms
300000	11ms
350000	11ms
400000	11ms
450000	13ms
500000	13ms
550000	12ms
600000	15ms
650000	15ms
700000	15ms
750000	16ms
800000	17ms
850000	17ms
900000	18ms
950000	19ms
1000000	23ms

- LA COURBE :



- ANALYSE DE LA COURBE :

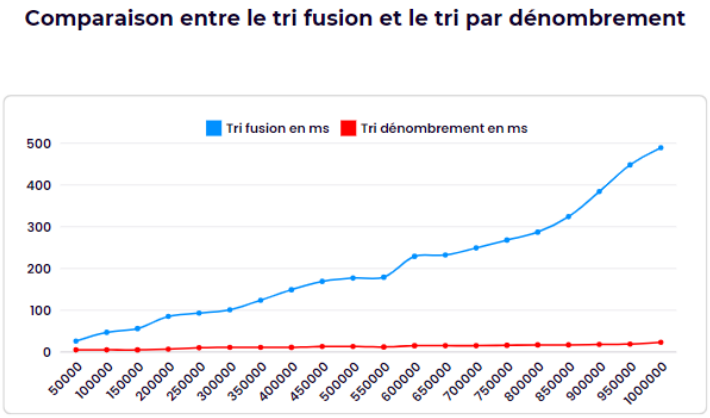
D'après le graphe ci-dessus, on peut considérer que le temps d'exécution du tri par dénombrement évolue très lentement par rapport aux autres tris. Le temps d'exécution a commencé à croître très doucement en allant d'une valeur de **5** ms pour une taille de **50000** à une valeur de **10** ms pour une taille de **250000** et se stabiliser dans la même valeur jusqu'à la taille de **400000** pour afin revoir encore l'évolution très lente du temps d'exécution en augmentant la taille du tableau jusqu'à **1000000** pour avoir une valeur de **23** ms au temps d'exécution qui se considère une valeur très petite par rapport à la taille du tableau donnée en paramètre. Ce qui confirme parfaitement les propriétés mentionnées précédemment concernant le tri par dénombrement.

COMPARAISON ENTRE LES 3 TRIS :

Le tri par insertion est tellement lent qu'on voit évidemment sa différence par rapport aux 2 autres. On passera donc dans cette analyse de courbe à l'étude uniquement du tri fusion et le tri par dénombrement pour qu'on puisse remarquer clairement la différence entre les deux tris.

COMPARAISON ENTRE LE TRI FUSION ET LE TRI PAR DENOMBREMENT :

- ### LA COURBE :



- ### ANALYSE DE LA COURBE :

On constate après étude des courbes représentées sur le graphique ci-dessus qu'on peut clairement voir la différence entre les deux tris.

Les deux courbes n'évoluent pas de la même manière. Les temps d'exécution des deux tris se séparent largement en augmentant la taille du tableau. On arrive à remarquer clairement la différence lorsqu'on atteint la taille de **100000** qui montre l'augmentation du temps d'exécution du tri fusion par rapport au tri par dénombrement qui a donc gardé à son tour une marge de temps plus minimisée.

Le tri fusion a donc reconnu une augmentation importante du temps d'exécution contrairement au tri par dénombrement qui est resté quasiment stable même en affectant de très grande taille au tableau.

BILAN GENERALE :

- **Le tri par insertion,** bien qu'étant considéré comme un algorithme classique et simple, possède une élégance algorithmique unique qui en fait un sujet d'étude fascinant. Son fonctionnement pas à pas, en plaçant chaque élément à sa place appropriée dans la séquence déjà triée, révèle une approche organique du triage qui peut être explorée en profondeur pour comprendre les principes fondamentaux des algorithmes de tri.

- **Le tri fusion** se considère comme l'un des algorithmes de tri les plus efficaces, ses performances sont relativement indépendantes de la répartition initiale des valeurs. En outre sa complexité optimale en ***O (n log n)*** en fait un choix idéal pour des tableaux de grande taille. Il est stable et peut être utilisé pour des données non triées ou déjà triées. De plus, son processus de fusion récursif, où des sous-tableaux triés sont combinés pour former un tableau trié plus large, illustre parfaitement ces concepts et offre une perspective fascinante sur la puissance des approches algorithmiques avancées.

- **Le tri par dénombrement** reste un algorithme de tri efficace qui peut être utilisé pour trier une collection d'objets. Il est particulièrement utile lorsqu'il s'agit de grandes collections d'objets, car il s'agit d'un algorithme stable et relativement simple à mettre en œuvre.

Toutefois, il n'est pas adapté au tri d'objets présentant un large éventail de valeurs ou des valeurs négatives. Globalement, le tri par dénombrement est une option intéressante pour les tableaux avec une plage de valeurs limitée. Sa complexité linéaire ***O (n + k)***, en fait un choix efficace pour trier des données telles que des entiers dans un intervalle connu. Cependant, il nécessite souvent plus de mémoire supplémentaire en fonction de la plage des valeurs possibles.