République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الاعلام الالي سابقا)
École nationale Supérieure
d'Informatique ex. INI (Institut
National de formation en Informatique)

Rapport de Travail Pratique (TP)

Module : CRP - Complexité et Résolution de Problème

2ème année Cycle Supérieur (2CS)

Option : Systèmes Intelligents et Données (SID)

Groupe : 2SD1

Thème :

Analyse de complexité et optimisation algorithmique

Application à la vérification de primalité

Réalisé par :

- ABOUD Ibrahim
- BOUYAKOUB Rayane

Proposé par :

- Mr. KECHID Amine

Année universitaire : 2024-2025

Table des matières

Liste des figures

Liste des tableaux

1. Introduction générale

La théorie de la complexité est un domaine fondamental en informatique qui permet d'évaluer et de comparer l'efficacité des algorithmes. Cette analyse est cruciale car elle nous permet de prédire le comportement d'un algorithme en termes de ressources (temps d'exécution, espace mémoire) en fonction de la taille des données d'entrée. La O-Notation est particulièrement utilisée pour exprimer cette complexité.

Dans ce rapport, nous nous intéressons à l'étude de la complexité algorithmique à travers un cas pratique : la vérification de la primalité d'un nombre. Cette problématique, bien que simple à comprendre, offre un excellent terrain d'exploration pour l'analyse de complexité et l'optimisation algorithmique.

Notre approche se structure en trois parties principales :

- Une première implémentation naïve de l'algorithme.
- Une première optimisation réduisant l'intervalle de recherche à N/2.
- Une optimisation plus poussée limitant la recherche à la racine carrée de N.

Pour chaque partie, nous procédons à une analyse théorique suivie d'une validation expérimentale, permettant ainsi de confronter la théorie à la pratique. Une analyse comparative approfondie des trois versions de l'algorithme est ensuite abordée, mettant en évidence les gains de performance obtenus à chaque amélioration.

2. Objectifs

Les objectifs de ce TP sont les suivants :

- Étudier la relation entre la complexité théorique et la complexité expérimentale des algorithmes à travers l'exemple de la vérification de primalité.

- Mettre en œuvre différentes optimisations d'un algorithme de vérification de primalité et analyser leur impact sur les performances.

- Valider expérimentalement les complexités théoriques calculées en mesurant les temps d'exécution.

- Comparer les performances des différentes versions optimisées de l'algorithme pour démontrer l'importance de l'optimisation algorithmique.

3. Partie 1 : Algorithme naïf de vérification de primalité

3.1. Conception de l'algorithme

Pour vérifier si N est un nombre premier, on parcourt les entiers allant de 2 jusqu'à N-1, si l'un de ses nombres divise N, alors N n'est pas premier. Dans le cas contraire, c'est-à-dire, si aucun nombre de cet intervalle divise N, alors N est premier. Cas particulier **:** Si N vaut 1 ou 0, l'algorithme doit indiquer que N n'est pas premier.

***Algorithme* 1** : *Algorithme naïf de vérification de primalité*

```
Variables N, i: Entier
Variable trouv: Booléen
Début
    Lire(N)
    Si (N ≤ 1) Alors
        Afficher ("N n'est pas premier")
    Fin Si
    Sinon
        i ← 2
        trouv ← Faux
        Tant que (trouv = Faux  ET  i < N ) Faire
            Si (i mod N = 0) Alors
                trouv ← Vrai
            Fin Si
            Sinon
                i ← i + 1
            Fin Sinon
        Fin Tant que
        Si (trouv = Vrai) Alors
            Afficher ("N est premier")
        Sinon
            Afficher ("N n'est pas premier")
        Fin Sinon
    Fin Sinon
Fin
```

3.2. Analyse théorique de la complexité

i. Meilleur cas

- **Premier meilleur cas**

Le meilleur cas se produit lorsque N est divisible par 2, c'est-à-dire lorsque N est un nombre pair. Dans ce cas, le programme se termine après l'exécution d'une seule itération de la boucle (i = 2). Ainsi, dans le meilleur cas, la complexité temporelle est O(1).

- **Deuxième meilleur cas**

Lorsque N est inférieur ou égale à 1, le programme entre directement dans le premier bloc conditionnel (if), affiche que N n'est pas premier, puis s'arrête. Dans ce cas, la complexité est O(1).

ii. Pire cas

Le pire cas survient lorsque N est un nombre premier, ce qui signifie qu'il n'a pas de diviseurs autres que lui-même et 1. Dans ce cas, la boucle va exécuter $N-1 - 2 +1 = N - 2$ itérations (en partant de i = 2 jusqu'à $i = N-1$), car il faut vérifier chaque nombre pour confirmer que N est premier. Ainsi, la complexité dans le pire cas est O(N), car le nombre d'itérations est proportionnel à N lorsque N est un nombre premier.

iii. Conclusion

- **Meilleur cas :** O(1).
- **Pire cas :** O(N).

La complexité est donc linéaire dans le pire cas, mais constante dans le meilleur cas.

3.3. Implémentation de l'algorithme

Pour l'implémentation de l'algorithme, nous avons opté pour le langage C, un langage de bas niveau, proche du matériel, et caractérisé par sa rapidité d'exécution, ce qui permet d'obtenir des mesures des temps d'exécution très précises.

Figure 1: Logo du langage de programmation C.

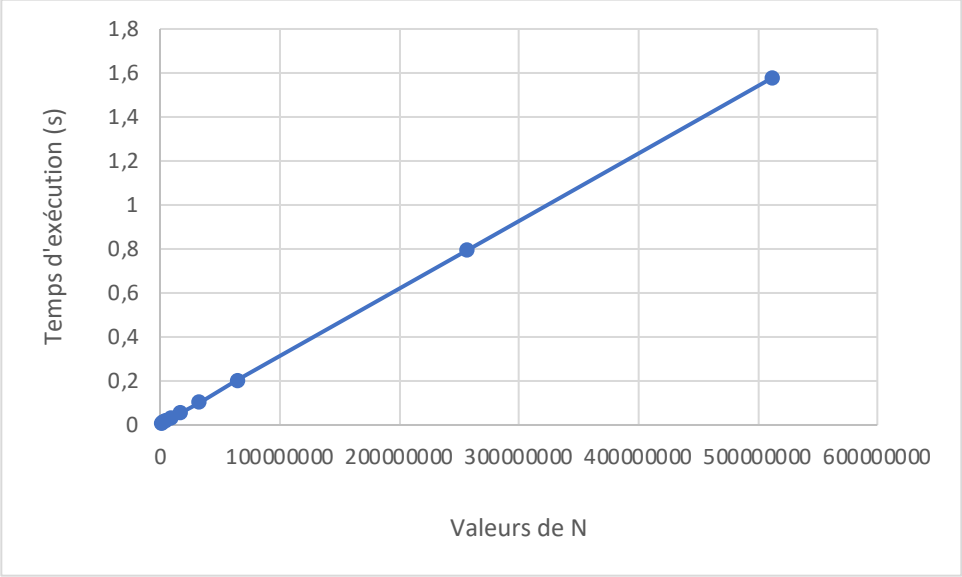
3.4. Mesures expérimentales

Pour obtenir une estimation précise du temps d'exécution de notre programme, nous avons intégré le bloc de vérification de primalité de N dans une boucle dont le nombre d'itérations peut être ajusté par l'utilisateur en fonction des besoins en précision des mesures. Dans notre cas, nous avons défini le nombre d'itérations à 1000. Ensuite, nous cumulons les temps d'exécution et divisons par le nombre d'itérations pour obtenir une moyenne. Les résultats des mesures des temps d'exécution sont résumés dans le tableau suivant.

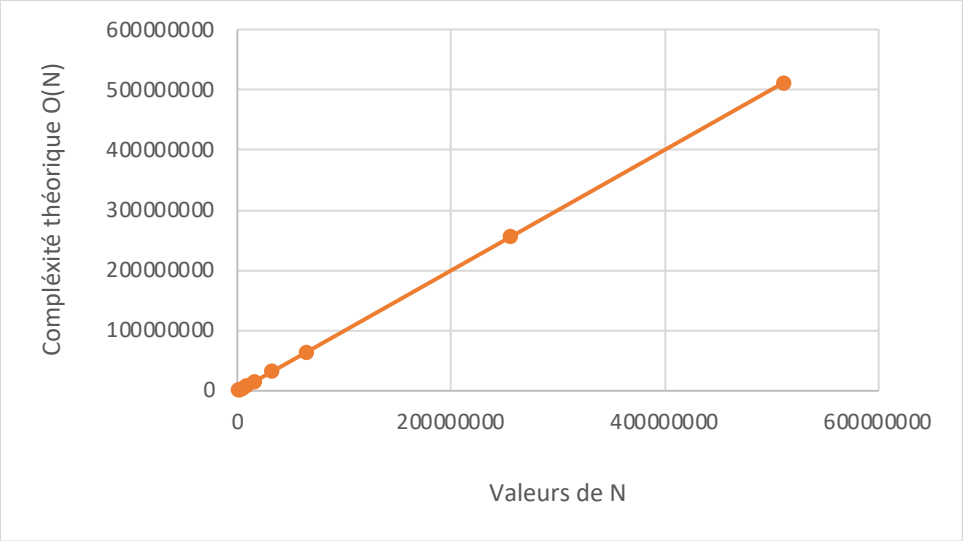
N	Nombre premier ?	Temps d'exécution (s)
1 000 003	Oui	0.003121
2 000 003	Oui	0.006290
4 000 037	Oui	0.012432
8 000 009	Oui	0.024764
16 000 057	Oui	0.049475
32 000 011	Oui	0.098741
64 000 031	Oui	0.197166
256 000 001	Oui	0.788759
512 000 009	Oui	1.576491
128 000 03	Non (467 est un diviseur)	0.000007
1024 000 0009	Non (37 est un diviseur)	0.000005
2018 000 011	Non (17 est un diviseur)	0.000004

Tableau 1: Temps d'exécution de l'algorithme naïf de vérification de primalité pour différentes valeurs de N.

3.5. Représentation graphique des résultats



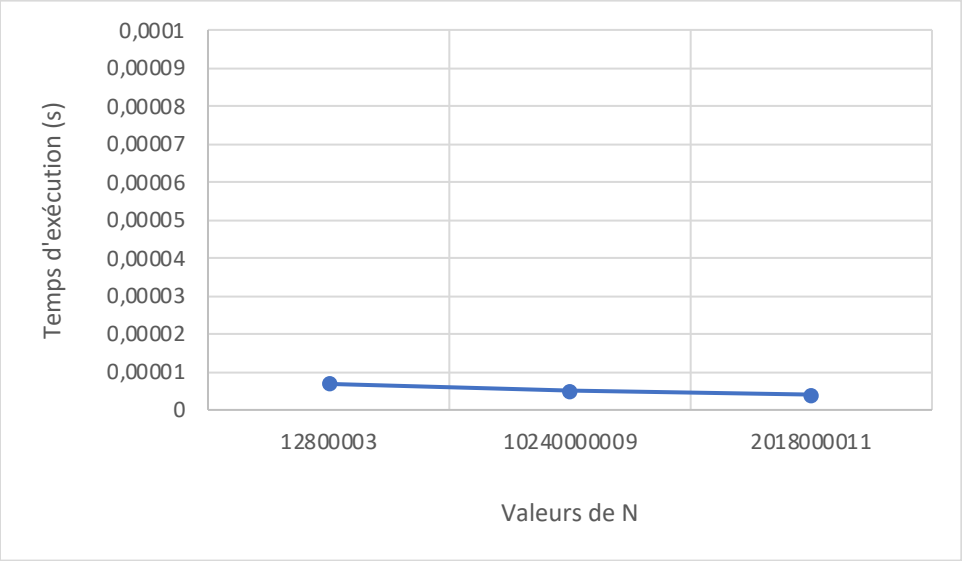
Temps d'exécution observés dans le pire cas (algorithme naïf).



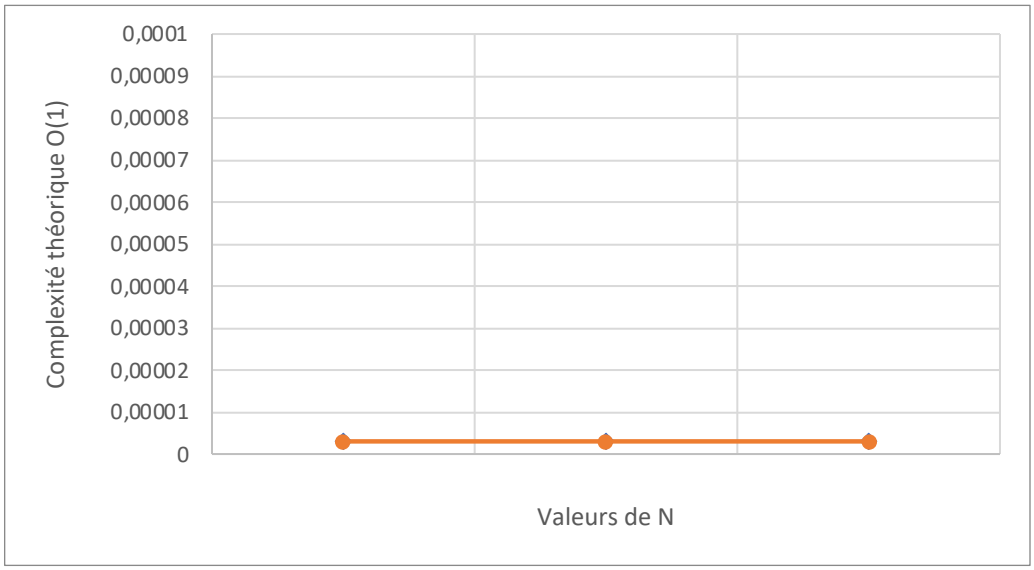
Complexité théorique O(N) dans le pire cas (algorithme naïf).

Figure 2: Représentation des temps d'exécution et de la complexité théorique dans le pire cas (algorithme naïf).

Pour représenter la complexité théorique dans le meilleur cas O(1), nous avons mesuré le temps d'exécution de notre programme pour le cas où N est pair. Ce temps d'exécution est d'environ 0,000003 secondes.



Temps d'exécution observés dans le meilleur cas (algorithme naïf).



Complexité théorique O(1) dans le meilleur cas (algorithme naïf).

Figure 3: Représentation des temps d'exécution et de la complexité théorique dans le meilleur cas (algorithme naïf).

3.6. Analyse des temps d'exécution

Les mesures de temps dans le tableau correspondent au meilleur et au pire cas, selon que N soit premier ou non.

- **Nombres premiers (Pire cas)**

Pour les nombres premiers, le temps d'exécution augmente avec l'augmentation de la taille de l'entier N, par exemple, de 0.003121 secondes pour N=1,000,003 à 1.576491 secondes pour N=512,000,009. De plus, l'augmentation du temps d'exécution est proportionnelle (linéaire) avec l'augmentation de la taille de l'entier N, comme le montre la courbe bleue dans la figure 2. Cette augmentation linéaire du temps d'exécution, correspond bien à la complexité O(N) attendue dans le pire cas.

- **Nombre non premiers (Meilleur cas)**

Pour les nombres non premiers, le temps d'exécution reste quasi constant et très faible (proche de zéro), même avec l'augmentation de la taille de N, comme le démontre la courbe bleue du graphique de droite de la figure 3. Par exemple, pour N=2,018,000,011, le temps d'exécution est de 0,000004 secondes, tandis qu'il est de 0,000005 secondes pour N=1,024,000,009. Cette stabilité dans les temps d'exécution confirme la complexité O(1) théorique dans le meilleur cas, qui demeure constante indépendamment de la taille de N.

3.7. Observations sur les données et les mesures

Pour les nombres premiers

On observe une croissance quasi linéaire du temps d'exécution.
Le temps double approximativement quand N double. Par exemple :

- N=4000037 → 0.012432s
- N=8000009 → 0.024764s (≈2x)
- N=16000057 → 0.049475s (≈2x)

Pour les nombres non premiers avec le premier diviseur trouvé étant petit (meilleur cas)

- Les temps d'exécution sont très faibles (≈0.000004-0.000007s).
- Ils restent pratiquement constants quelle que soit la taille de N.

3.8. Comparaison entre la complexité théorique et la complexité expérimentale

- **Comparaison dans le pire cas**

La figure 2 montre que la courbe représentant le temps d'exécution expérimental suit la même tendance linéaire que celle représentant la complexité théorique O(N), Ceci confirme que dans le pire cas, la complexité expérimentale de l'algorithme correspond bien à sa complexité théorique O(N).

- **Comparaison dans le meilleur cas**

Les résultats expérimentaux sont en accord avec notre analyse théorique de O(1). En effet, malgré l'utilisation de très grands nombres (allant jusqu'à 1,0240,000,009), les temps d'exécution restent remarquablement constants (entre 0,000004s et 0,000007s). Ceci s'explique par la présence de petits diviseurs (17, 37 et 467) qui sont trouvés rapidement, conduisant à un arrêt rapide de l'algorithme.

Cette analyse est confirmée visuellement par les deux courbes de la figure 3: la courbe orange montre la complexité théorique O(1) avec une droite horizontale à 0.000003 secondes, tandis que la courbe bleue représente les temps d'exécution expérimentaux qui suivent également une droite quasi horizontale proche de 0. Cette proximité entre les deux courbes, aussi bien dans leur forme (horizontale) que dans leurs valeurs (proches de zéro), illustre bien le comportement en O(1) attendu dans le meilleur cas.

Bien que ces cas ne soient pas des nombres pairs (le meilleur cas théorique), ils démontrent un comportement similaire en termes de performance, confirmant ainsi notre analyse de complexité O(1) dans le meilleur cas.

- **Conclusion**

En résumé, les résultats expérimentaux confirment parfaitement l'analyse théorique de la complexité. La complexité est de O(1) dans le meilleur cas et elle vaut O(N) dans le pire cas.

4. Partie 2 : Première amélioration de l'algorithme de vérification de primalité

4.1. Conception de l'algorithme

On garde le même principe que dans le premier algorithme, la seule différence étant qu'on teste les entiers de 2 à N/2 au lieu de tester tous les entiers de 2 à N-1.

Algorithme 2 : *Première amélioration de l'algorithme de vérification de primalité*

```
Variables N, i: Entier
Variable trouv: Booléen
Début
    Lire(N)
    Si (N ≤ 1) Alors
        Afficher ("N n'est pas premier")
    Fin Si
    Sinon
        i ← 2
        trouv ← Faux
        Tant que (trouv = Faux  ET  i ≤ N/2 ) Faire
            Si (i mod N = 0) Alors
                trouv ← Vrai
            Fin Si
            Sinon
                i ← i + 1
            Fin Sinon
        Fin Tant que
        Si (trouv = Vrai) Alors
            Afficher ("N est premier")
        Sinon
            Afficher ("N n'est pas premier")
        Fin Sinon
    Fin Sinon
Fin
```

4.2. Analyse théorique de la complexité

i. Meilleur cas

Le meilleur cas sera identique à celui du premier algorithme: lorsque N est inférieur ou égal à 1, ou lorsque N est divisible par 2. Ainsi, dans le meilleur cas, la complexité temporelle est O(1).

ii. Pire cas

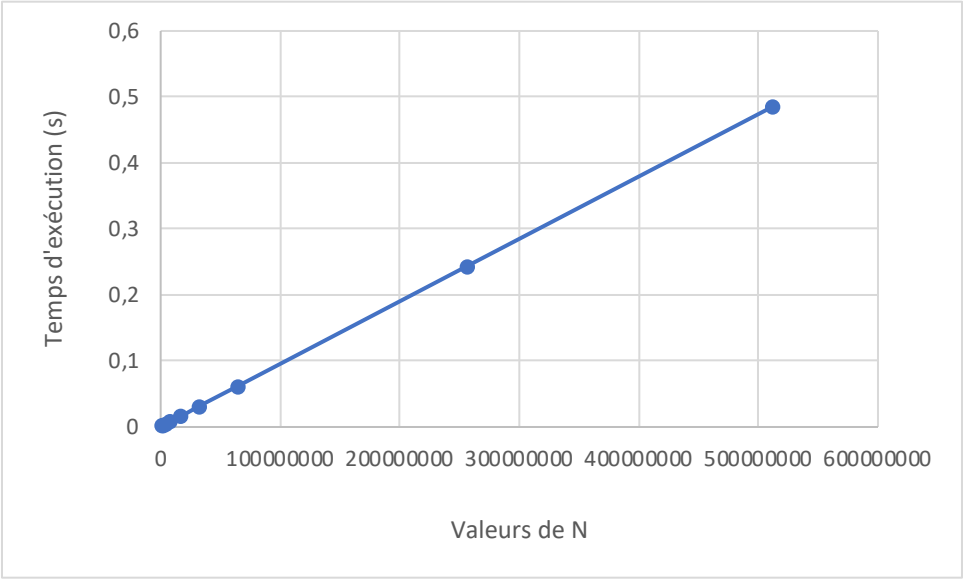
Le pire cas survient lorsque N est un nombre premier, car la boucle devra vérifier tous les entiers i de 2 à N/2 pour confirmer que N est premier. La boucle s'exécute alors jusqu'à ce que i atteigne N/2. Cela signifie que, dans le pire des cas, la boucle s'exécute N/2 - 2 + 1 = (N/2 - 1) fois. Ainsi, la complexité dans le pire cas est O(N), car le nombre d'itérations est proportionnel à N.

4.3. Mesures expérimentales

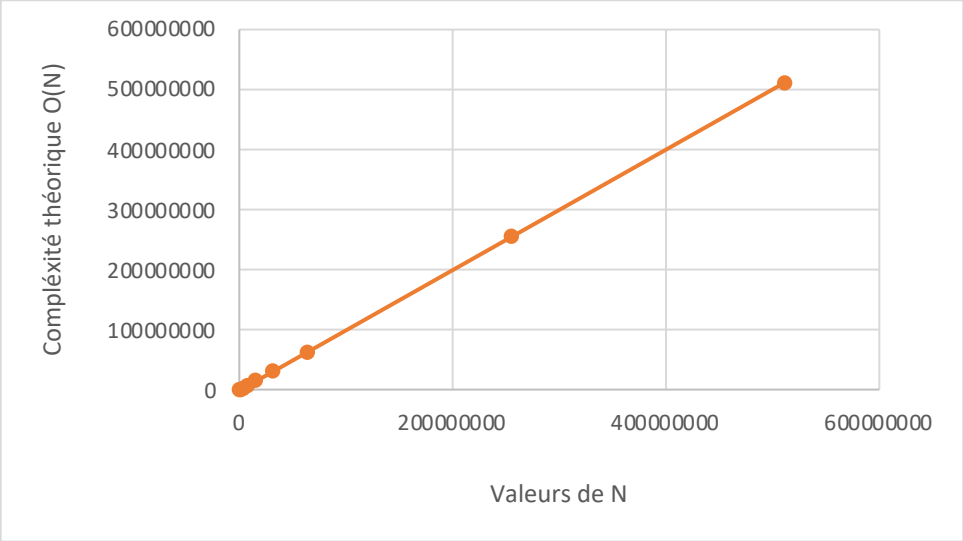
N	Nombre premier ?	Temps d'exécution (s)
1 000 003	Oui	0.000999
2 000 003	Oui	0.001962
4 000 037	Oui	0.003868
8 000 009	Oui	0.007665
16 000 057	Oui	0.015282
32 000 011	Oui	0.030374
64 000 031	Oui	0.060643
256 000 001	Oui	0.242260
512 000 009	Oui	0.485135
128 000 03	Non (467 est un diviseur)	0.000005
1024 000 0009	Non (37 est un diviseur)	0.000003
2018 000 011	Non (17 est un diviseur)	0.000003

Tableau 2: Temps d'exécution de la première amélioration de l'algorithme de vérification de primalité pour différentes valeurs de N.

4.4. Représentation graphique des résultats

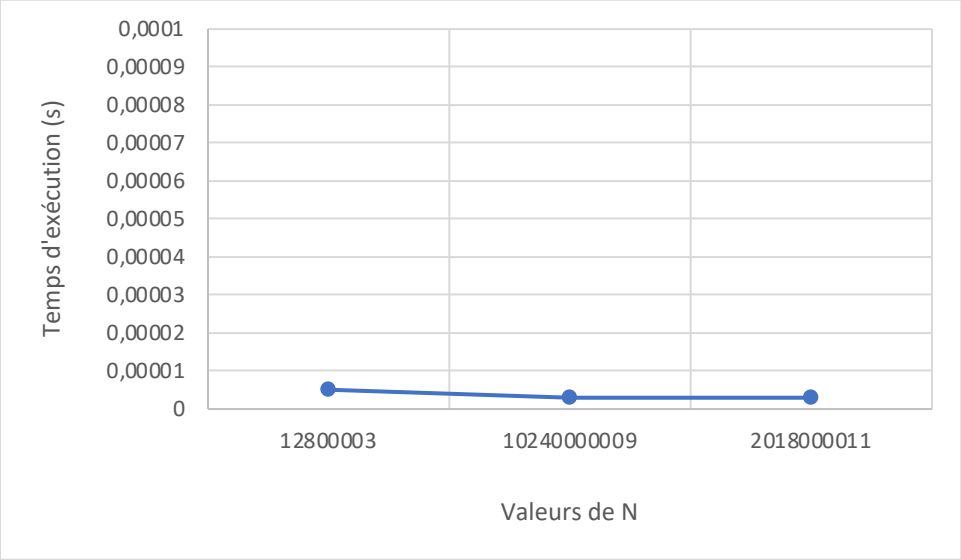


Temps d'exécution observés dans le pire cas (première amélioration de l'algorithme de vérification de primalité).

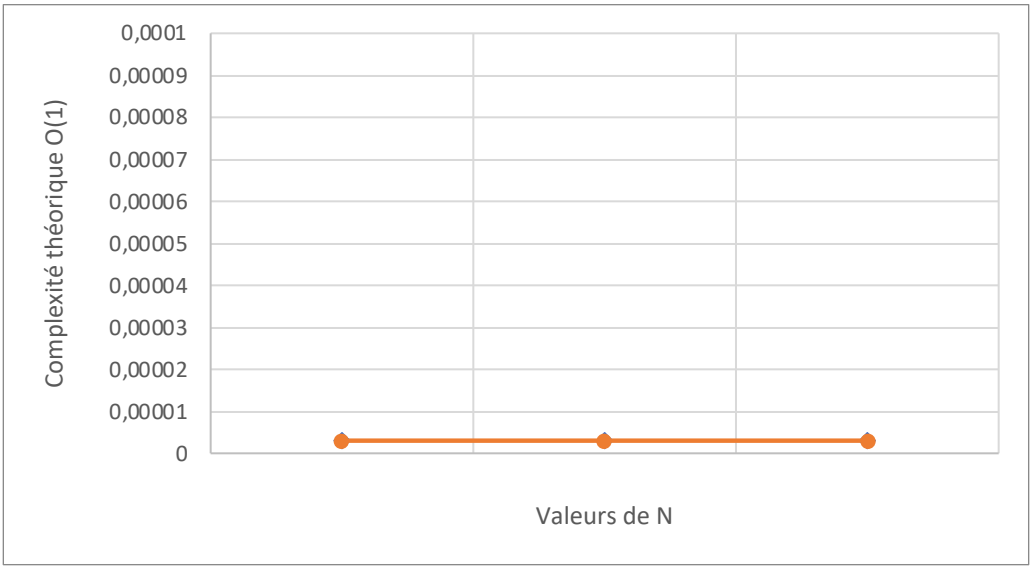


Complexité théorique O(N) dans le pire cas (première amélioration de l'algorithme de vérification de primalité).

Figure 4: Représentation des temps d'exécution et de la complexité théorique dans le pire cas (première amélioration de l'algorithme de vérification de primalité).



Temps d'exécution observés dans le meilleur cas (première amélioration de l'algorithme de vérification de primalité).



Complexité théorique O(1) dans le meilleur cas (première amélioration de l'algorithme de vérification de primalité).

Figure 5: Représentation des temps d'exécution et de la complexité théorique dans le meilleur cas (première amélioration de l'algorithme de vérification de primalité).

Étant donné que la complexité théorique de ce deuxième algorithme est identique à celle du premier (O(1) dans le meilleur cas et O(N) dans le pire cas), nous allons présenter une analyse concise des résultats expérimentaux, les explications détaillées ayant déjà été fournies dans la première partie.

4.5. Analyse des temps d'exécution

Les mesures de temps dans le tableau correspondent au meilleur et au pire cas, selon que N soit premier ou non.

- **Pire cas**

Pour les nombres premiers, on observe des temps d'exécution qui augmentent linéairement avec N (ex: de 0.000999s pour N=1,000,003 à 0.485135s pour N=512,000,009), correspondant à la complexité O(N) dans le pire cas.

- **Meilleur cas**

Pour les nombres non premiers avec petits diviseurs, les temps restent très faibles et constants (environ 0.000003s à 0.000005s), correspondant à la complexité O(1) dans le meilleur cas.

4.6. Observations sur les données et les mesures

- **Pour les nombres premiers**

Pour les nombres premiers : le temps d'exécution augmente linéairement avec N, doublant approximativement quand N double (ex: N=32,000,011 → 0.030374s, N=64,000,031 → 0.060643s).

- **Pour les nombres non premiers**

Temps d'exécution négligeables (≈0.000003-0.000005s) et constants, indépendamment de la taille de N.

4.7. Comparaison entre la complexité théorique et la complexité expérimentale

- **Pire cas:** La relation linéaire entre le temps d'exécution et N valide la complexité théorique O(N) dans le pire cas.

- **Meilleur cas:** Les temps constants pour les nombres non premiers confirment la complexité O(1) dans le meilleur cas.

- **Conclusion:** En résumé, les résultats expérimentaux confirment parfaitement l'analyse théorique de la complexité. La complexité est de O(1) dans le meilleur cas et elle vaut O(N) dans le pire cas.

4.8. Comparaison entre le premier et le deuxième algorithme

- **Complexité théorique :** Les deux algorithmes partagent les mêmes complexités théoriques, avec O(1) dans le meilleur cas (par exemple, pour les nombres pairs ou les entiers ≤ 1) et O(N) dans le pire cas (pour les grands nombres premiers)
  .
- **Comportement dans le meilleur cas:** Dans le meilleur cas, les deux algorithmes se comportent pratiquement de la même façon avec un temps d'exécution similaire d'environ 0.000003 secondes.

- **Efficacité globale** : En moyenne, le deuxième algorithme est environ 3 fois plus rapide que le premier pour les grands nombres premiers. Ce gain de performance est illustré par les résultats suivants :

N	Temps d'exécution Algorithme 1 (s)	Temps d'exécution Algorithme 2 (s)	Accélération
1 000 003	0.003121	0.000999	3,124124124
2 000 003	0.006290	0.001962	3,20591233
4 000 037	0.012432	0.003868	3,214064116
8 000 009	0.024764	0.007665	3,230789302
16 000 057	0.049475	0.015282	3,237468918
32 000 011	0.098741	0.030374	3,250839534
64 000 031	0.197166	0.060643	3,251257359
256 000 001	0.788759	0.242260	3,255836704
512 000 009	1.576491	0.485135	3,249592382
Accélération moyenne : 3,224431641			

Tableau 3: Comparaison des temps d'exécution entre l'algorithme naïf et la première amélioration pour différentes valeurs de N premiers.

Ce tableau montre une accélération moyenne d'environ 3,2. Ceci se justifie par le fait que le deuxième algorithme n'examine que les entiers jusqu'à N/2, réduisant ainsi le nombre total d'itérations. Cela le rend particulièrement avantageux pour les grands nombres premiers.

- **Fiabilité et précision** : Les deux algorithmes offrent la même fiabilité dans la détection des nombres premiers. L'amélioration des performances est obtenue sans compromettre la précision ou la fiabilité de l'algorithme.

En conclusion, bien que les deux algorithmes partagent la même complexité théorique, le deuxième algorithme offre une amélioration pratique significative avec des temps d'exécution réduits, le rendant plus efficace pour des applications réelles, particulièrement avec de grands nombres.

5. Partie 3 : Deuxième amélioration de l'algorithme de vérification de primalité

5.1. Conception de l'algorithme

On garde le même principe que dans le premier algorithme, la seule différence étant qu'on teste les entiers de 2 à $\sqrt{N}$.

Algorithme 3 : *Deuxième amélioration de l'algorithme de vérification de primalité*

```
Variables N, i: Entier
Variable trouv: Booléen
Début
    Lire(N)
    Si (N ≤ 1) Alors
        Afficher ("N n'est pas premier")
    Fin Si
    Sinon
        i ← 2
        trouv ← Faux
        Tant que (trouv = Faux  ET  i ≤ √N) Faire
            Si (i mod N = 0) Alors
                trouv ← Vrai
            Fin Si
            Sinon
                i ← i + 1
            Fin Sinon
        Fin Tant que
        Si (trouv = Vrai) Alors
            Afficher ("N est premier")
        Sinon
            Afficher ("N n'est pas premier")
        Fin Sinon
    Fin Sinon
Fin
```

5.2. Analyse théorique de la complexité

i. Meilleur cas

Le meilleur cas sera identique à celui du premier algorithme: lorsque N est inférieur ou égal à 1, ou lorsque N est divisible par 2. Ainsi, dans le meilleur cas, la complexité temporelle est O(1).

ii. Pire cas

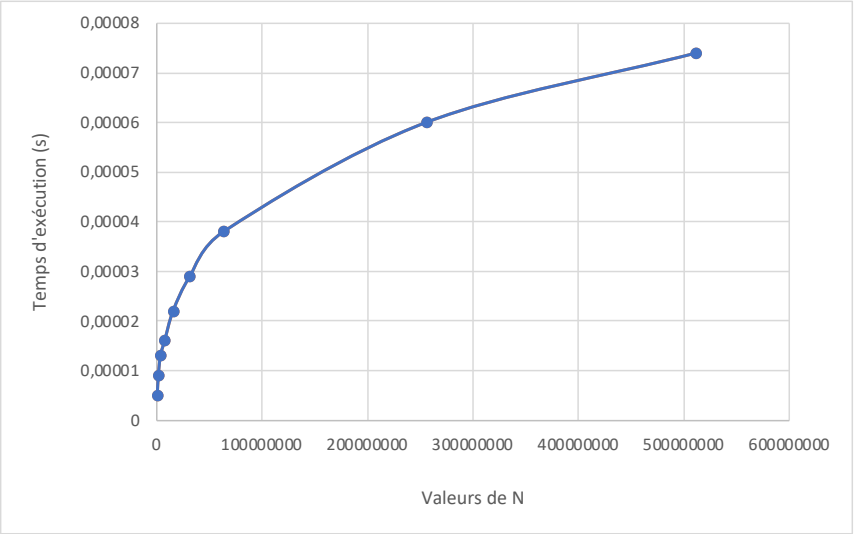
Le pire cas survient lorsque N est un nombre premier, car la boucle devra vérifier tous les entiers i de 2 à $\sqrt{N}$ pour confirmer que N est premier. La boucle s'exécute alors jusqu'à ce que i atteigne $\sqrt{N}$. Cela signifie que, dans le pire des cas, la boucle s'exécute $\sqrt{N}$ - 2 + 1 = ($\sqrt{N}$ - 1) fois. Ainsi, la complexité dans le pire cas est O($\sqrt{N}$).

5.3. Mesures expérimentales

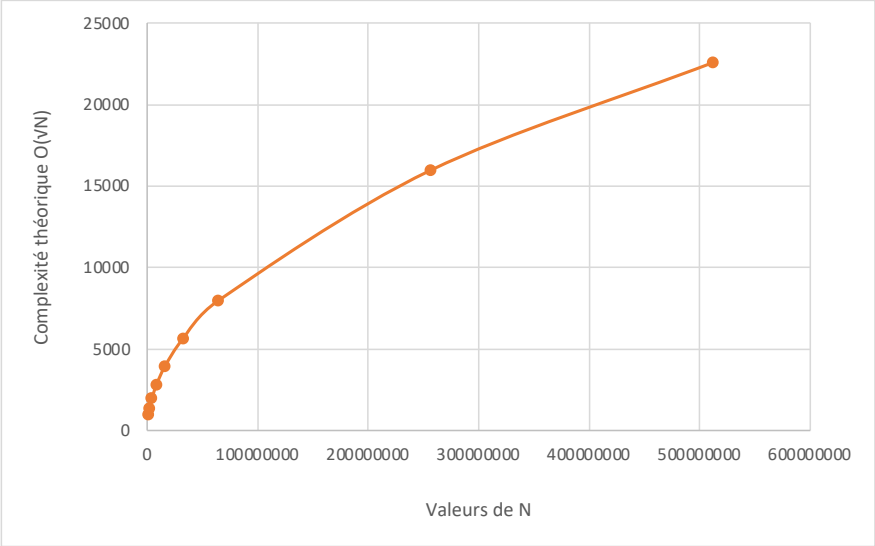
N	Nombre premier ?	Temps d'exécution (s)
1 000 003	Oui	0.000005
2 000 003	Oui	0.000009
4 000 037	Oui	0.000013
8 000 009	Oui	0.000016
16 000 057	Oui	0.000022
32 000 011	Oui	0.000029
64 000 031	Oui	0.000038
256 000 001	Oui	0.000060
512 000 009	Oui	0.000074
128 000 03	Non (467 est un diviseur)	0.000005
1024 000 0009	Non (37 est un diviseur)	0.000003
2018 000 011	Non (17 est un diviseur)	0.000003

Tableau 4: Temps d'exécution de la deuxième amélioration de l'algorithme de vérification de primalité pour différentes valeurs de N.

5.4. Représentation graphique des résultats

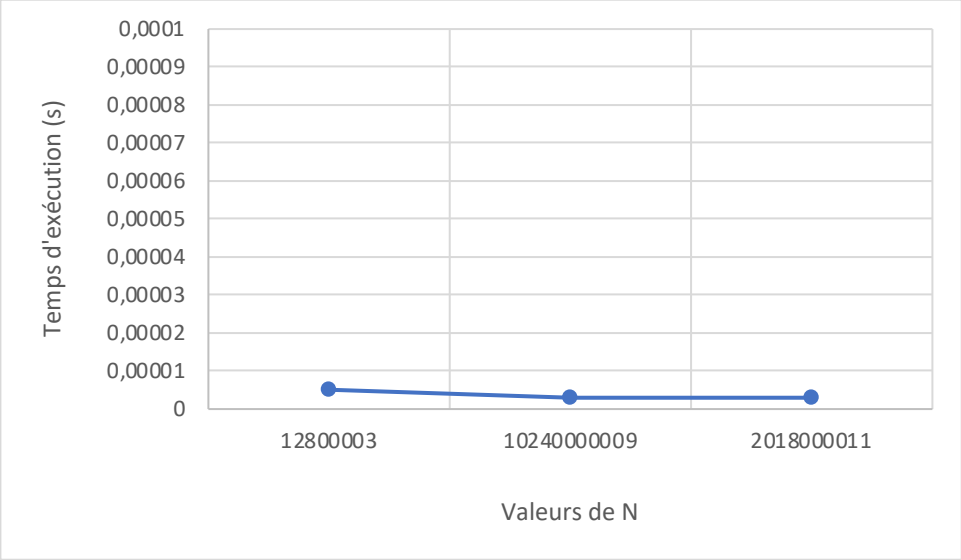


Temps d'exécution observés dans le pire cas (deuxième amélioration de l'algorithme de vérification de primalité).

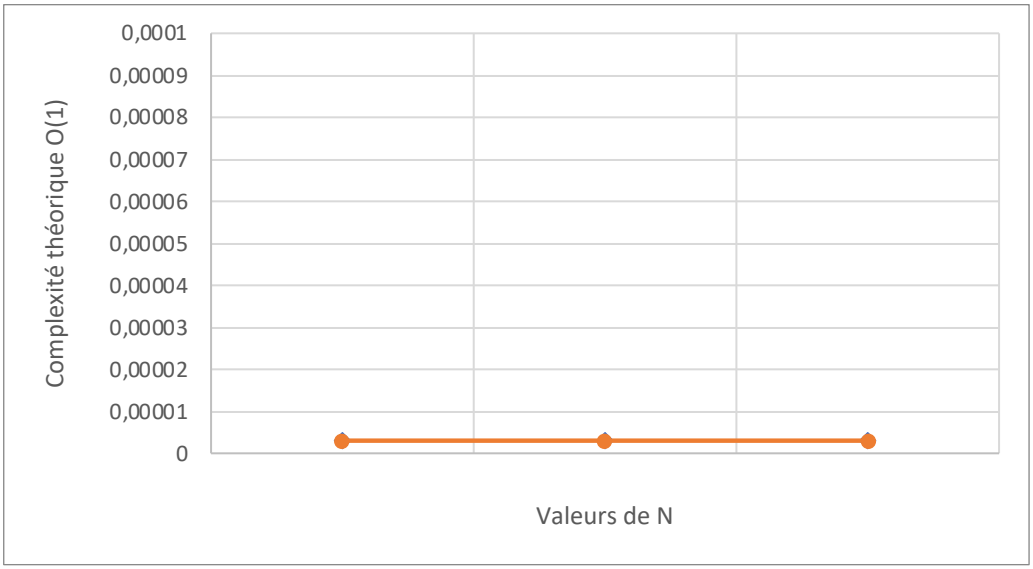


Complexité théorique O($\sqrt{N}$) dans le pire cas (deuxième amélioration de l'algorithme de vérification de primalité).

Figure 6: Représentation des temps d'exécution et de la complexité théorique dans le pire cas (deuxième amélioration de l'algorithme de vérification de primalité).



Temps d'exécution observés dans le meilleur cas (deuxième amélioration de l'algorithme de vérification de primalité).



Complexité théorique O(1) dans le meilleur cas (deuxième amélioration de l'algorithme de vérification de primalité).

Figure 7: Représentation des temps d'exécution et de la complexité théorique dans le meilleur cas (deuxième amélioration de l'algorithme de vérification de primalité).

5.5. Analyse des temps d'exécution

Les mesures du temps obtenues correspondent au meilleur ou au pire cas selon que N soit premier ou non.

- **Pour les nombres premiers**

Pour les nombres premiers, le temps d'exécution augmente avec l'augmentation de la taille de l'entier N, par exemple, de 0.000005 secondes pour N=1,000,003 à 0.000074 secondes pour N=512,000,0099. De plus, l'augmentation du temps d'exécution est proportionnelle à $\sqrt{N}$, comme le montre la courbe bleue dans la figure 6. Cette augmentation proportionnelle à $\sqrt{N}$ correspond bien à la complexité O($\sqrt{N}$) attendue dans le pire cas.

- **Meilleur cas (Nombres non premiers)**

Lorsqu'un diviseur est trouvé rapidement, l'algorithme s'arrête très tôt, produisant des temps d'exécution constants et proches de zéro (ex. entre 0.000003 et 0.000005 secondes). Cette performance constante reflète bien la complexité O(1) dans le meilleur cas.

5.6. Observations sur les données et les mesures

- **Pour les nombres premiers**

Les données montrent que le temps d'exécution croît proportionnellement à $\sqrt{N}$. Par exemple:
N1=4,000,037 → $\sqrt{N1} \approx 2000$→ 0.000013s.
N2=16,000,057 → $\sqrt{N2} \approx 4000$ ($\approx 2\sqrt{N1}$) → 0.000016s (≈2x).

- **Pour les nombres non premiers**

Les temps d'exécution sont très faibles et restent constants (entre 0.000003 et 0.000005 secondes), même pour des valeurs de N élevées.

5.7. Comparaison entre la complexité théorique et la complexité expérimentale

- **Pire cas**

Les résultats expérimentaux suivent bien la courbe théorique en O($\sqrt{N}$), comme le montre les courbes de la figure 6. L'évolution des temps d'exécution suit une courbe caractéristique de racine carrée, confirmant que la complexité expérimentale correspond à la complexité théorique qui est de O($\sqrt{N}$).

- **Meilleur cas**

En théorie, le meilleur cas est O(1), correspondant aux situations où un diviseur est trouvé rapidement. Cette analyse est confirmée par les résultats expérimentaux, où les temps d'exécution demeurent constants et très bas pour les nombres non premiers divisibles par un petit entier. Cette analyse est confirmée visuellement par les deux courbes de la figure 7: où la courbe expérimentale (en bleu) et théorique (en orange) sont toutes deux horizontales et proches de zéro, démontrant le comportement O(1) attendu.

- **En conclusion**

Les mesures expérimentales valident parfaitement l'analyse théorique de la complexité : O(1) dans le meilleur cas et O($\sqrt{N}$) dans le pire cas.

5.8. Comparaison entre les trois algorithmes

- **Complexité théorique**

i. **Premier algorithme :** Teste tous les entiers de 2 à N−1.
   - Complexité dans le meilleur cas : O(1), lorsque N est pair ou inférieur ou égal à 1.
   - Complexité dans le pire cas : O(N), particulièrement long pour les grands nombres premiers, car chaque entier doit être vérifié.

ii. **Deuxième algorithme :** Limite le test aux entiers de 2 à N/2.
   - Complexité dans le meilleur cas : O(1), similaire au premier algorithme.
   - Complexité dans le pire cas : O(N), avec un nombre d'itérations réduit par rapport au premier algorithme, mais restant proportionnel à N.

iii. **Troisième algorithme :** Teste uniquement jusqu'à $\sqrt{N}$, ce qui réduit significativement le nombre d'itérations nécessaires.
   - Complexité dans le meilleur cas : O(1), comme pour les autres algorithmes.
   - Complexité dans le pire cas : O($\sqrt{N}$), ce qui est bien plus efficace, en particulier pour les grands nombres premiers.

- **Comportement dans le meilleur cas**

Dans le meilleur cas, les trois algorithmes affichent des temps d'exécution similaires d'environ 0,000003 secondes. Ce comportement identique montre qu'aucun algorithme n'est pénalisé dans les scénarios optimaux.

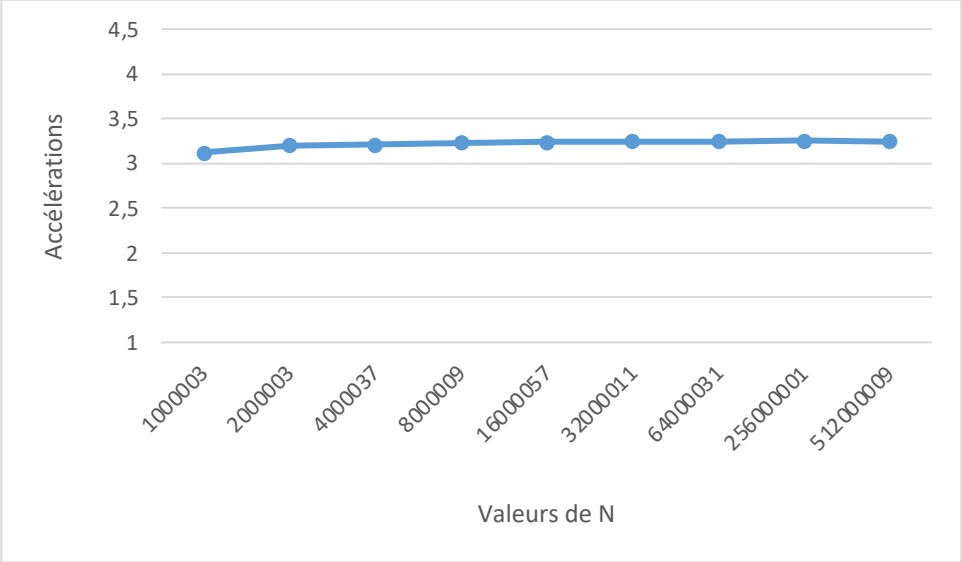
- **Comparaison expérimentale**

Résumons les mesures obtenues pour les pires cas des parties précédentes dans le tableau suivant:

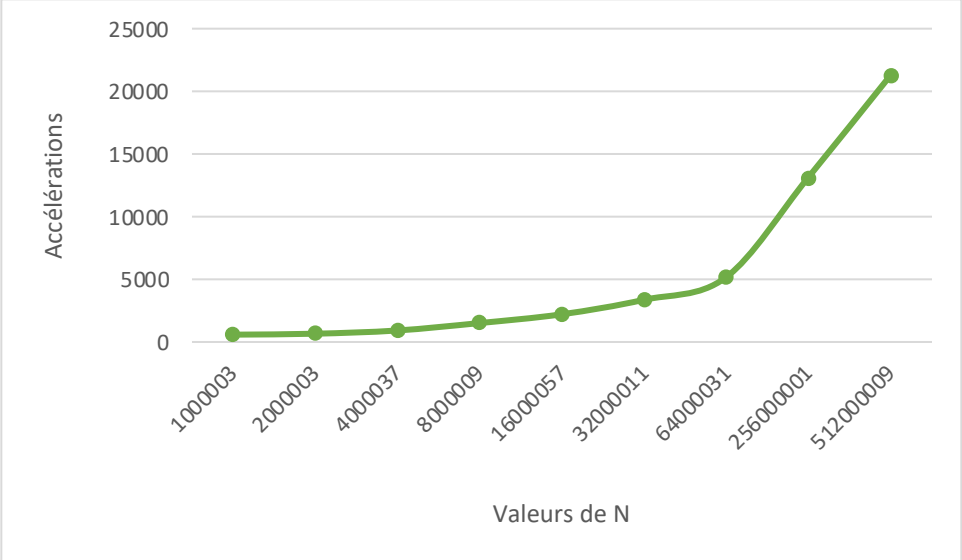
N	Temps d'exécution Algorithme 1 (s)	Temps d'exécution Algorithme 2 (s)	Temps d'exécution Algorithme 3 (s)	Accélération obtenue avec l'algorithme 2	Accélération obtenue avec l'algorithme 3
1 000 003	0.003121	0.000999	0.000005	3,12	624,2
2 000 003	0.006290	0.001962	0.000009	3,21	698,89
4 000 037	0.012432	0.003868	0.000013	3,21	956,31
8 000 009	0.024764	0.007665	0.000016	3,23	1547,75
16 000 057	0.049475	0.015282	0.000022	3,24	2248,86
32 000 011	0.098741	0.030374	0.000029	3,25	3404,86
64 000 031	0.197166	0.060643	0.000038	3,25	5188,58
256 000 001	0.788759	0.242260	0.000060	3,26	13145,98
512 000 009	1.576491	0.485135	0.000074	3,25	21303,93

Tableau 5: Comparaison des performances des trois algorithmes de test de primalité.

Pour visualiser efficacement les gains de performance obtenus avec les différents algorithmes, les graphes ci-dessous ont été établis.



Accélération obtenue avec la première amélioration (Algorithme 2) pour différentes valeurs de N premiers.



Accélération obtenue avec la deuxième amélioration (Algorithme 3) pour différentes valeurs de N premiers.

Figure 8: Visualisation des gains de performance obtenus avec les algorithmes 2 et 3 pour différentes valeurs de N premiers.

- **Premier algorithme** : Son temps d'exécution est proportionnel à N, ce qui le rend particulièrement lent pour les grands nombres premiers.

- **Deuxième algorithme** : Il offre une accélération d'environ 3x par rapport au premier, grâce à la réduction de l'intervalle de recherche de [2, N-1] à [2, N/2]. Cette accélération est pratiquement constante, comme le montre la courbe bleue de la figure 8.

- **Troisième algorithme** : Ce dernier présente un gain exponentiel en performance, atteignant parfois jusqu'à plus de 20 000 fois la vitesse originale pour les valeurs les plus élevées de N, grâce à la réduction de l'intervalle de recherche de [2, N-1] à [2, $\sqrt{N}$] qui permet de réduire drastiquement le nombre d'itérations. Cette accélération exponentielle est illustrée par la courbe verte de la figure 8.

- **Fiabilité et précision**

Les trois algorithmes partagent une fiabilité identique dans la détection des nombres premiers. Bien que le troisième algorithme soit nettement plus rapide, l'amélioration des performances est obtenue sans aucune perte de précision ou de fiabilité, ce qui les rend tous adéquats pour des applications où la justesse de la détection est cruciale.

- **Conclusion**

Bien que tous trois partagent une complexité O(1) dans le meilleur cas, le troisième algorithme est de loin le plus efficace dans le pire cas, offrant un gain exponentiel en performance pour les nombres premiers de grande taille, ce qui le rend idéal pour les applications réelles où N est élevé.

6. Conclusion générale

Ce travail pratique nous a permis d'explorer en profondeur la relation entre théorie et pratique dans l'analyse de la complexité algorithmique. À travers l'étude de trois versions d'un algorithme de vérification de primalité, nous avons pu confirmer que les prédictions théoriques correspondent bien aux mesures expérimentales. L'optimisation progressive de l'algorithme, passant d'une complexité O(N) à O($\sqrt{N}$), a démontré des gains de performance significatifs, avec une accélération allant jusqu'à 21303 fois pour les grands nombres dans la version finale. Cette étude a ainsi atteint ses objectifs en illustrant concrètement l'importance de l'analyse de complexité et de l'optimisation algorithmique dans le développement de solutions performantes.