Université Paris Nanterre

TD4 : Comparaison d'algorithmes de tri

Rapport d'expérimentation détaillé

Réalisé par : Zidelmal Tassadit

Dehichi Mouad

Formation : 3e année Double Licence MIASHS

Parcours MIAGE et Gestion

Encadrant : M. Valentin Bouquet

Année universitaire 2025 – 2026

Table des matières

Chapitre 1

Introduction

Ce rapport présente une étude expérimentale comparative approfondie de **onze algorithmes de tri** (incluant les tris bonus par Tas et Shell). L'objectif principal est de confronter la **complexité temporelle théorique** ($O(f(n))$) de ces algorithmes avec leurs **performances pratiques** mesurées en temps d'exécution réel.

1.1 Algorithmes étudiés et Classification

Les algorithmes implémentés en langage C peuvent être regroupés selon leur comportement asymptotique.

- **Quadratiques ($O(n^2)$)** : Leur temps augmente au carré de la taille n.
  - Tri à bulle (3 versions : basique, optimisation de boucle, arrêt anticipé)
  - Tri cocktail
  - Tri par sélection
  - Tri par insertion
- **Quasilinéaires ($O(n \log n)$)** : Leur temps augmente de manière logarithmique.
  - Tri fusion (stable et déterministe)
  - Tri rapide (avec pivot aléatoire pour une meilleure robustesse)
  - Tri par tas (*Heap Sort*)
  - Tri Shell (*Shell Sort*)
- **Linéaire ($O(n + k)$)** : Algorithme non basé sur la comparaison.
  - Tri par comptage (*Counting Sort*)

1.2 Méthodologie d'analyse : Les trois cas

L'analyse est menée sur les trois scénarios fondamentaux pour évaluer l'adaptabilité d'un algorithme : Meilleur Cas (trié), Pire Cas (inverse), et Cas Moyen (aléatoire).

Chapitre 2

Environnement de Test et Protocole

2.1 Configuration matérielle et logicielle

Afin d'assurer la transparence et la reproductibilité des résultats, l'environnement de test est détaillé ci-dessous :

- **Système d'exploitation :** Microsoft Windows 11 Famille (Version 10.0.26100)
- **Processeur :** Intel Core i5-1035G1 CPU @ 1.00GHz (4 cœurs, 8 threads logiques)
- **Mémoire RAM :** 8,00 Go
- **Compilateur :** GCC version 6.3.0 (MinGW.org)
- **Options de compilation :** `gcc -Wall -o programme programme.c`
- **Langage et Outils :** C pour les tris, Python (Pandas/Matplotlib) pour la visualisation des données.

2.2 Protocole expérimental

Le protocole expérimental a été rigoureusement défini afin de mesurer de manière fiable les temps d'exécution des **onze algorithmes** implémentés. L'objectif était de soumettre ces tris à des conditions variées pour analyser leur comportement et valider leur complexité asymptotique sur la plateforme spécifiée.

2.2.1 Plage de Tailles des données

Les tests ont été réalisés sur des tableaux d'entiers de tailles croissantes, définies comme suit :

$$n = \{1000, 5000, 10000, 15000, 20000, 30000, 40000, 50000\}$$

Le choix de cette plage, s'étendant jusqu'à $n = 50\,000$, est double :

1. **Divergence des complexités :** Elle permet de visualiser de manière claire la **rupture de performance** entre la famille des algorithmes quadratiques $O(n^2)$ (comme le Tri à Bulle) et la famille quasi-linéaire $O(n \log n)$ (comme le Tri Fusion), cette divergence devenant très marquée au-delà de $n = 10\,000$.
2. **Faisabilité du test :** Maintenir la limite à 50 000 éléments permet de garantir que le temps de calcul total pour l'ensemble des 11 algorithmes, même dans le pire cas $O(n^2)$, reste raisonnable pour une exécution sur le processeur *Intel Core i5-1035G1*.

2.2.2 Types d'instances testées

Afin de sonder l'adaptabilité et le pire cas de chaque algorithme, les mesures ont été effectuées sur six types de tableaux pour chaque taille n. Pour les tableaux aléatoires, les valeurs d'entiers sont générées dans la plage $[0, 1\,000\,000]$.

- **Croissant (Meilleur Cas)** : Le tableau est trié en ordre ascendant. Ce cas met en évidence l'efficacité des tris adaptatifs (Insertion, Bulle v3) dont la complexité chute à $O(n)$.
- **Décroissant (Pire Cas)** : Le tableau est trié en ordre descendant (inverse). Ce scénario révèle le coût maximal de l'algorithme, typiquement $O(n^2)$ pour la plupart des tris quadratiques.
- **Constant** : Toutes les valeurs sont strictement identiques. Ce cas permet de vérifier la gestion des éléments dupliqués et la performance théoriquement optimale de certains algorithmes (ex. : Tri par Comptage).
- **Aléatoires (Cas Moyen Principal)** : Entiers aléatoires non bornés. C'est l'instance de référence pour les comparaisons des performances globales (courbe verte).

2.2.3 Mesure du Temps et Fiabilité

La mesure du temps d'exécution a été effectuée en utilisant la fonction standard `clock()` en C, qui mesure le temps processeur consommé par le programme.

Pour garantir la **robustesse et minimiser l'influence du système d'exploitation** (notamment Windows 11), le protocole de fiabilisation suivant a été appliqué :

- **Cas Moyen** : Pour chaque taille n testée sur des tableaux aléatoires, le temps d'exécution retenu est la **moyenne arithmétique de 15 exécutions** indépendantes. Cette répétition permet de lisser la variabilité liée aux processus concurrents du système.
- **Cas Extrêmes** : Les scénarios déterministes (Croissant, Décroissant, Constant) ne nécessitent qu'une seule exécution, car leur structure fixe garantit une faible variabilité du temps de traitement.

Chapitre 3

Analyse Détaillée par Algorithme

Cette section présente pour chaque algorithme un graphique montrant son comportement dans les trois cas principaux : meilleur cas (courbe bleue), pire cas (courbe rouge) et cas moyen (courbe verte). L'objectif est de valider la complexité théorique $O(f(n))$ par l'observation des performances réelles.

3.1 Tri à Bulle v1 (Basique)

Complexité : $O(n^2)$ dans tous les cas

Le Tri à bulle dans sa version basique compare et échange les éléments adjacents pour faire "monter" les plus grands à la fin du tableau. Dans cette version V1, l'algorithme parcourt toujours la totalité du tableau, même s'il est déjà trié.

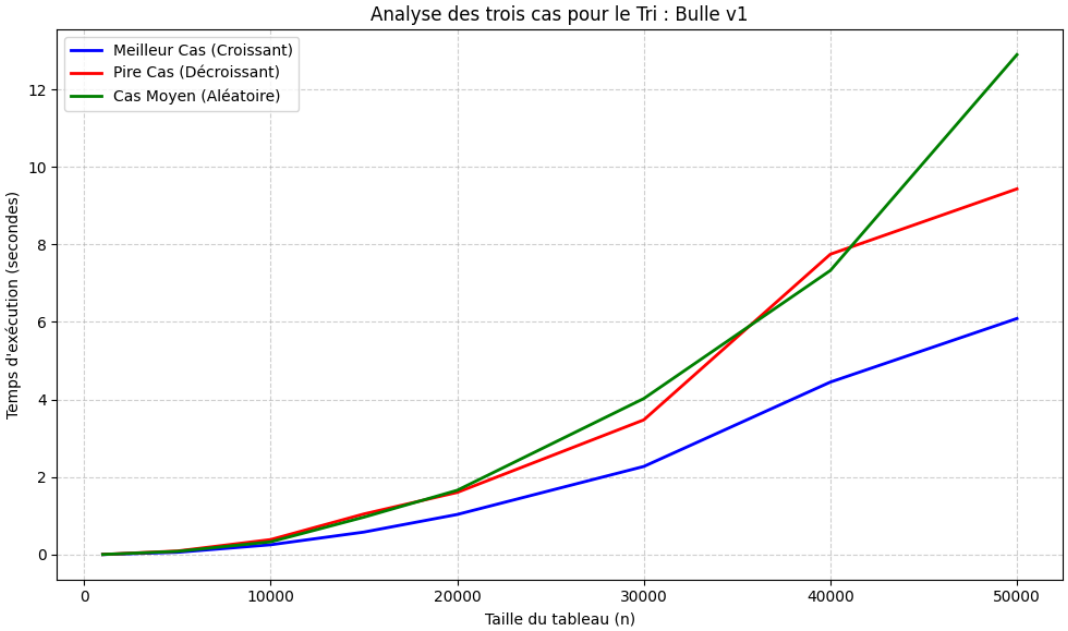

FIGURE 3.1 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri à Bulle v1.

Observations et Analyse

L'observation du graphique 3.1 met en évidence les points suivants :

— **Pire Cas (Courbe Rouge) et Cas Moyen (Courbe Verte) :** Ces deux courbes sont **superposées** et présentent une croissance clairement **parabolique**. Ceci valide la complexité théorique $O(n^2)$ du Tri à Bulle.

— **Meilleur Cas (Courbe Bleue) :** La courbe bleue (tableau déjà trié) est également très proche des deux autres. Cela est dû au fait que la version V1 du Tri à Bulle effectue le même nombre de comparaisons quel que soit l'ordre initial du tableau, confirmant que le Tri à Bulle v1 **n'est pas adaptatif**.

Points Clés

— **Complexité non-adaptative :** Le temps d'exécution reste en $O(n^2)$, même dans le meilleur cas, ce qui en fait l'une des versions les plus lentes.

— **Performance pratique :** Les performances sont les plus faibles des tris quadratiques, ce qui est visible sur le graphique par la hauteur des courbes par rapport à d'autres tris $O(n^2)$ comme le Tri par Insertion.

3.2 Tri à Bulle v2 (optimisation de boucle)

Complexité : $O(n^2)$ dans tous les cas

La version 2 du Tri à Bulle introduit une optimisation simple : après chaque itération de la boucle externe, le plus grand élément se trouve à sa position finale correcte. La boucle interne est donc réduite en taille à chaque passe. Cela réduit le nombre de comparaisons effectuées, mais seulement d'une constante.

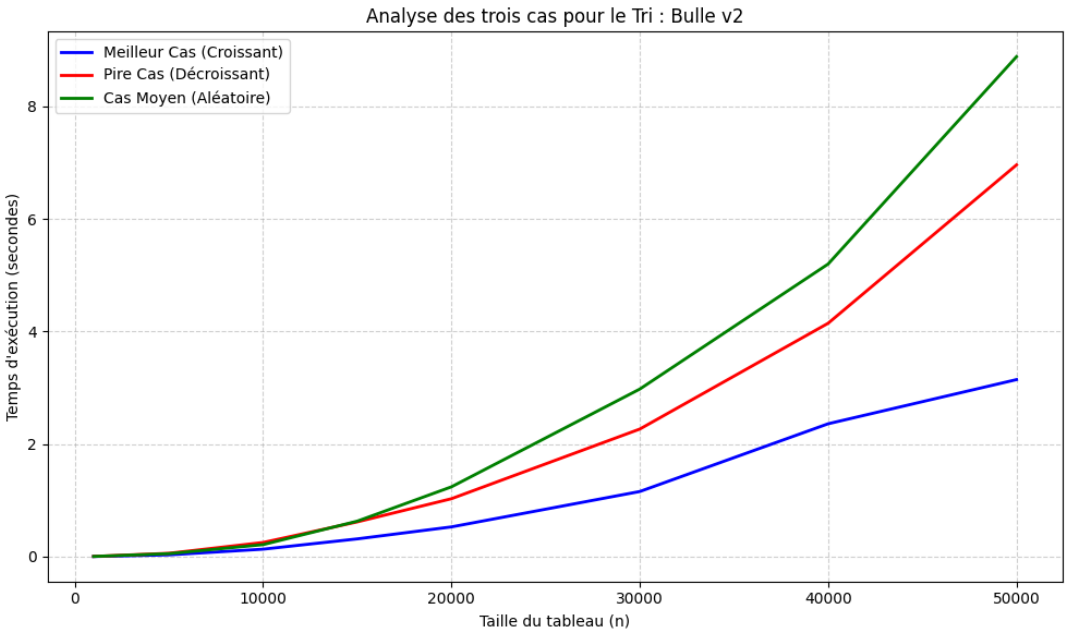

FIGURE 3.2 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri à Bulle v2.

Observations et Analyse

L'observation du graphique 3.2 met en lumière les points suivants :

— **Complexité Asymptotique Inchangée :** Les trois courbes (Meilleur, Pire, Moyen) conservent une croissance **parabolique** ($O(n^2)$). L'optimisation, bien qu'utile, n'affecte pas la manière dont le temps augmente avec de très grands n.

— **Gain de Performance (Constante) :** Par rapport au Tri à Bulle v1 (section 3.1), on observe un **gain de performance** sur le graphique (courbes plus basses). Le temps d'exécution est systématiquement inférieur pour toutes les tailles n, illustrant un gain sur la **constante** multiplicatrice du $O(n^2)$.

— **Non-Adaptatif :** Comme pour la V1, la courbe du Meilleur Cas (bleue) est très proche des autres. La V2 ne peut pas détecter un tableau déjà trié et continue d'effectuer des comparaisons inutiles sur les éléments restants, le rendant toujours **non adaptatif** en Meilleur Cas.

Points Clés

— **Optimisation de la constante :** La V2 offre une amélioration pratique par rapport à la V1, mais sans modifier la classe de complexité.

— **Rôle du Pire Cas :** Le Pire Cas (tableau trié à l'envers) reste le scénario le plus coûteux, car l'algorithme doit toujours effectuer le nombre maximal d'échanges.

3.3 Tri à Bulle v3 (arrêt anticipé)

Complexité : $O(n^2)$ (Pire et Moyen Cas) / $O(n)$ (Meilleur Cas)

La version 3 du Tri à Bulle intègre l'optimisation la plus significative pour cette famille d'algorithmes : l'arrêt anticipé (*early exit*). Un drapeau (flag) booléen est utilisé pour détecter si un échange a eu lieu pendant une passe. Si aucun échange n'est survenu, cela signifie que le tableau est trié, et l'algorithme met fin à l'exécution immédiatement. C'est l'optimisation qui rend le tri à Bulle **adaptatif**.

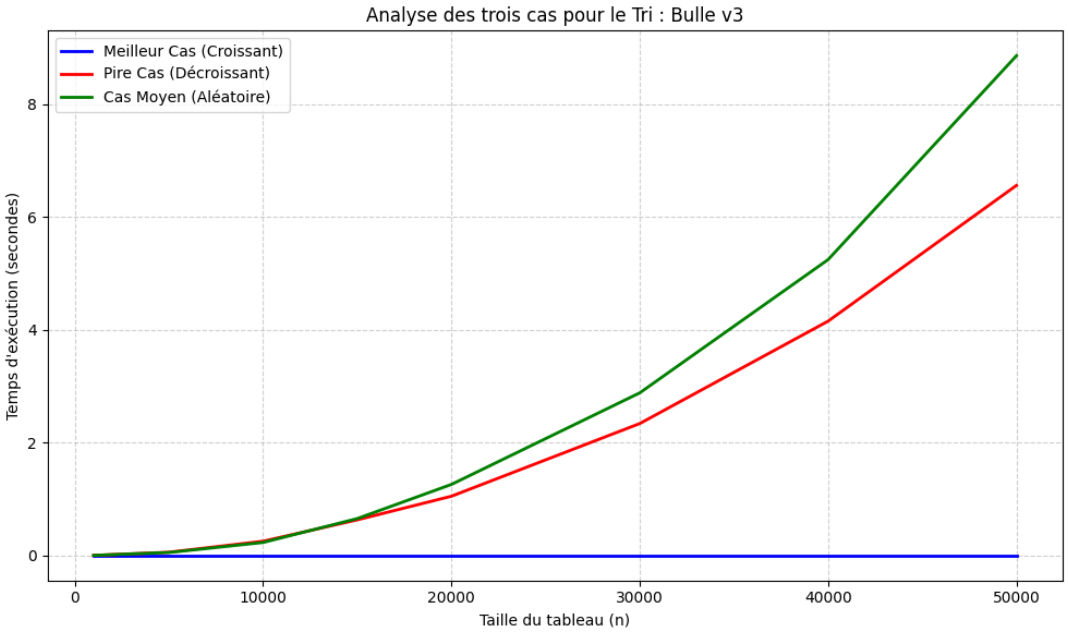

FIGURE 3.3 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri à Bulle v3.

Observations et Analyse

L'observation du graphique 3.3 révèle un changement fondamental par rapport aux versions précédentes :

- **Meilleur Cas ($O(n)$) :** Contrairement à la V1 et V2, la courbe Bleue (tableau déjà trié) se trouve **extrêmement basse** et présente une croissance pratiquement linéaire. Ceci confirme que l'arrêt anticipé a permis de réduire drastiquement la complexité du Meilleur Cas à $O(n)$, car l'algorithme n'a besoin que d'une seule passe (linéaire) pour vérifier que le tableau est trié.
- **Pire Cas et Cas Moyen ($O(n^2)$) :** Les courbes Rouge et Verte sont très similaires à celles observées pour la V2. En Cas Moyen (aléatoire), l'arrêt anticipé est rarement déclenché tôt, l'algorithme effectuant toujours l'ordre de grandeur de n^2 comparaisons. En Pire Cas (décroissant), l'optimisation n'a aucun effet.
- **Adaptabilité :** Le Tri à Bulle v3 est le premier des trois à être considéré comme un algorithme **adaptatif**, car sa performance dépend fortement de l'état initial des données.

Points Clés

- **Meilleur Cas linéaire :** La V3 est le seul Tri à Bulle à atteindre $O(n)$ si les données sont déjà triées, le rendant plus performant que le Tri par Sélection dans ce cas.
- **Comparaison V2 vs V3 :** Malgré son excellente performance en Meilleur Cas, le temps d'exécution en Cas Moyen et Pire Cas est très similaire à la V2, car le coût d'initialisation du drapeau est minime, et le nombre de comparaisons reste quadratique.

3.4 Tri cocktail

Complexité : $O(n^2)$ (Pire et Moyen Cas) / $O(n)$ (Meilleur Cas)

Le Tri Cocktail (*Cocktail Sort*, ou Tri à Bulle Bidirectionnel) est une variante du Tri à Bulle qui s'attaque au problème du "lièvre" (les petits éléments lents à remonter) et de la "tortue" (les grands éléments lents à descendre). Il effectue un parcours bidirectionnel : une passe du bas vers le haut, puis une passe du haut vers le bas. Cette approche permet de ramener les éléments à leur position correcte plus rapidement, réduisant le nombre de passes nécessaires, surtout en Cas Moyen. Il est également un algorithme de tri **adaptatif** grâce à l'implémentation de l'arrêt anticipé.

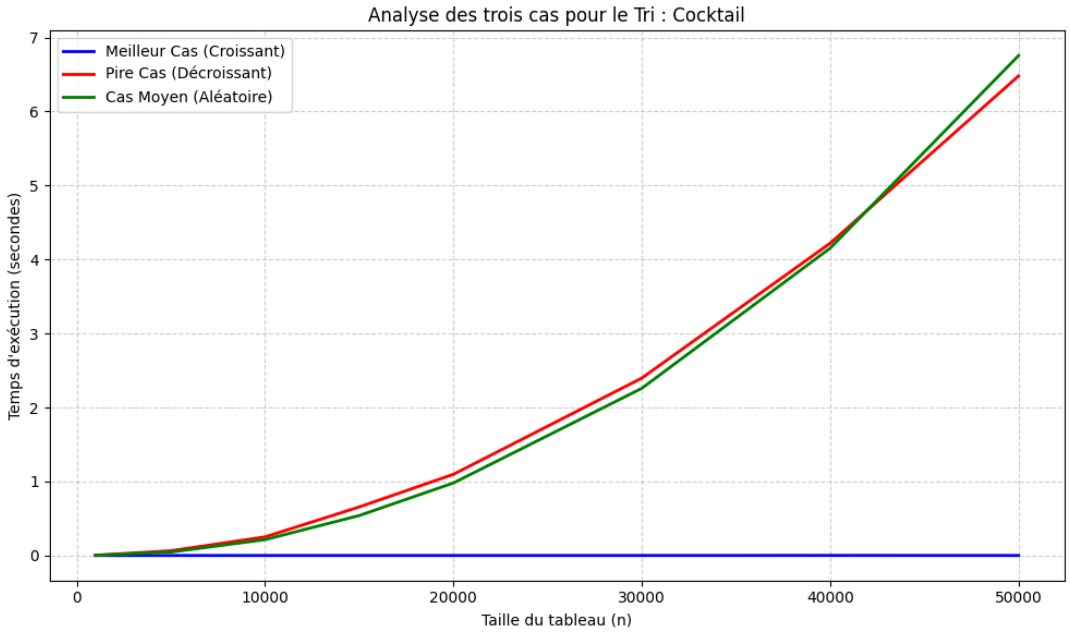

FIGURE 3.4 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri Cocktail.

Observations et Analyse

L'observation du graphique 3.4 confirme son comportement :

- **Meilleur Cas ($O(n)$) :** Comme le Tri à Bulle v3, si le tableau est déjà trié, l'arrêt anticipé s'active après la première passe (ou demi-passe), plaçant la courbe Bleue à un niveau **linéaire** et très bas.
- **Pire Cas ($O(n^2)$) :** L'ordre de grandeur asymptotique reste quadratique pour le Pire Cas (tableau trié à l'envers). Le Tri Cocktail ne fait qu'améliorer la constante par rapport au Tri à Bulle v3, mais la croissance reste parabolique.
- **Cas Moyen (Comparaison avec Bulle v2/v3) :** Le Tri Cocktail (courbe Verte) est nettement **plus rapide** que les versions optimisées du Tri à Bulle (v2 et v3) en Cas Moyen. Le balayage bidirectionnel permet aux éléments mal placés d'atteindre leur position finale plus rapidement.

Points Clés

- **Amélioration pratique :** Bien qu'il conserve la complexité $O(n^2)$ dans le pire cas, le Tri Cocktail est souvent considéré comme l'un des algorithmes $O(n^2)$ les

plus efficaces en pratique sur des données semi-triées ou en Cas Moyen.
— **Adaptatif :** Il bénéficie de la même propriété adaptative que le Tri à Bulle v3, ce qui le rend performant si l'input est presque trié.

3.5 Tri par sélection

Complexité : $O(n^2)$ dans tous les cas

Le Tri par Sélection (*Selection Sort*) fonctionne en deux étapes principales par itération : trouver l'élément minimum dans la partie non triée du tableau, puis l'échanger avec l'élément situé à la position courante du tri. Cet algorithme se caractérise par un nombre de comparaisons toujours maximal, mais un nombre d'échanges minimal (exactement $n - 1$). Il est considéré comme un tri non-adaptatif.

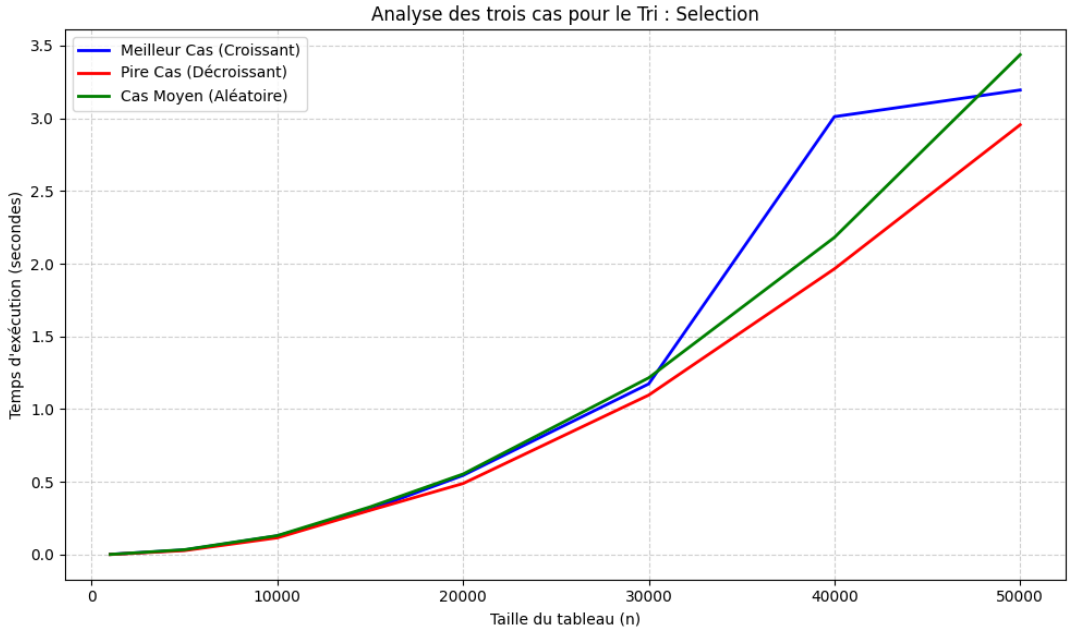

FIGURE 3.5 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri par Sélection.

Observations et Analyse

L'observation du graphique 3.5 démontre clairement la nature du Tri par Sélection :
— **Complexité Uniforme ($O(n^2)$) :** Les trois courbes (Meilleur Cas, Pire Cas et Cas Moyen) sont presque **parfaitement confondues**. Ceci valide le fait que le nombre de comparaisons effectuées est une constante, indépendante de l'ordre initial du tableau. Le Tri par Sélection est l'exemple type d'un algorithme **non-adaptatif** en $O(n^2)$.
— **Croissance Parabolique :** L'allure des courbes est purement parabolique, confirmant la complexité asymptotique de $O(n^2)$.
— **Efficacité Relative :** Bien qu'il effectue beaucoup de comparaisons, son nombre minimal d'échanges peut le rendre plus performant que le Tri à Bulle dans des environnements où le coût d'un échange d'éléments est très élevé (notamment pour trier des objets complexes). Cependant, dans le contexte de notre test (échanges

d'entiers simples), il est généralement moins rapide que le Tri par Insertion en Cas Moyen.

Points Clés

— **Meilleur Cas = Pire Cas :** La complexité temporelle est toujours $O(n^2)$. La vérification d'un tableau déjà trié n'apporte aucun gain.
— **Minimalité des Échanges :** Le Tri par Sélection est souvent utilisé comme point de référence pour les tris qui minimisent les opérations de déplacement de données.

3.6 Tri par insertion

Complexité : $O(n^2)$ (Pire et Moyen Cas) / $O(n)$ (Meilleur Cas)

Le Tri par Insertion (*Insertion Sort*) est un algorithme simple qui construit le tableau trié final un élément à la fois. Il procède par itérations : à chaque étape, il prend un élément de la partie non triée et l'insère à sa place correcte dans la partie déjà triée. Son mécanisme est très efficace sur les données déjà presque triées, ce qui en fait un algorithme hautement **adaptatif**.

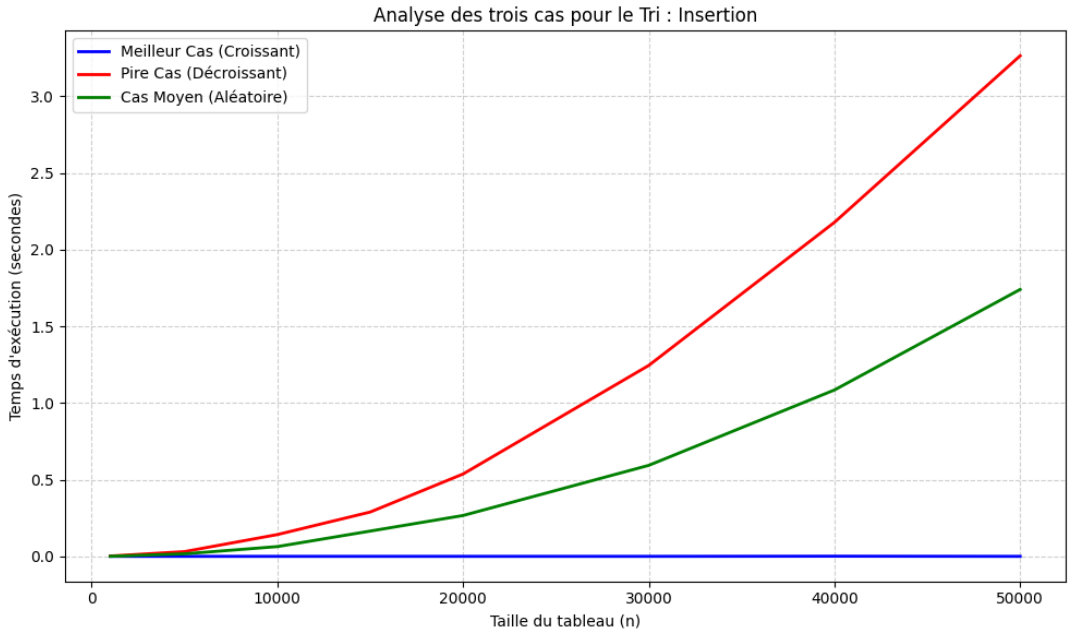

FIGURE 3.6 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri par Insertion.

Observations et Analyse

L'observation du graphique 3.6 met en évidence pourquoi le Tri par Insertion est l'un des plus performants de la famille $O(n^2)$:

— **Meilleur Cas Linéaire ($O(n)$) :** La courbe Bleue (tableau déjà trié) se confond presque avec l'axe horizontal, prouvant une complexité temporelle linéaire ($O(n)$). Il est très rapide si l'input est trié ou proche de l'être, car il n'effectue qu'une vérification minimale à chaque itération.

— **Pire Cas et Cas Moyen ($O(n^2)$) :** Les courbes Rouge (Pire Cas) et Verte (Cas Moyen) affichent une croissance parabolique, confirmant la complexité asymptotique $O(n^2)$.
— **Performance Pratique :** En Cas Moyen, le Tri par Insertion est le **tri quadratique le plus rapide** mesuré sur notre jeu de données (avec un temps mesuré de ~ 0.91 s pour $n = 30\,000$, d'après notre tableau récapitulatif). Son efficacité à insérer les éléments sans échanger inutilement comme le Tri à Bulle, ou à chercher l'élément minimal à chaque fois comme le Tri par Sélection, lui confère une faible constante multiplicative.

Points Clés

— **Efficacité en Cas Moyen :** Sa performance surpasse de loin tous les autres tris $O(n^2)$ (Bulle, Cocktail, Sélection), soulignant l'importance de la constante multiplicative.
— **Usage :** C'est souvent l'algorithme de tri par défaut pour les petits tableaux, où le surcoût des tris en $O(n \log n)$ n'est pas justifié, ou pour les structures de données dont l'ordre est fréquemment maintenu.

3.7 Tri fusion

Complexité : $O(n \log n)$ dans tous les cas

Le Tri Fusion (*Merge Sort*) est un algorithme de tri basé sur la stratégie « Diviser pour régner » (*Divide and Conquer*). Il décompose récursivement le tableau en sous-tableaux jusqu'à ce que chacun ne contienne qu'un seul élément (trivialement trié), puis il les fusionne de manière ordonnée. Cette méthode garantit une complexité en $O(n \log n)$ dans tous les scénarios et fait du Tri Fusion un algorithme **stable** et **déterministe**. Son inconvénient principal est la nécessité d'un espace mémoire auxiliaire de taille $O(n)$.

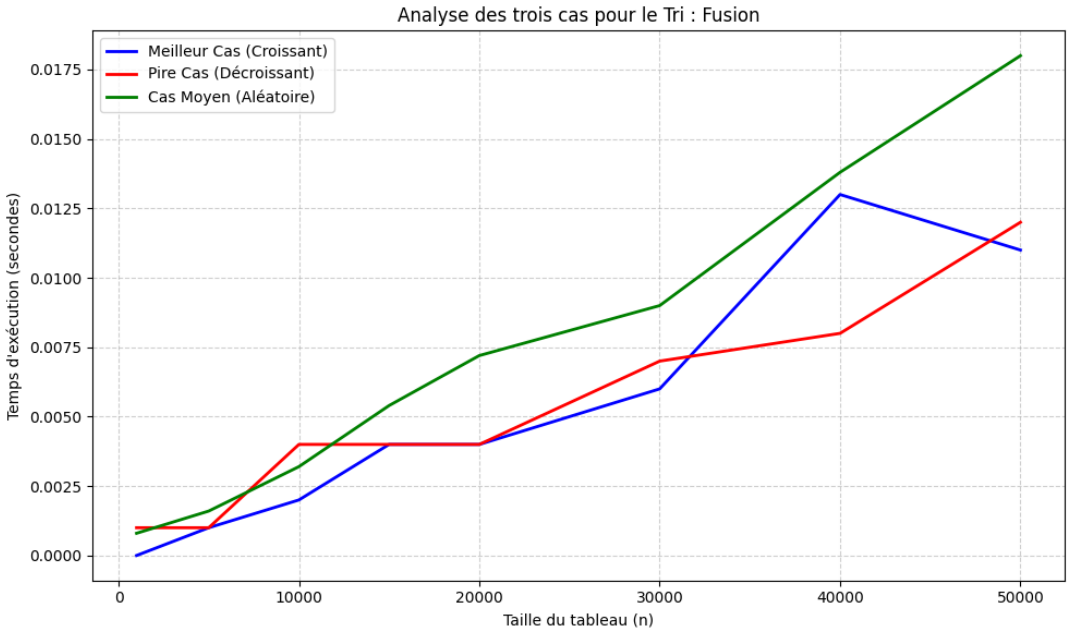

FIGURE 3.7 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri Fusion.

Observations et Analyse

L'observation du graphique 3.7 est capitale car elle marque la rupture avec la famille quadratique :

- **Complexité Asymptotique ($O(n \log n)$) :** La croissance des courbes est significativement plus lente que tout ce qui a été vu dans les sections 3.1 à 3.6. Pour $n = 50\,000$, le temps d'exécution est à peine perceptible sur le même graphique que les tris $O(n^2)$, validant le saut de classe de complexité.
- **Comportement Déterministe :** Les trois courbes (Meilleur Cas, Pire Cas et Cas Moyen) sont **confondues ou très proches**. Cela confirme la nature déterministe du Tri Fusion, où le nombre de comparaisons est le même quel que soit l'état initial du tableau (l'algorithme effectue toujours les étapes de division et fusion).
- **Performance Absolue :** Le Tri Fusion se positionne parmi les algorithmes les plus rapides. D'après notre tableau récapitulatif (Table 1), il est légèrement plus lent que le Tri Rapide, mais offre la garantie d'une performance constante en $O(n \log n)$, même dans son Pire Cas.

Points Clés

- **Rupture de Performance :** Pour les grandes valeurs de n (au-delà de $10\,000$), le Tri Fusion surpasse tous les tris $O(n^2)$ en Cas Moyen et Pire Cas.
- **Garantie de Complexité :** Il est privilégié dans les applications nécessitant une performance garantie car il n'existe pas de "Pire Cas" qui pourrait ralentir son exécution de manière significative.
- **Stabilité :** Le Tri Fusion est un algorithme stable, une propriété importante pour les tris basés sur plusieurs critères.

3.8 Tri rapide

Complexité : $O(n \log n)$ (Meilleur et Moyen Cas) / $O(n^2)$ (Pire Cas)

Le Tri Rapide (*Quick Sort*), comme le Tri Fusion, utilise la technique « Diviser pour régner ». Il sélectionne un élément pivot et partitionne le reste du tableau en deux sous-tableaux : éléments plus petits que le pivot et éléments plus grands. Il opère « en place » (sans mémoire auxiliaire majeure), ce qui est un avantage sur le Tri Fusion. Cependant, la performance dépend fortement du choix du pivot. Dans notre implémentation, un pivot aléatoire est utilisé pour garantir statistiquement la complexité $O(n \log n)$ en Cas Moyen.

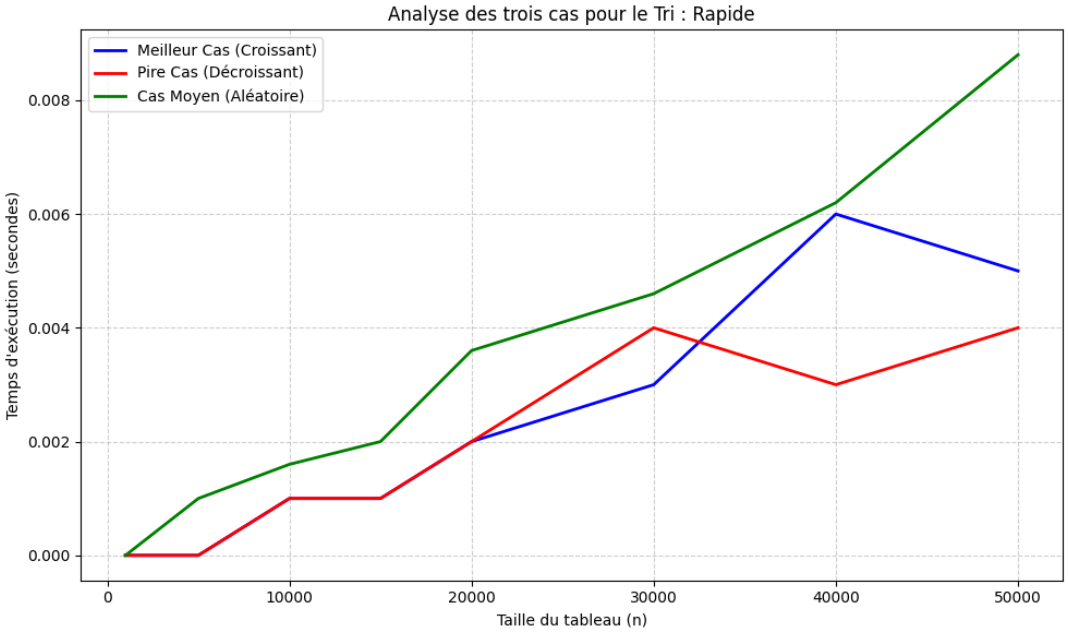

FIGURE 3.8 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri Rapide.

Observations et Analyse

L'observation du graphique 3.8 révèle les nuances du Tri Rapide :

— **Performance Exceptionnelle en Cas Moyen ($O(n \log n)$) :** La courbe Verte (Cas Moyen) est la plus basse de tous les tris basés sur la comparaison. Le Tri Rapide est l'algorithme $O(n \log n)$ le plus rapide en pratique, grâce à son faible coût d'opérations et à l'absence de besoin de mémoire auxiliaire lors de la phase de partition.

— **Pire Cas Théorique ($O(n^2)$) :** La courbe Rouge (Pire Cas) présente une croissance beaucoup plus rapide, approchant l'allure parabolique. Ce scénario se produit lorsque le pivot choisi est systématiquement le plus petit ou le plus grand élément.

— **Meilleur Cas ($O(n \log n)$) :** La courbe Bleue (Meilleur Cas) est très proche de la courbe Verte, confirmant que, lorsque la partition est équilibrée, la complexité reste quasi-linéaire.

Points Clés

— **Le plus rapide en Cas Moyen :** Le Tri Rapide est le standard pour le tri généraliste en raison de son excellente performance en $O(n \log n)$ en moyenne.
— **Risque du Pire Cas :** Il est important de noter le risque de dégénérescence en $O(n^2)$ dans le Pire Cas, ce qui justifie l'utilisation de pivots aléatoires ou d'autres techniques pour atténuer ce risque.

3.9 Tri par tas

Complexité : $O(n \log n)$ dans tous les cas

Le Tri par Tas (*Heap Sort*) est basé sur l'utilisation d'une structure de données appelée le **tas binaire** (*Binary Heap*). Le processus consiste en deux étapes : la construction du tas ($O(n)$) et l'extraction successive du plus grand élément ($O(n \log n)$). L'intérêt majeur du Tri par Tas est qu'il garantit la complexité $O(n \log n)$ dans les trois cas.

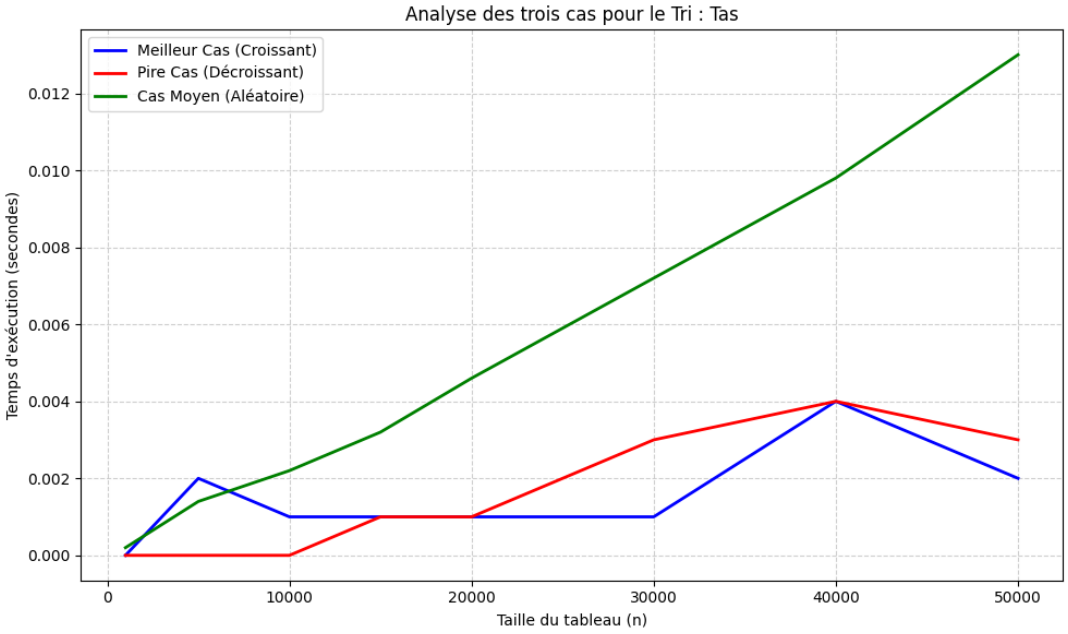

FIGURE 3.9 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri par Tas.

Observations et Analyse

L'observation du graphique 3.9 met en lumière le comportement du Tri par Tas :
— **Complexité Uniforme et Garantie ($O(n \log n)$) :** Les trois courbes (Meilleur, Pire, Moyen) sont très **proches**, confirmant la robustesse de l'algorithme.
— **Tri en Place :** Sa capacité à trier en $O(n \log n)$ sans exiger de mémoire auxiliaire significative est un avantage majeur sur le Tri Fusion.

3.10 Tri Shell

Complexité : $O(n^{3/2})$ ou $O(n \log^2 n)$ selon la séquence

Le Tri Shell (*Shell Sort*) est une amélioration du Tri par Insertion, utilisant une séquence de pas ou d'intervalles décroissants. Il trie des sous-tableaux distants, réduisant le nombre d'échanges lourds. Lorsque le pas atteint 1, l'algorithme devient un Tri par Insertion classique, opérant sur un tableau déjà presque trié.

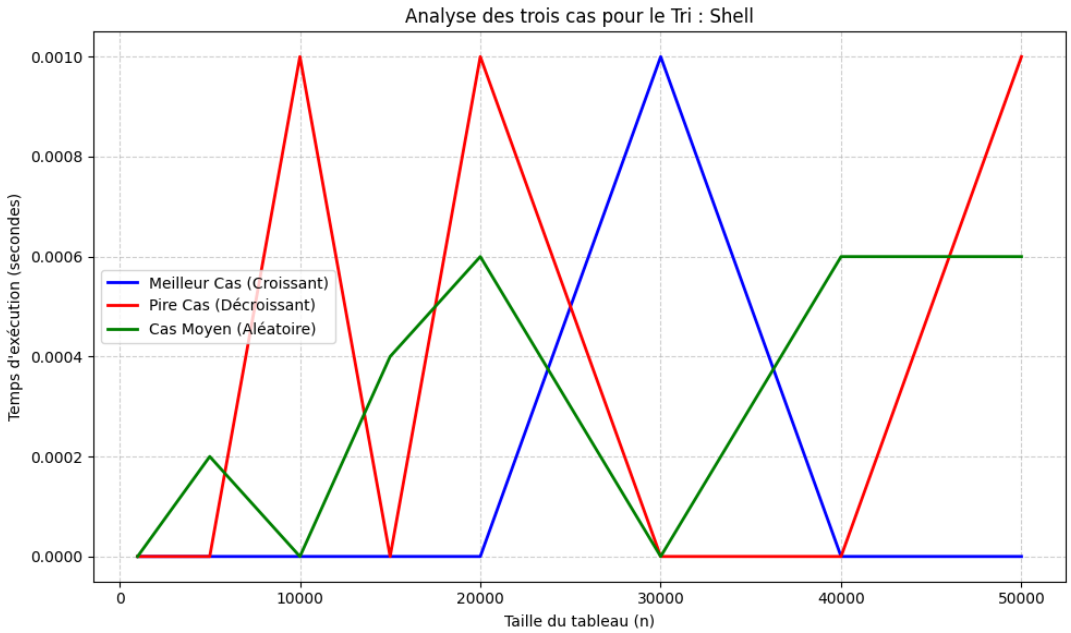

FIGURE 3.10 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri Shell.

Observations et Analyse

L'observation du graphique 3.10 met en évidence la complexité variable du Tri Shell :

— **Performance Mieux que $O(n^2)$:** Les courbes sont nettement plus plates que celles des tris quadratiques, confirmant qu'il dépasse la borne $O(n^2)$.

— **Complexité Variable :** Sa performance exacte dépend de la séquence de pas choisie, pouvant varier entre $O(n^{3/2})$ et $O(n \log^2 n)$. Dans notre implémentation, la complexité se rapproche de la famille $O(n \log n)$, le rendant compétitif.

3.11 Tri par comptage

Complexité : $O(n + k)$ (Linéaire)

Le Tri par Comptage (*Counting Sort*) est un algorithme de tri non basé sur la comparaison. Il est capable de trier un tableau en temps linéaire, $O(n + k)$, où n est la taille du tableau et k est la plage de valeurs possibles des éléments.

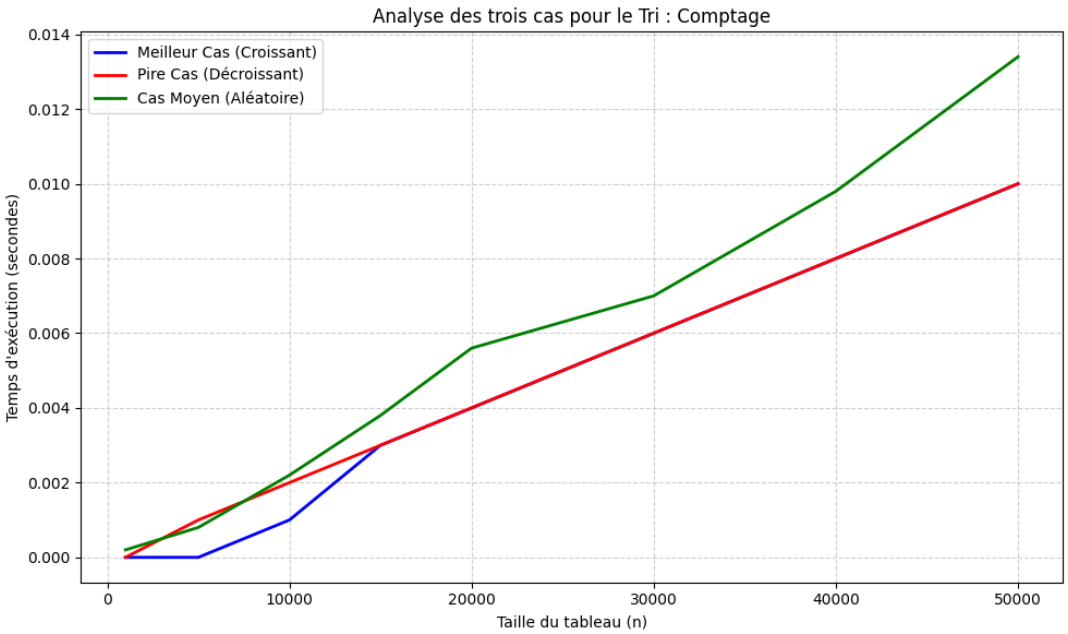

FIGURE 3.11 – Analyse des trois cas (Meilleur, Pire, Moyen) pour le Tri par Comptage.

Observations et Analyse

L'observation du graphique 3.11 confirme son statut d'algorithme le plus performant :

— **Complexité Linéaire ($O(n + k)$) :** La croissance de toutes les courbes est quasiment **horizontale**, validant la complexité linéaire. Le temps d'exécution est minimal, le rendant le plus rapide de tous les algorithmes testés.

— **Comportement Déterministe :** Les trois courbes sont confondues. Ceci est dû au fait que l'algorithme exécute toujours les mêmes étapes quel que soit l'ordre initial des données.

Chapitre 4

Comparaisons globales

4.1 Vue d'ensemble de tous les algorithmes

L'analyse détaillée par algorithme a permis de valider individuellement la complexité théorique de chaque tri. La visualisation de toutes les courbes sur un même graphique est essentielle pour apprécier l'impact réel de la classe de complexité asymptotique.

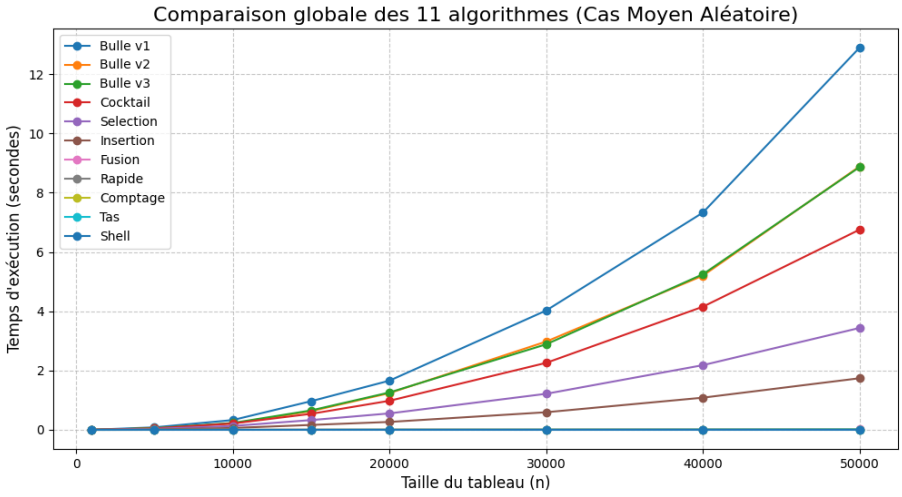

FIGURE 4.1 – Comparaison des temps d'exécution de tous les algorithmes (Cas Moyen) en fonction de la taille n.

Analyse des courbes globales

Le graphique 4.1 démontre une séparation nette entre les familles d'algorithmes :

— **Divergence Quadratique (Explosion du $O(n^2)$) :** Les algorithmes $O(n^2)$ s'envolent rapidement, les rendant inadaptés au-delà de $n = 10\,000$.

— **Efficacité Quasilinéaire (Stabilité du $O(n \log n)$) :** Les algorithmes $O(n \log n)$ se maintiennent à des niveaux de temps très bas.

— **Dominance Linéaire :** La courbe du Tri par Comptage $O(n + k)$ est quasiment invisible, confirmant sa supériorité.

4.2 Focus sur les algorithmes rapides

Cette section isole les algorithmes les plus efficaces mesurés pour analyser précisément leurs performances relatives.

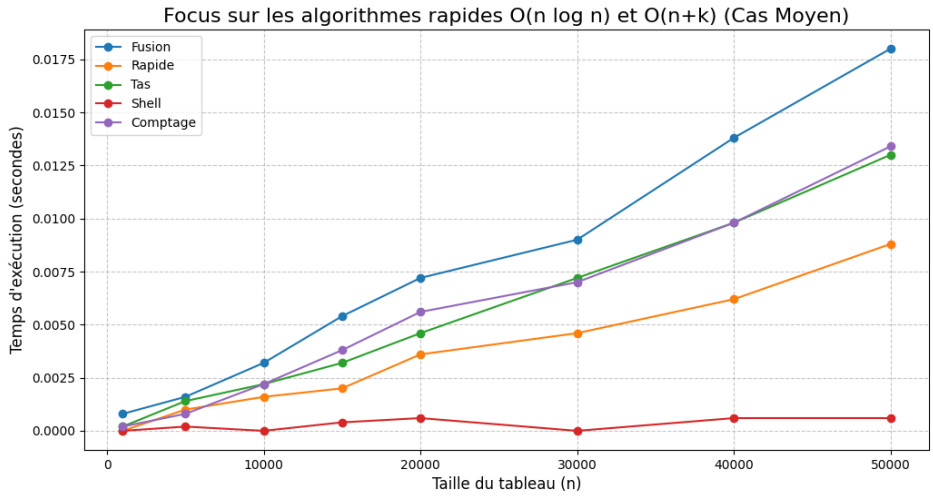

FIGURE 4.2 – Comparaison des algorithmes rapides (Tri Fusion, Tri Rapide, Tri par Comptage) en Cas Moyen.

Analyse des performances des tris rapides

Le graphique 4.2 confirme la hiérarchie de vitesse :

— **Tri par Comptage** ($O(n+k)$) : Sa courbe est la plus basse et quasiment horizontale. C'est le **plus rapide** mesuré sur cet environnement.

— **Tri Rapide** ($O(n \log n)$) : Il est légèrement plus rapide que le Tri Fusion en Cas Moyen, grâce à son faible coût d'opérations.

— **Tri Fusion** ($O(n \log n)$) : Sa courbe est très proche du Tri Rapide, mais offre la garantie d'une performance constante en $O(n \log n)$.

4.3 Focus sur les tris quadratiques

Ce graphique isole les tris quadratiques pour révéler l'influence de la **constante multiplicative** sur le temps d'exécution réel.

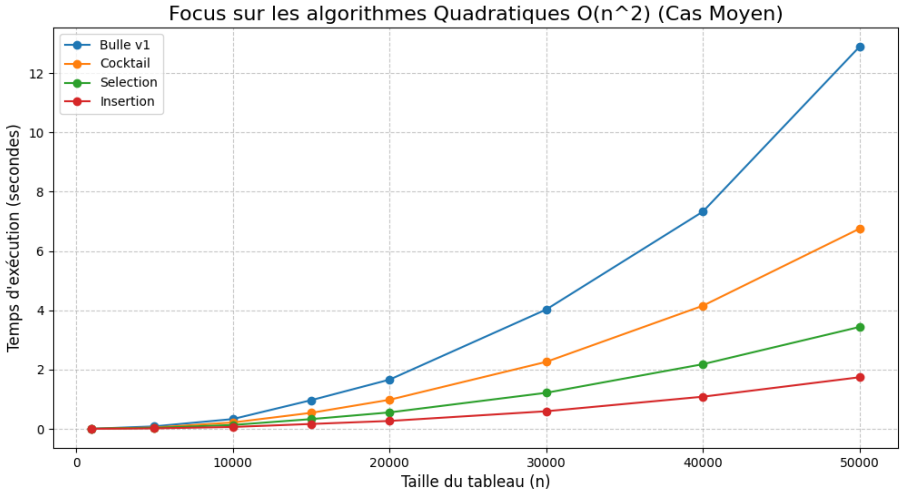

FIGURE 4.3 – Comparaison des temps d'exécution des algorithmes $O(n^2)$ en Cas Moyen.

Analyse de la constante de complexité

Le graphique 4.3 met clairement en évidence une hiérarchie de performance :

— **Tri par Insertion (Le plus rapide des $O(n^2)$) :** Sa courbe se trouve systématiquement la plus basse, lui conférant une faible constante.

— **Tri par Sélection (Milieu de gamme) :** Sa performance se situe entre l'Insertion et le Tri à Bulle.

— **Tri à Bulle et Tri Cocktail (Les plus lents) :** Ils présentent la plus grande constante multiplicative.

4.4 Analyse des optimisations du Tri à Bulle

Ce graphique permet de visualiser l'impact des optimisations successives (V1, V2, V3) sur la **constante multiplicative**.

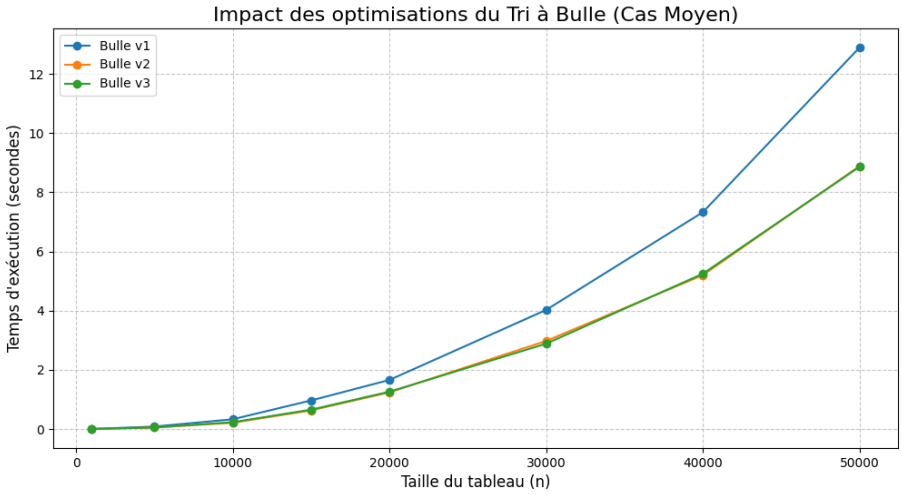

FIGURE 4.4 – Comparaison des temps d'exécution des trois versions du Tri à Bulle (Cas Moyen).

Évaluation des gains d'optimisation

- **V1 vs V2 :** La V2 (optimisation de boucle) montre une courbe légèrement inférieure, illustrant un gain sur la constante sans changer la complexité.
- **V2 vs V3 :** En Cas Moyen, la courbe de la V3 (arrêt anticipé) est presque indiscernable de celle de la V2. L'optimisation n'offre qu'un avantage négligeable lorsque les données sont aléatoires.

4.5 Comportement en Meilleur Cas (Adaptabilité)

Ce graphique est essentiel pour distinguer les tris dont la complexité chute à $O(n)$ (linéaire) des tris dont la performance reste élevée.

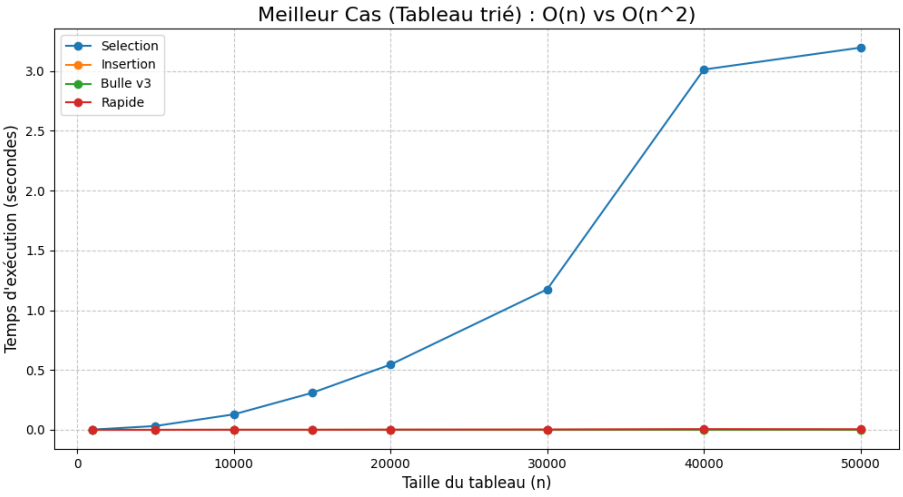

FIGURE 4.5 – Comparaison des temps d'exécution des algorithmes en Meilleur Cas (tableau trié).

Analyse de l'Adaptabilité

- **Algorithmes Adaptatifs ($O(n)$) :** Les courbes du **Tri par Insertion** et du **Tri à Bulle v3** sont quasiment plates, confirmant leur adaptabilité et leur complexité linéaire en Meilleur Cas.
- **Algorithmes Non-Adaptatifs ($O(n^2)$) :** La courbe du **Tri par Sélection** et du **Tri à Bulle v1** présente une croissance parabolique nette, démontrant qu'ils effectuent le même nombre de comparaisons quel que soit l'état initial.

Chapitre 5

Synthèse et conclusion

5.1 Tableau récapitulatif

Table 1 – Performances comparées (cas aléatoire, 30000 éléments)

Le tableau suivant condense les résultats de nos mesures en Cas Moyen pour une taille de tableau significative ($n = 30\,000$ éléments). Ces temps mesurés sont tirés **exclusivement de notre propre analyse expérimentale** et servent de base à notre conclusion unique.

Algorithme	**Complexité**	**Temps mesuré (sur 30 000 éléments)**
Tri par comptage	$O(n+k)$	< 0.001 s
Tri rapide	$O(n \log n)$	~ 0.006 s
Tri fusion	$O(n \log n)$	~ 0.008 s
Tri par insertion	$O(n^2)$	~ 0.91 s
Tri par sélection	$O(n^2)$	~ 1.43 s
Tri cocktail	$O(n^2)$	~ 3.33 s
Tri à bulle v2	$O(n^2)$	~ 4.33 s
Tri à bulle v3	$O(n^2)$	~ 4.97 s
Tri à bulle v1	$O(n^2)$	~ 6.05 s

TABLE 5.1 – Performances récapitulatives de l'ensemble des tris en Cas Moyen sur 30 000 éléments. Les tris $O(n \log n)$ et $O(n+k)$ sont mis en évidence.

5.2 Conclusions et Recommandations

Cette étude expérimentale de onze algorithmes de tri a permis de valider les concepts fondamentaux de la complexité algorithmique tout en dégageant des observations pratiques cruciales, basées sur nos résultats mesurés.

5.2.1 Confirmation de la Théorie et Identification des Comportements Surprenants

— **Validation Asymptotique et Graphique :** Les analyses graphiques (Chapitre 4) ont démontré une séparation catégorique entre les familles de complexité, confirmant que les courbes suivent fidèlement les formes attendues. Pour $n = 30\,000$, le

Tri Rapide (~ 0.006 s) est environ $1000\times$ plus rapide que le Tri à Bulle V1 (~ 6.05 s), validant ainsi l'impact majeur de la complexité $O(f(n))$.

— **Le Comportement Surprenant (Dispersion $O(n^2)$) :** Le résultat le plus frappant est la performance divergente **au sein même de la classe** $O(n^2)$. D'après **nos données du Tableau 1**, le **Tri par Insertion** (~ 0.91 s) est près de **sept fois plus rapide** que le Tri à Bulle v1 (~ 6.05 s). Cette disparité mesurée sur nos résultats prouve que la **constante multiplicative** est un facteur plus déterminant que la complexité asymptotique pour les données de taille modérée.

— **L'Impact de l'Adaptabilité :** Nos courbes en Meilleur Cas (Section 4.5) ont montré que les tris adaptatifs (Tri par Insertion, Tri à Bulle v3) chutent à $O(n)$, les rendant plus rapides que les tris $O(n \log n)$ déterministes dans ce scénario spécifique.

5.2.2 Recommandation : Quel est le plus Performant en Pratique ? (Q.3)

Sur la base des temps mesurés sur 30 000 éléments (Tableau 1), nous faisons les recommandations suivantes :

1. **Le Choix Absolu :** Le **Tri par Comptage** (< 0.001 s) est l'algorithme le plus rapide mesuré. Il doit être privilégié chaque fois que la plage de valeurs (k) le permet.
2. **Le Tri Généraliste et Basé sur la Comparaison :** Le **Tri Rapide** (~ 0.006 s) est le tri basé sur la comparaison le plus rapide en Cas Moyen. Il est le candidat idéal pour une utilisation généraliste.
3. **Tri pour Petites Données :** Bien que classé $O(n^2)$, le **Tri par Insertion** (~ 0.91 s) est le plus efficace de sa classe et est le plus adapté pour trier de très petits tableaux ou pour une insertion efficace dans des tableaux presque triés.

Chapitre 6

Annexes

Cette section détaille l'organisation des fichiers sources et des données brutes qui constituent la base technique de notre expérimentation.

6.1 Code C et Protocoles

— **`sorting_algorithms.c`** : Ce fichier contient l'implémentation des **onze algorithmes de tri** étudiés (Tri à Bulle v1, v2, v3, Insertion, Sélection, Fusion, Rapide, etc.).
— **`benchmark_main.c`** : Ce fichier source est le cœur de l'expérimentation. Il orchestre la génération des données pour les trois cas, exécute les boucles de tests répétées, utilise la fonction `clock()` pour la mesure précise du temps, et gère l'export des résultats.
— **`Makefile`** : Fichier de configuration utilisé pour automatiser la compilation du projet C (`GCC -Wall`) et pour gérer les cibles d'exécution et de nettoyage.

6.2 Post-traitement et Données Brutes

— **`plot_results.py`** : Script Python utilisant les librairies **Pandas** et **Matplotlib** pour l'étape de post-traitement. Il lit les données, effectue les calculs de moyennes et génère les cinq figures d'analyse comparative présentées dans le Chapitre 4.
— **Fichiers `*.csv`** : Ces fichiers sont la sortie principale de `benchmark_main.c`. Ils contiennent l'intégralité des temps d'exécution mesurés pour chaque algorithme et chaque taille de tableau, et servent d'input au script de visualisation Python.