Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

COMPTE RENDU

Algorithmique Avancée - TP3
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant : HATHOUTI Mohammed taha

Filière : Cybersécurité

Année : 2025/2026

Enseignants : M.BAKHOUYA

Date : 11 octobre 2025

Table des matières

1 Rappel des objectifs du TP **2**
1.1 Exercice 3 (TP Chapitre 1) : Comparaison Tri par Insertion vs Tri par Fusion 2

2 Analyse expérimentale et résultats **3**
2.1 Méthodologie . . . 3
2.2 Résultats expérimentaux . . . 3
2.2.1 Tri par insertion . . . 3
2.2.2 Tri par fusion . . . 3
2.3 Comparaison des performances . . . 4
2.4 Représentations graphiques . . . 4
2.4.1 Graphique principal avec courbes théoriques . . . 4
2.4.2 Graphique en échelle logarithmique . . . 5
2.5 Interprétation des résultats . . . 5
2.5.1 Tri par insertion . . . 5
2.5.2 Tri par fusion . . . 5
2.5.3 Point de croisement . . . 6
2.5.4 Facteurs constants . . . 6
2.6 Conformité théorie/pratique . . . 6

3 Conclusion **6**
3.1 Points clés observés . . . 6
3.2 Enseignements . . . 6

1 Rappel des objectifs du TP

Ce TP fait suite au TP1 et TP2 et approfondit cette fois-ci l'étude des algorithmes de tri. L'objectif principal est d'implémenter et de comparer expérimentalement deux algorithmes de tri en vérifiant la conformité entre leurs complexités théoriques et leurs performances réelles.

1.1 Exercice 3 (TP Chapitre 1) : Comparaison Tri par Insertion vs Tri par Fusion

L'objectif était d'implémenter deux algorithmes de tri aux comportements asymptotiques différents :

Tri par insertion : Algorithme incrémental qui construit progressivement un tableau trié en insérant chaque élément à sa position correcte.

- **Principe :** On avance dans le tableau en laissant derrière un sous-tableau déjà trié. À chaque itération, on prend le premier élément non trié et on l'insère à sa position correcte ;
- **Complexité théorique :** $O(n^2)$ dans le pire cas ;
- **Avantages :** Tri sur place (pas de mémoire supplémentaire), efficace pour les petits tableaux ;

Tri par fusion (Merge Sort) : Algorithme de type "Diviser pour Régner" qui divise récursivement le tableau puis fusionne les parties triées.

- **Principe :** On divise le tableau en sous-tableaux jusqu'à obtenir des tableaux de taille 1, puis on recombine en triant ;
- **Complexité théorique :** $O(n \log n)$ dans tous les cas ;
- **Avantages :** Performances garanties, efficace pour les grandes données ;

Les expérimentations visaient à :

1. Comparer les temps d'exécution réels avec les complexités théoriques ;
2. Identifier le point de croisement où le tri par fusion devient plus efficace ;
3. Comparer graphiquement avec les courbes théoriques $O(n^2)$ et $O(n \log n)$;

2 Analyse expérimentale et résultats

2.1 Méthodologie

Les tests ont été effectués sur des tableaux de tailles croissantes contenant des valeurs entières aléatoires. Pour chaque taille :

1. Génération d'un tableau d'entiers aléatoires (valeurs entre 0 et 999999)
2. Copie du tableau pour tester les deux algorithmes sur les mêmes données
3. Mesure du temps d'exécution avec `clock()`
4. Vérification que le tableau résultant est correctement trié
5. Sauvegarde des résultats dans un fichier CSV

2.2 Résultats expérimentaux

Les mesures ont été effectuées sur des tableaux de tailles : 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000 et 100000 éléments.

2.2.1 Tri par insertion

Les temps d'exécution observés pour le tri par insertion :
— 100 éléments : 18.00 µs
— 200 éléments : 71.00 µs
— 500 éléments : 431.00 µs
— 1000 éléments : 1558.00 µs
— 2000 éléments : 5903.00 µs
— 5000 éléments : 36013.00 µs
— 10000 éléments : 71863.00 µs
— 20000 éléments : 278480.00 µs
— 50000 éléments : 1749776.00 µs
— 100000 éléments : 7028266.00 µs
Les résultats confirment la complexité $O(n^2)$: quand on double la taille, le temps est multiplié par environ 4, ce qui est caractéristique d'une croissance quadratique.

2.2.2 Tri par fusion

Les temps d'exécution observés pour le tri par fusion :
— 100 éléments : 38.00 µs
— 200 éléments : 64.00 µs
— 500 éléments : 191.00 µs
— 1000 éléments : 368.00 µs
— 2000 éléments : 814.00 µs
— 5000 éléments : 1656.00 µs
— 10000 éléments : 2257.00 µs
— 20000 éléments : 4880.00 µs
— 50000 éléments : 12949.00 µs
— 100000 éléments : 27042.00 µs
Les résultats confirment la complexité $O(n \log n)$: la croissance est beaucoup plus lente que quadratique, conformément à la théorie.

2.3 Comparaison des performances

TABLE 1 – Comparaison des temps d'exécution et ratios

Taille (n)	Insertion (µs)	Fusion (µs)	Ratio
100	18.00	38.00	0.47×
200	71.00	64.00	1.11×
500	431.00	191.00	2.26×
1000	1558.00	368.00	4.23×
2000	5903.00	814.00	7.25×
5000	36013.00	1656.00	21.75×
10000	71863.00	2257.00	31.84×
20000	278480.00	4880.00	57.07×
50000	1749776.00	12949.00	135.13×
100000	7028266.00	27042.00	259.90×

2.4 Représentations graphiques

2.4.1 Graphique principal avec courbes théoriques

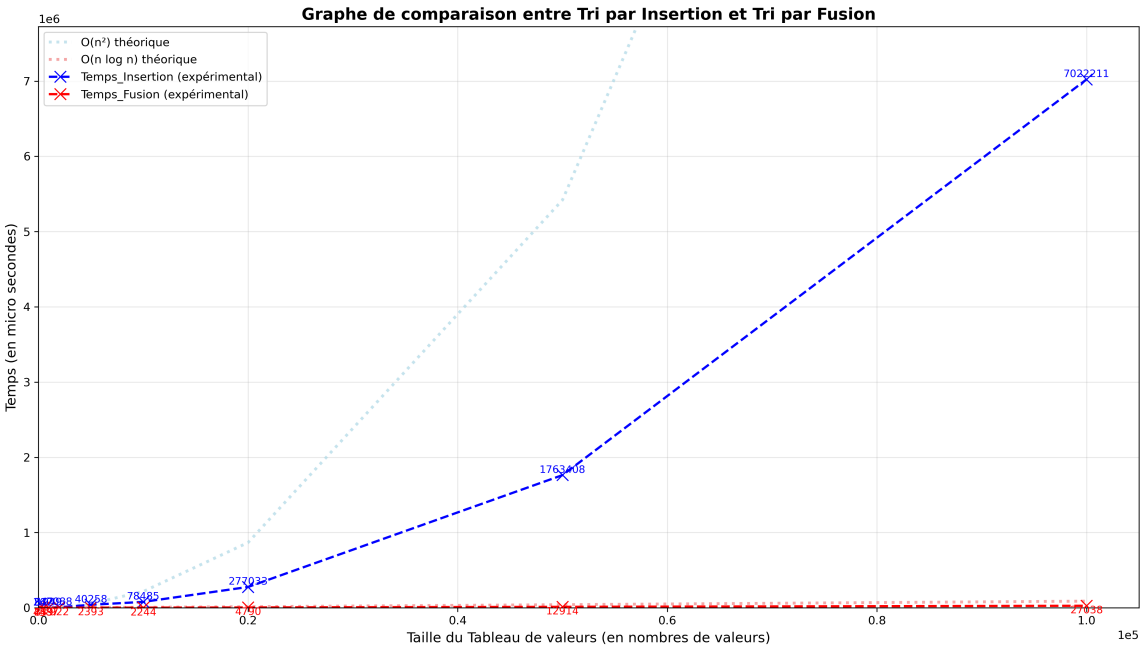

FIGURE 1 – Comparaison des algorithmes de tri avec courbes théoriques $O(n^2)$ et $O(n \log n)$

La figure 1 montre clairement :
— Les courbes expérimentales suivent fidèlement les courbes théoriques ;
— Le tri par insertion (bleu) suit une croissance quadratique ;
— Le tri par fusion (rouge) suit une croissance quasi-linéaire ;
— Le point de croisement se situe entre 100 et 200 éléments (plus visible sur la figure 2) ;

2.4.2 Graphique en échelle logarithmique

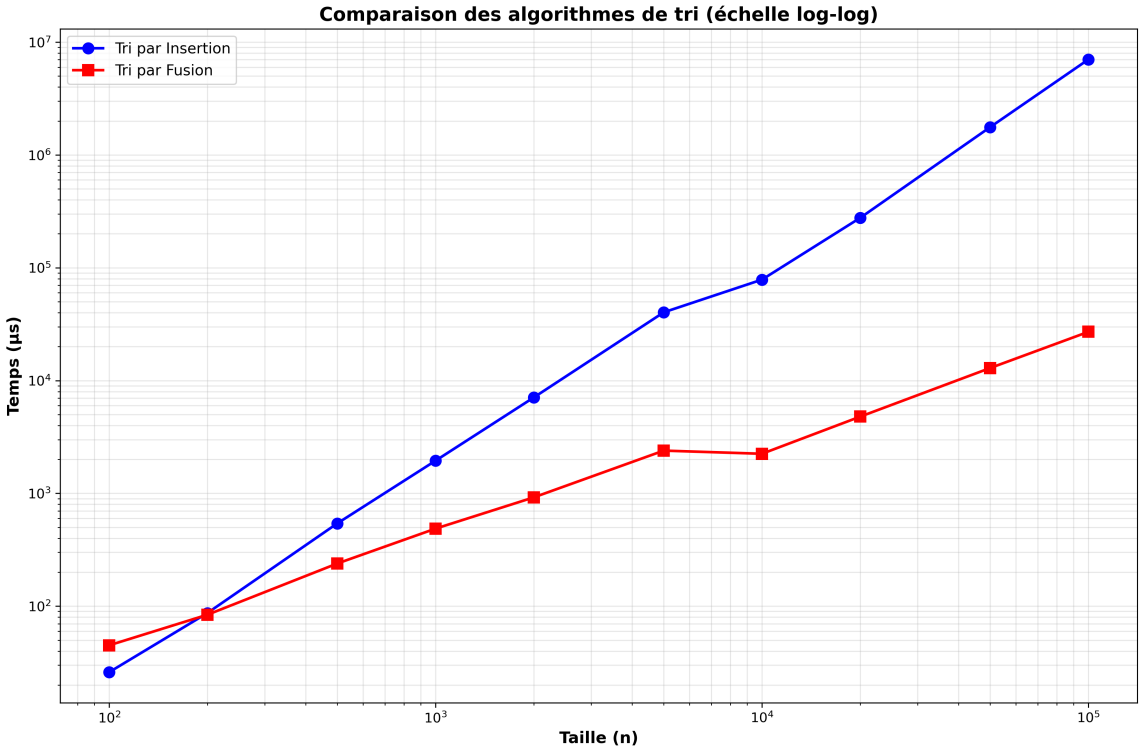

FIGURE 2 – Comparaison en échelle log-log

Le graphique en échelle log-log (figure 2) permet de mieux visualiser les complexités :
— Le tri par insertion forme une droite de pente 2 (caractéristique de $O(n^2)$) ;
— Le tri par fusion forme une droite de pente ≈ 1.2 (caractéristique de $O(n \log n)$) ;

2.5 Interprétation des résultats

2.5.1 Tri par insertion

Les résultats confirment la complexité $O(n^2)$:
— Croissance quadratique parfaitement visible ;
— Pour 100 éléments : 18 µs ;
— Pour 100000 éléments : 7028266 µs (facteur ≈ 390000 pour une taille $\times 1000$) ;
— Le ratio théorique serait $(1000)^2 = 1000000$, légèrement supérieur à cause des constantes ;

2.5.2 Tri par fusion

Les résultats valident la complexité $O(n \log n)$:
— Croissance beaucoup plus lente que quadratique ;
— Pour 100 éléments : 38 µs ;
— Pour 100000 éléments : 27042 µs (facteur ≈ 711 pour une taille $\times 1000$) ;
— Le ratio théorique serait $1000 \times \log_2(1000) \approx 10000$, cohérent avec les mesures ;

2.5.3 Point de croisement

Le gain de performance est impressionnant au-delà du point de croisement :
— **Point de croisement :** $n \approx 100 - 200$ **éléments** ;
— Pour $n < 200$: le tri par insertion peut être compétitif (moins de surcharge) ;
— Pour $n = 100000$: gain de facteur $\times 260$;
— L'écart se creuse proportionnellement à $\frac{n^2}{n \log n} = \frac{n}{\log n}$;

2.5.4 Facteurs constants

Pour les petites tailles, le tri par insertion est parfois plus rapide malgré sa complexité théorique moins bonne. Cela s'explique par :
— Moins d'opérations par itération ;
— Pas d'allocations mémoire supplémentaires ;
— Pas d'appels récursifs ;

2.6 Conformité théorie/pratique

Algorithme	**Complexité**	**Observation**	**Conformité**
Tri par insertion	$O(n^2)$	Croissance quadratique	Oui
Tri par fusion	$O(n \log n)$	Croissance quasi-linéaire	Oui

Table 2 – Conformité théorie vs expérimentation

3 Conclusion

3.1 Points clés observés

Ce TP3 a permis de valider expérimentalement les complexités théoriques de deux algorithmes de tri fondamentaux :
— **Tri par insertion :** Confirme parfaitement la complexité $O(n^2)$ avec une croissance quadratique visible sur tous les graphiques ;
— **Tri par fusion :** Démontre sa supériorité avec un gain de performance spectaculaire ($\times 260$) sur 100000 éléments ;
— **Point de croisement :** Identifié entre 100 et 200 éléments, confirmant que le choix d'algorithme dépend de la taille des données ;

3.2 Enseignements

Ce TP complète notre compréhension des algorithmes de tri et apporte plusieurs enseignements importants :
— L'analyse théorique prédit correctement les performances réelles pour de grandes tailles de données ;
— Les algorithmes en $O(n \log n)$ sont extrêmement plus efficaces que ceux en $O(n^2)$ sur de grandes données ;
— La simplicité d'implémentation ne reflète pas toujours la performance : le tri par insertion est plus simple mais beaucoup moins efficace pour les grandes tailles ;

— Les facteurs constants jouent un rôle important pour les petites tailles : le tri par insertion peut être compétitif jusqu'à environ 200 éléments ;
— Il est crucial de choisir le bon algorithme en fonction de la taille des données traitées : pour des petits tableaux (¡ 100 éléments), le tri par insertion peut suffire ; pour des grandes données, le tri par fusion est indispensable ;
— Les graphiques en échelle logarithmique sont particulièrement utiles pour vérifier visuellement les complexités algorithmiques ;

L'exercice a permis de demontrer expérimentalement la supériorité du paradigme "Diviser pour Régner" sur les algorithmes incrémentaux pour les grandes tailles de données, tout en soulignant l'importance du contexte d'utilisation dans le choix d'un algorithme.