-TP N°1 Test de primalité-

OULDGOUGAM Madjid Riad 2222231472314
YEDDOU Abdelkader Raouf 222231501902
FERGUENE Abdelraouf 222231361813

Algorithme 1 (A1) : Approche naïve

1. Cette solution comporte une boucle dans laquelle on va tester si le nombre N est divisible par 2, 3, …, N-1. Ecrire l'algorithme correspondant.

```
Si (N=0) ou (N=1) alors retourner faux
Sinon
    Pour i←2 à N-1 faire
            Si (N mod i=0) alors retourner faux ;
            Fsi ;
     Fait ;
Retourner vrai ;
```

2. Calculer la complexité théorique au pire cas de cet algorithme notée O(N).

   Le nombre d'itérations de la boucle pour = (N-1)-2+1=N-2 itérations$\sim N$

   donc la complexité est de l'ordre de O(N)

3. Ecrire le programme correspondant.

```
Bool isPrime_A1(long long N) {
  if (N <= 1) return false;
  for (long long i = 2; i < N-1; i++) {
     if (N % i == 0) return false;
  }
  return true;
}
```

   a. Vérifier que les nombres N proposés dans le tableau ci-dessous (1000003, 2000003, …) sont premiers.

Ils sont tous premiers

b. Mesurer les temps d'exécution T pour l'échantillon des nombres N ci-dessous et compléter le tableau :

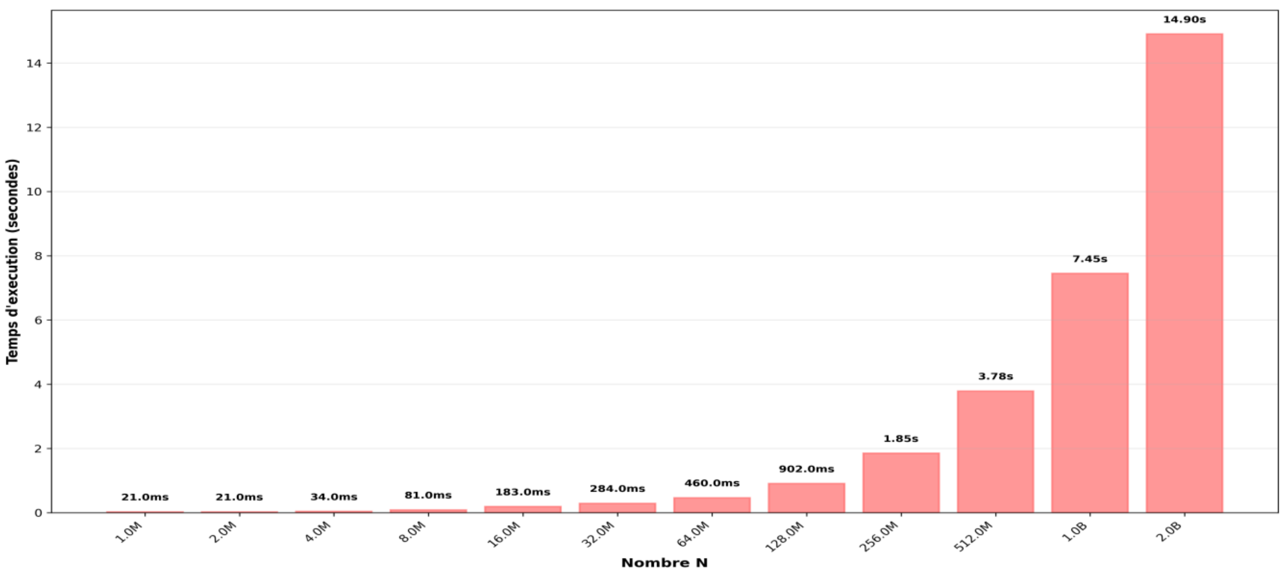
N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0.02100000	0.02100000	0.03400000	0.08100000	0.18300000	0.28400000	0.46000000

N	128000003	256000001	512000009	1024000009	2048000011
T	0.90200000	1.84700000	3.78400000	7.44700000	14.90000000

c. Que remarque-t-on sur les données de l'échantillon et sur les mesures obtenues ? (Indication : comparer chaque nombre N avec le suivant et chaque mesure du temps avec la suivante.)
On remarque que le temps est à peu près multiplié par deux en passant d'une valeur à une autre (sachant que la valeur courante est à peu près égal au double de la précédente) donc le temps augmente linéairement.

d. Comparer la complexité théorique et les mesures expérimentales.
Les prédictions théoriques sont-elles compatibles avec les mesures expérimentales ?
Oui les deux résultats sont compatibles.

e. Représenter avec un histogramme



Algorithme 2 (A2) : Amélioration de l'approche naïve

On sait que tout diviseur i du nombre N vérifie la relation : $i \leq N/2$, avec $i \neq N$.

1. Développer un 2ème algorithme en tenant compte de cette propriété et reprendre les mêmes questions précédentes[1].

Si (N=0) ou (N=1) alors retourner faux

Sinon

  Pour i←2 à N/2 faire

    Si (N mod i=0) alors retourner faux ;

    Fsi ;

  Fait ;

Retourner vrai ;

Le nombre d'itérations de la boucle pour = (N/2)-2+1=N/2-1 itérations~N

donc la complexité est de l'ordre de O(N)

```
bool isPrime_A2(long long N) {
  if (N <= 1) return false;
  for (long long i = 2; i <= N / 2;
    i++) {
    if (N % i == 0) return false;
  }
  return true;
}
```

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0.00600000	0.00900000	0.02100000	0.05100000	0.10500000	0.1200000	0.23700000

N	128000003	256000001	512000009	1024000009	2048000011
T	0.47300000	0.93000000	1.87900000	3.96400000	7.66100000

2. Comparer algorithmes A1 et A2 (représenter pour cela dans une même figure les graphes des 2 algorithmes). Lequel des 2 algorithmes est meilleur (ou plus performant) ?

Les deux algorithmes présentent une croissance linéaire du temps d'exécution en fonction de N, ce qui est cohérent avec leur complexité théorique O(N). Cependant, l'algorithme A2 est systématiquement plus rapide que A1 pour toutes les valeurs testées.

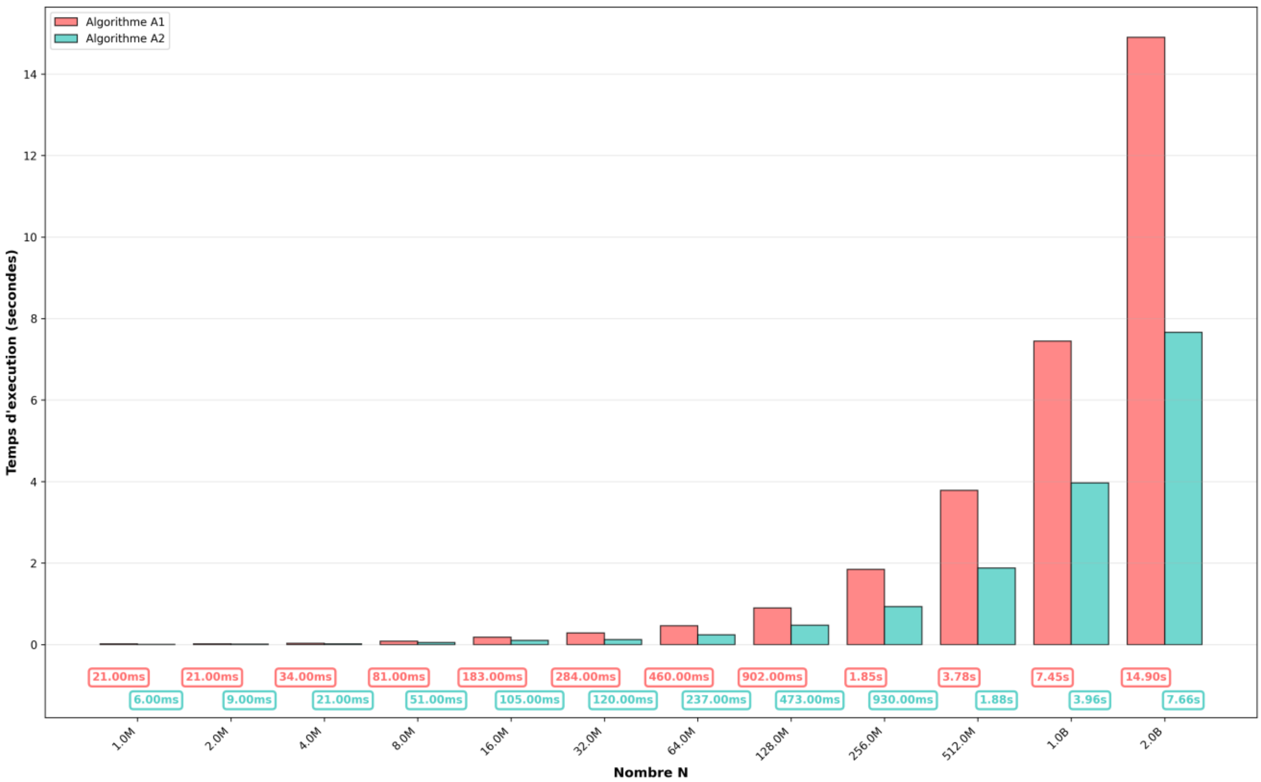
En effet :

Dans A1, la boucle teste tous les diviseurs de 2 à N−1.

Dans A2, la boucle ne teste que jusqu’à N/2, ce qui divise le nombre d’itérations par 2.

Les résultats expérimentaux confirment cette différence :
le temps d’exécution de A2 est environ deux fois plus faible que celui de A1 pour les mêmes valeurs de N.

Conclusion : A2 est plus performant que A1, tout en gardant la même complexité asymptotique O(N).



Algorithme 3 (A3) :

1. Développer un 2ème algorithme en tenant compte de cette propriété et reprendre les mêmes questions précédentes[1].

```
bool isPrime_A3(long long N) {
   if (N <= 1) return false;
   long long limit = sqrt(N);
   for (long long i = 2; i <= limit; i++) {
      if (N % i == 0) return false;
   }
   return true;
}
```

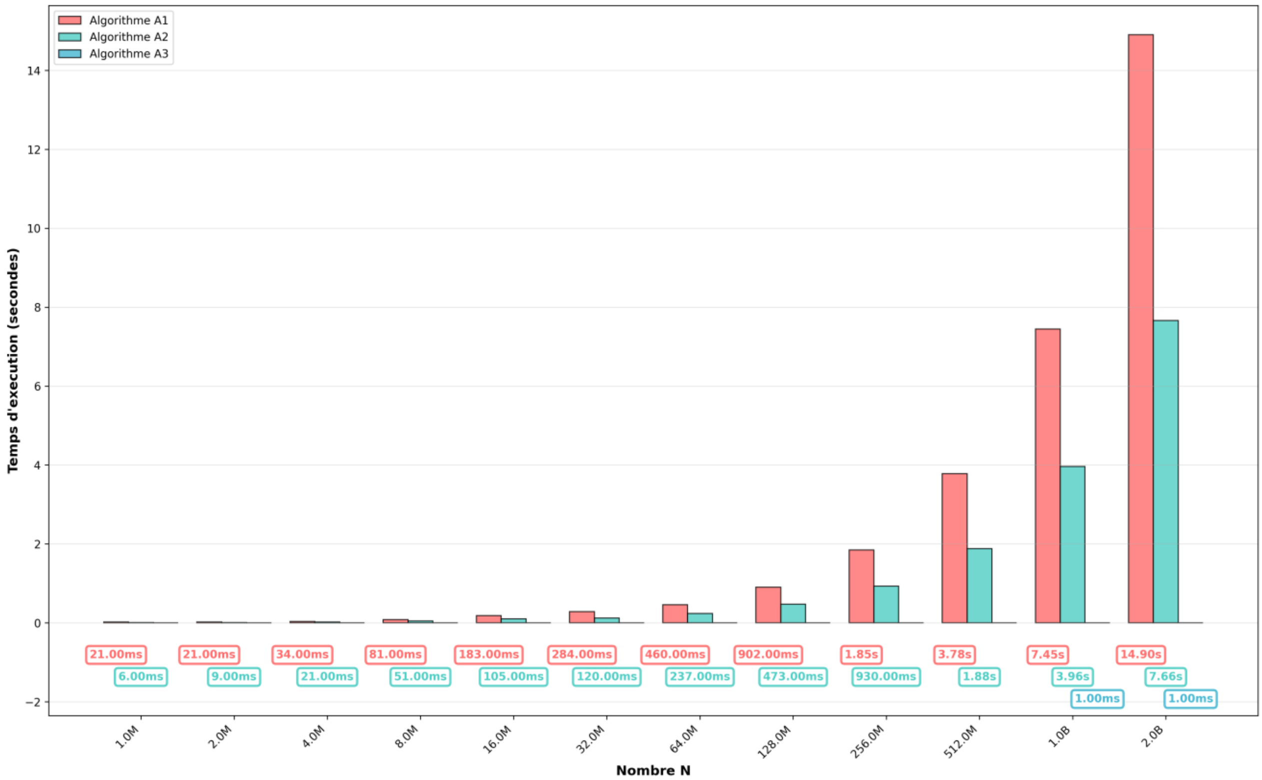
Le nombre d'itérations de la boucle pour =

donc la complexite est de l'ordre O(N)

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000001300	0.00000000

N	128000003	256000001	512000009	1024000009	2048000011
T	0.00000000	0.00000000	0.00000000	0.00100000	0.001000000

2. Comparer les 3 algorithmes. Lequel des 3 algorithmes est meilleur (ou plus performant) ?



L'algorithme 3 est plus performant.

Algorithme 4 (A4) :

1. Développer un 4ème algorithme A4 en tenant compte de cette proposition et reprendre les mêmes questions précédentes[1].

```
bool isPrime_A4(long long N) {
   if (N <= 1) return false;
   if (N == 2) return true;
   if (N % 2 == 0) return false;
   long long limit = sqrt(N);
   for (long long i = 3; i <= limit; i += 2) {
      if (N % i == 0) return false;
```

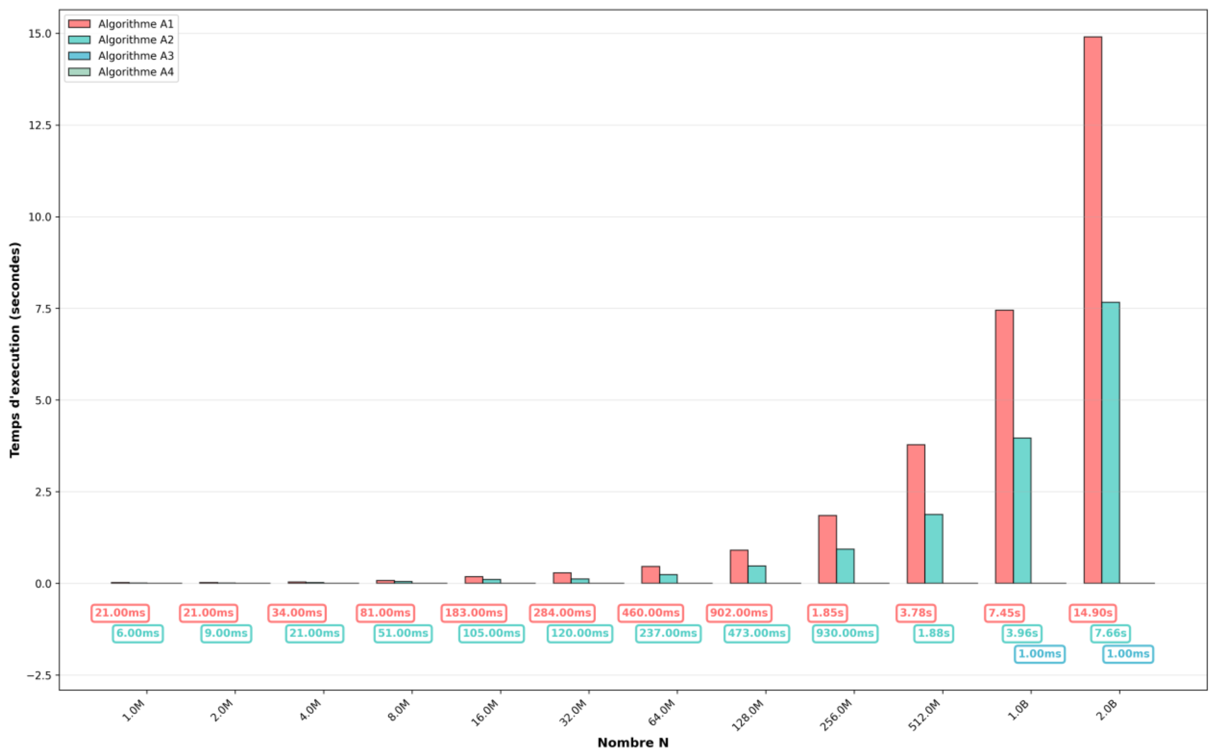
}
return true;

Le nombre d'itérations de la boucle pour = $\sqrt{N}$-2+1=$\sqrt{N}$-1 itérations~N donc la complexité est de l'ordre de O(N)

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000

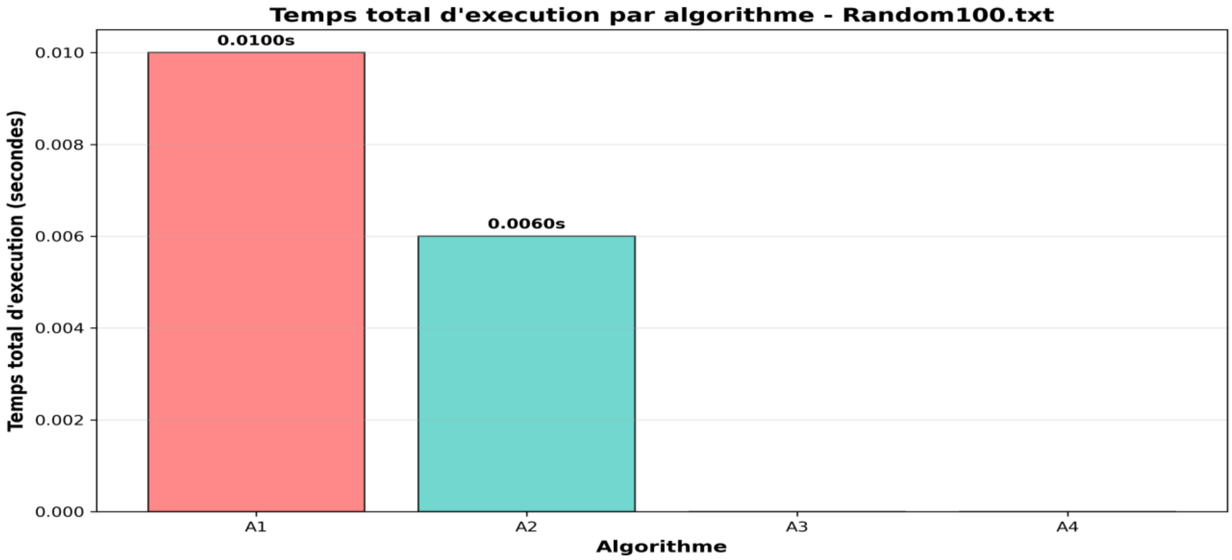
N	128000003	256000001	512000009	1024000009	2048000011
T	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000

1. Comparer les 4 algorithmes. Lequel des 4 algorithmes est meilleur (ou plus performant) ?



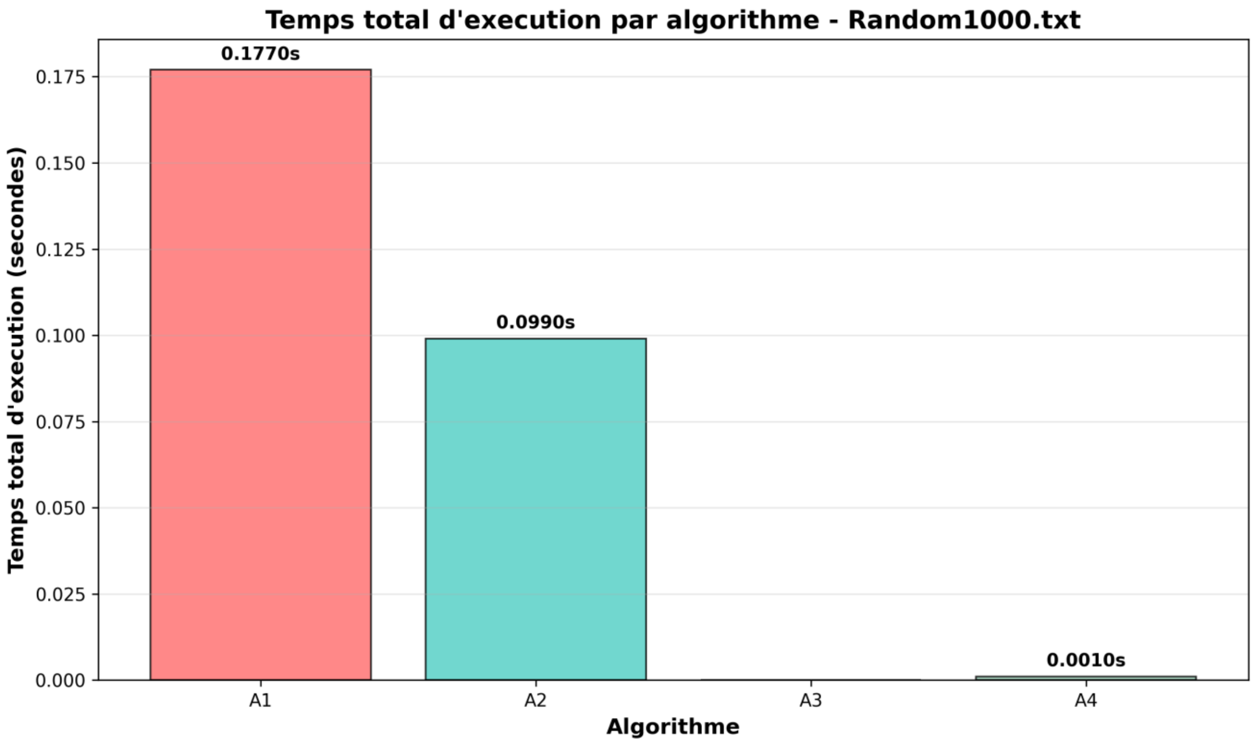
Le plus performant est l'algorithme 4 quand le nombre est pair (non premier) sinon c'est l'algorithme 3.
Au pire cas l'algorithme 3 est plus performant (avec un test en moins que l'algorithme 4).

Histogramme avec Dataset Random100 :



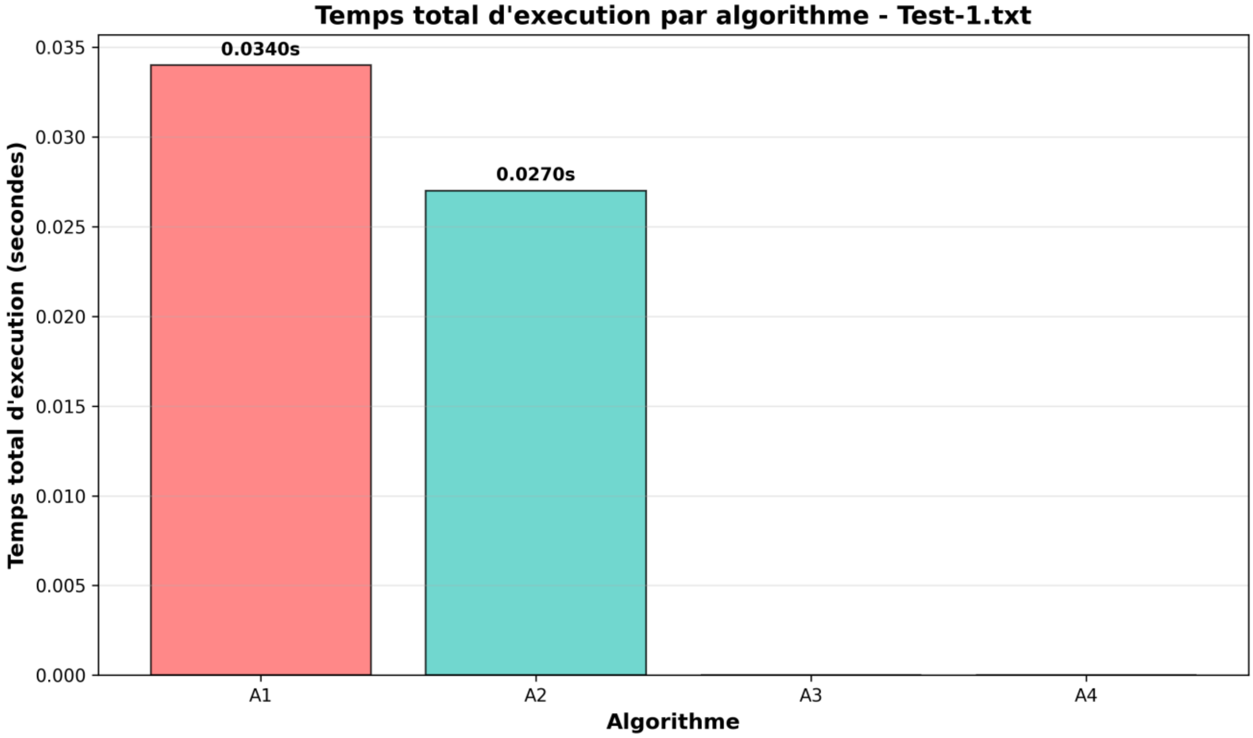
Cette illustration montre les temps d’exécution des algorithmes sur Random100.txt. A1 reste le plus lent, tandis qu’A2 est plus performant et A3–A4 sont trop rapides pour être visibles.

Histogramme avec Dataset Random1000 :



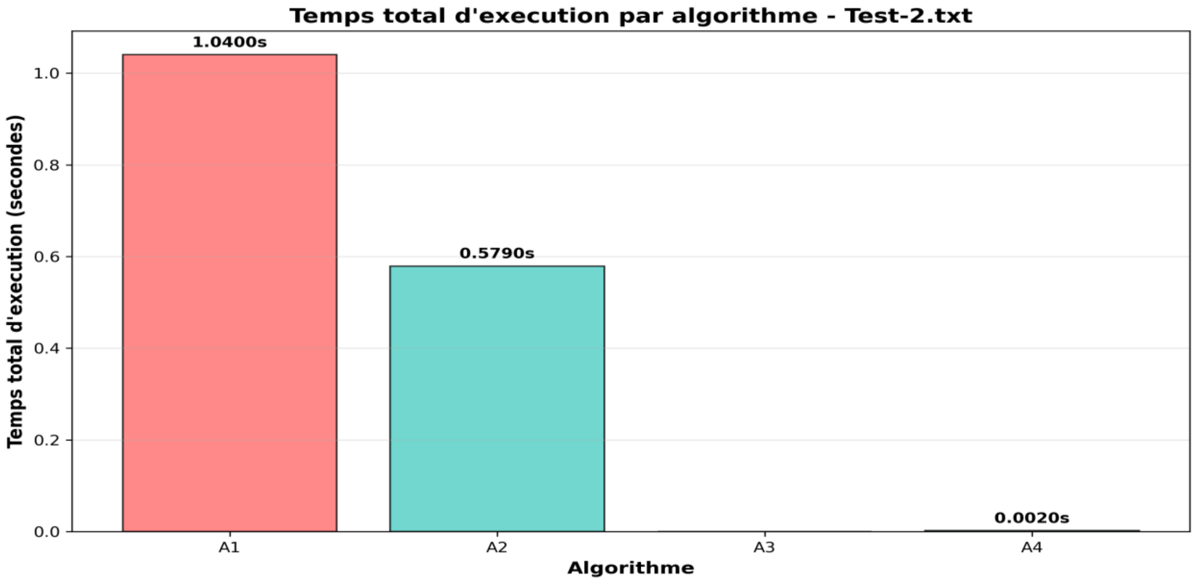
Ce histogramme compare le temps total d'exécution des quatre algorithmes sur le fichier Random1000.txt. On observe que A3 est largement le plus rapide, tandis que A1 est le plus lent.

Histogramme avec Dataset Test-1:



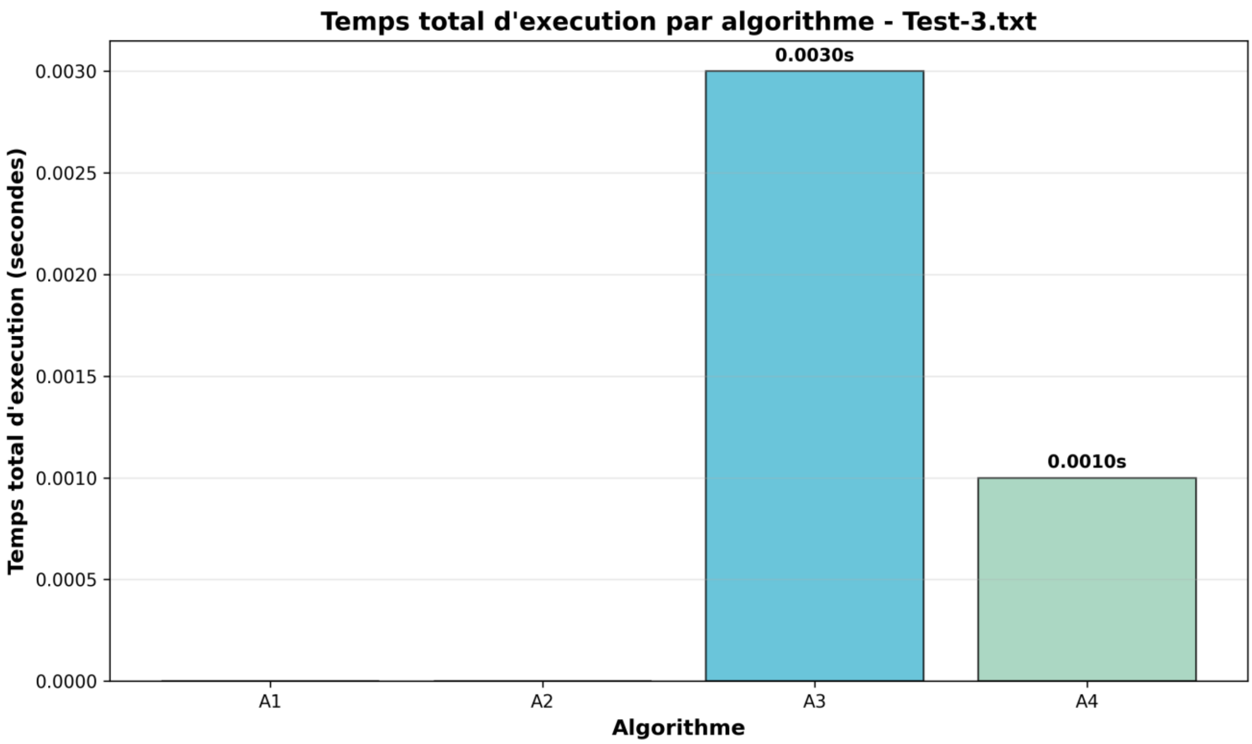
Cette image présente les temps d'exécution sur Test-1.txt, montrant qu'A1 est le Plus lent tandis qu'A2 est légèrement plus rapide ; A3 et A4 restent trop rapides Pour apparaître visiblement.

Histogramme avec Dataset Test-2:



Ce graphique présente les temps d'exécution sur Test-2.txt, montrant qu'A1 est le Plus lent tandis qu'A2 est légèrement plus rapide ; A3 et A4 restent trop rapides pour apparaître visiblement.

Histogramme avec Dataset Test-3:



Ce graphique présente les temps d'exécution sur Test-3.txt, montrant qu'A3 est le plus lent tandis qu’A4 est légèrement plus rapide .