Corrigé TP N°1
Test de primalité

L'objectif de ce TP est de tester différentes méthodes pour un même problème et aussi apprendre à mesurer le temps d'exécution d'un programme.
Il s'agit ici de tester si un nombre entier naturel est premier.
Quatre algorithmes sont proposés qu'il faut implémenter en langage C, puis les comparer en utilisant les fonctions de gestion du temps qui sont fournies dans la bibliothèque time.h.

Rappel : Un nombre entier naturel N est premier s'il n'a que 2 diviseurs : le nombre 1 et le nombre N lui-même.

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

3. Ecrire le programme correspondant.

```
int estPremier1(int N){
     if(N==0||N==1) return 0;
     else { for(int i=2;i<=N-1;i++)
              if(N%i==0) return 0;
              return 1;
            }
}
```

   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 :

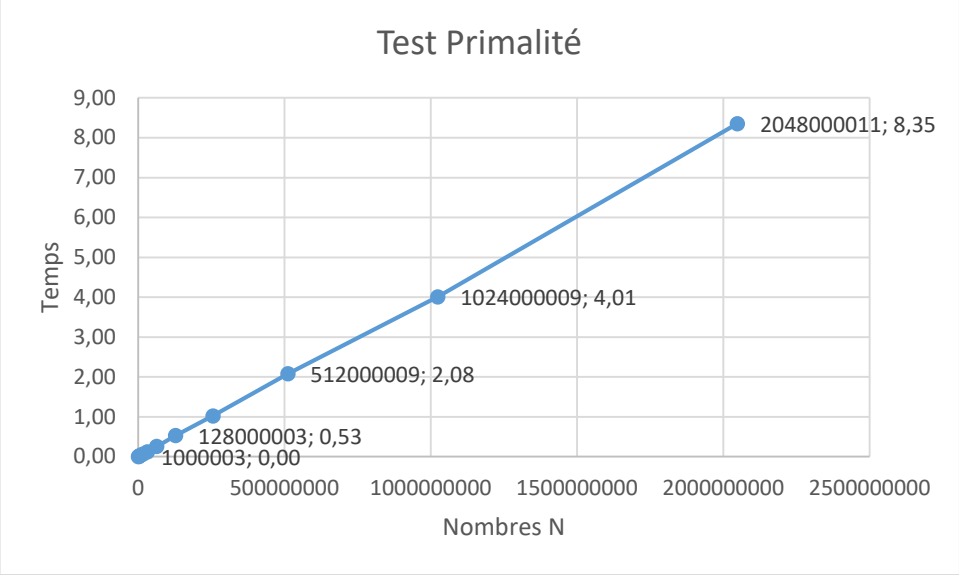
N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0,00	0,01	0,02	0,03	0,06	0,12	0,26

N	128000003	256000001	512000009	1024000009	2048000011
T	0,53	1,03	2,08	4,01	8,35

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 2 graphes les variations du temps d'exécution T(N) et les variations de la complexité au pire cas O(N). Utiliser pour cela un logiciel graphique comme Excel.



On constate que la courbe est une droite donc l'algorithme est linéaire

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)

```
int estPremier2(int N){
    if(N==0||N==1) return 0;
    else {
            for(int i=2;i<=N/2;i++)
          if(N%i==0) return 0;
       return 1;
    }
}
```

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0,00	0,00	0,01	0,02	0,04	0,07	0,13

N	128000003	256000001	512000009	1024000009	2048000011
T	0,28	0,67	1,07	2,11	4,08

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) ?

[1] Copier-coller le programme précédent puis modifier-le.

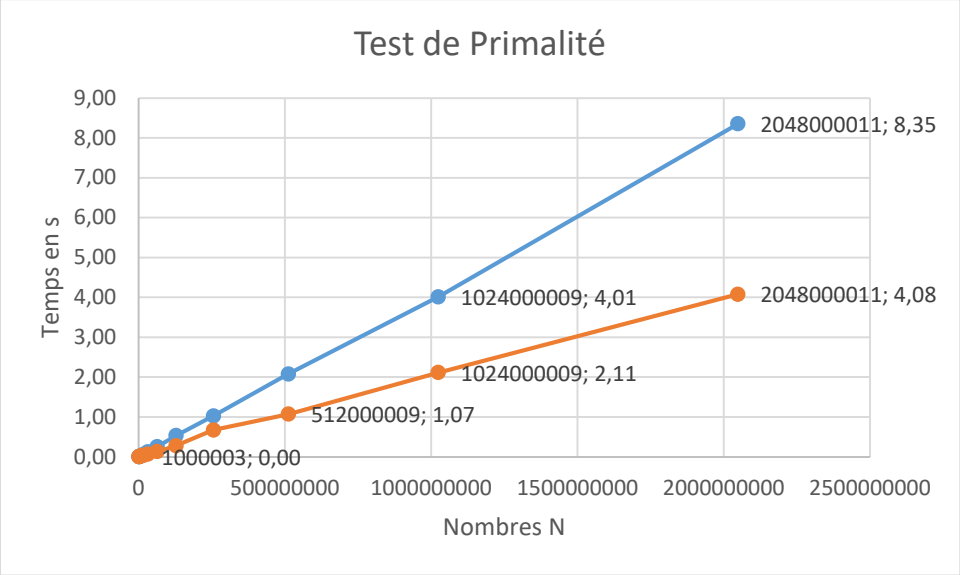


Figure 1 : Algorithme 1 et Algorithme 2

Algorithme 3 (A3) :

Il existe une propriété mathématique sur les nombres entiers :

Propriété : Les diviseur d'un nombre entier N sont pour la moitié $\leq N^{1/2}$ ($=\sqrt{N}$) et pour l'autre moitié $> N^{1/2}$

1. Développer un 3ème algorithme A3 en tenant compte de cette propriété et reprendre les mêmes questions précédentes[1].

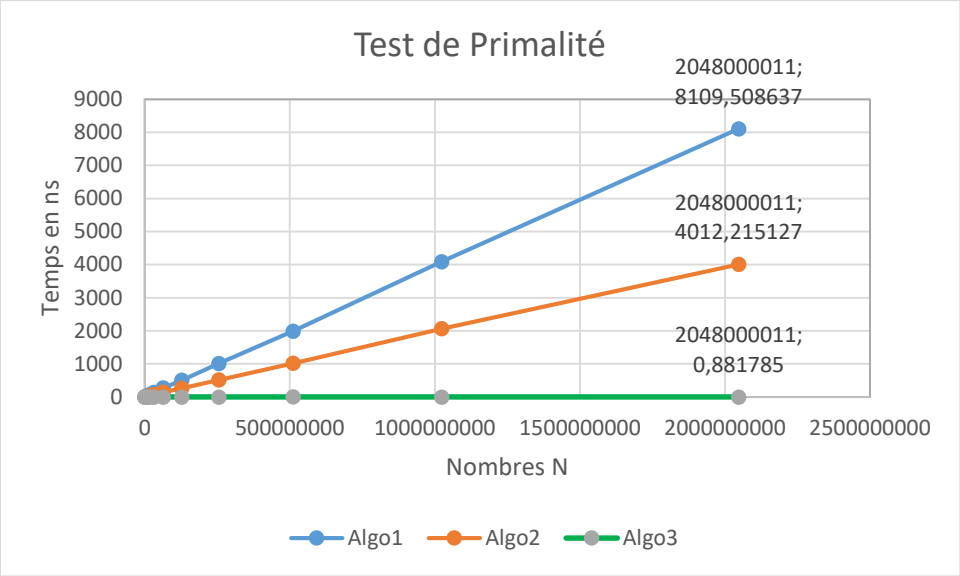
Si (N=0) ou (N=1) alors retourner faux
Sinon
  Pour i←2 à $\sqrt{N}$ faire
    Si (N mod i=0) alors retourner faux ;
    Fsi ;
  Fait ;
Retourner vrai ;
Le nombre d'itérations de la boucle pour = $\sqrt{N}$-2+1=$\sqrt{N}$-1 itérations~N
donc la complexité est de l'ordre de O(N)

```
int estPremier3(int N){
    if(N==0||N==1) return 0;
    else {
            for(int i=2;i<=sqrt(N);i++)
          if(N%i==0) return 0;
       return 1;
    }
}
```

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0,00	0,00	0,00	0,00	0,00	0,00	0,00

N	128000003	256000001	512000009	1024000009	2048000011
T	0,00	0,00	0,00	0,00	0,00

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



L'algorithme 3 est plus performant.

Algorithme 4 (A4) :

Une autre amélioration possible consiste à tester si N est impair et dans ce cas dans la boucle, il ne faut tester la divisibilité de N que par les nombres impairs.

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

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

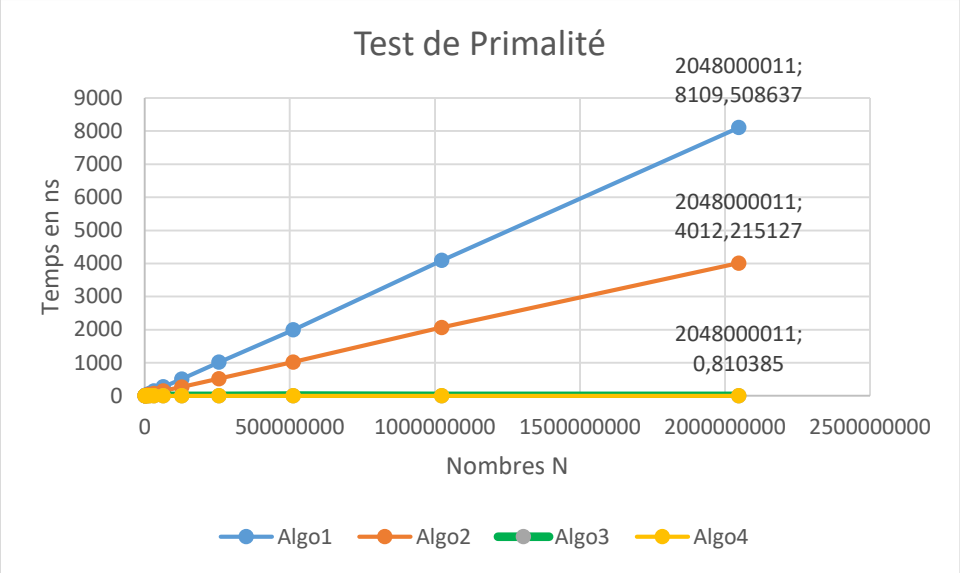
Le nombre d'itérations de la boucle pour = $\sqrt{N}$-2+1=$\sqrt{N}$-1 itérations~N
donc la complexité est de l'ordre de O(N)

```
int estPremier4(int N){
    if(N==0||N==1||N%2==0) return 0;
    else {  for(int i=2;i<=sqrt(N);i++)
                if(N%i==0) return 0;
            return 1;
          }
}
```

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T	0,00	0,00	0,00	0,00	0,00	0,00	0,00

N	128000003	256000001	512000009	1024000009	2048000011
T	0,00	0,00	0,00	0,00	0,00

4. 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).

Code 1 où le temps est en seconde :

```
#include<stdio.h>
#include<time.h>
#include<math.h>
int estPremier1(int N){
        if(N==0||N==1) return 0;
        else {  for(int i=2;i<=N-1;i++)
                    if(N%i==0) return 0;
                return 1;
        }
}
int estPremier2(int N){
        if(N==0||N==1) return 0;
        else {  for(int i=2;i<=N/2;i++)
                    if(N%i==0) return 0;
                return 1;
        }
}
int estPremier3(int N){
        if(N==0||N==1) return 0;
        else {  for(int i=2;i<=sqrt(N);i++)
                    if(N%i==0) return 0;
                return 1;
        }
}
int estPremier4(int N){
        if(N==0||N==1||N%2==0) return 0;
        else {  for(int i=2;i<=sqrt(N);i++)
                    if(N%i==0) return 0;
                return 1;
        }
}
main(){
  int N;
  int t[12]={1000003,2000003,4000037,8000009,16000057, 32000011,64000031,
             128000003,256000001,512000009,1024000009,2048000011};
   clock_t deb, fin;     double tps;
   for(int i=0;i<12;i++){
                deb = clock();
                if(estPremier4(t[i])) printf("\n%d est premier",t[i]);
                else printf("\n%d n'est pas premier",t[i]);
                fin = clock();
        tps = (double) (fin-deb)/ CLOCKS_PER_SEC;
        printf("\nLe temps d'execution = %f",tps);
  }
}
```