Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département IA et SD

Rapport travaux pratiques n° 1

Module : Conception et Complexité des Algorithmes

Mesure du temps d'exécution d'un programme

Travail présenté par :

- **BENKOUITEN Aymen———-191931046409**
- **MALKI Omar Chouaab———-191931081333**
- **KENAI Imad Eddine————-191932017671**
- **MEKKAOUI Mohamed———-191931081338**

2022/2023

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Lorsqu'on cherche une solution a un problème algorithmique il faut toujours trouver la solution la plus optimale et la plus efficace parmi toutes les solutions qu'on a programmées... Mais la question qui se pose est **comment déterminer laquelle de ces solutions est la plus efficace et la plus optimale ?**[1]

L'objectif dans ce TP est le calcul de complexité algorithmique temporelle pour pouvoir comparer l'efficacité d'algorithmes résolvant le même problème, cela permet donc d'établir lequel des algorithmes disponibles est le plus optimal.

Solution apportée

0.1 Développement de l'algorithme et du programme correspondant

0.1.1 Question 01

Écrire six algorithmes différents pour déterminer si un nombre entier est premier ou composé. Évaluer la complexité pour chacun des algorithmes proposés (en langage c).

Réponse 01 :

Algorithme 01 : Sachant qu'un nombre premier n est un nombre entier qui n'est divisible que par 1 et par lui-même. L'algorithme A1 va donc consister en une boucle dans laquelle on va tester si le nombre n est divisible par 2, 3, ..., n-1.

Pseudo code :

```
Algorithme A1 ;
début
    premier = vrai ;
    i = 2 ;
    tant que (i <= n-1) et premier faire
        si (n mod i = 0) alors
            premier = faux
        sinon
            i = i+1 ;
fin.
```

FIGURE 1 – pseudo code d'algorithme 1

Complexité de l'algorithme 01 : Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à n-2. **La complexité est donc en O(n).**

Algorithme 02 : Sachant que si n est divisible par 2, il est aussi divisible par n/2 et s'il est divisible par 3, il est aussi divisible par n/3. De manière générale, si n est divisible par i pour i = 1 ... [n/2] où [n/2] dénote la partie entière de n/2, il est aussi divisible par n/i.

Pseudo code :

```
Algorithme A2 ;
début
      premier = vrai ;
      i = 2 ;
      tant que (i <= [n/2]) et premier faire
            si (n mod i = 0) alors
                  premier = faux
            sinon
                  i = i+1 ;
fin.
```

FIGURE 2 – pseudo code d'algorithme 2

Complexité de l'algorithme 02 : Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à n/2 -2. **La complexité est donc en O(n).**

Algorithme 03 : Si n est divisible par x, il est aussi divisible par n/x. Il serait intéressant d'améliorer A2 en ne répétant le test de la divisibilité que jusqu'à x = n/x.

Pseudo code :

```
Algorithme A3 ;
début
      premier = vrai ;
      i = 2 ;
      tant que i <= √n et premier faire
            si (n mod i = 0) alors
                  premier = faux
            sinon
                  i = i+1 ;
fin.
```

FIGURE 3 – pseudo code d'algorithme 3

Complexité de l'algorithme 03 : Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à (Sqrt[n])-1. **La complexité est donc en O($\sqrt{n}$).**

Algorithme 04 : **Dans le cas où n est impair, il ne faut tester la divisibilité de n que par les nombres impairs.**

Pseudo code :

```
Algorithme A4 ;
début
premier = vrai ;
si (n <> 2) et (n mod 2 = 0)
    alors premier = faux
    sinon si ( n <> 2) alors
            début
                i=3 ;
                tant que (i <= n-2) et premier
                faire
                    si (n mod i = 0) alors premier = faux sinon i = i+2 ;
                fin
            fin.
```

FIGURE 4 – pseudo code d'algorithme 4

Complexité de l'algorithme 04 : Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à [n/2] -2. La complexité est donc en O(n).

Algorithme 05 : À partir des algorithmes A2 et A4 on déduit.

Pseudo code :

```
Algorithme A5;
début
    premier = vrai ;
    si (n <> 2) et (n mod 2 = 0) alors
        premier = faux
    sinon si ( n <> 2) alors
      début
        i=3 ;
       tant que (i <= [n/2]) et premier faire
            si (n mod i = 0) alors premier = faux sinon i = i+2 ;
       fin
fin.
```

FIGURE 5 – pseudo code d'algorithme 5

Complexité de l'algorithme 05 : Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à [n/4] -1. La complexité est donc en O(n).

Algorithme 06 : **À partir des algorithmes A3 et A4 on déduit.**

Pseudo code :

```
Algorithme A6 ;
début
    premier = vrai ;
    si (n <> 2) et (n mod 2 = 0) alors premier = faux
    sinon si ( n <> 2) alors
        début
            i=3 ;
            tant que (i <= √n) et premier faire
                si (n mod i = 0) alors premier = faux sinon i = i+2 ;
            fin
        fin.
```

FIGURE 6 – pseudo code d'algorithme 6

Complexité de l'algorithme 06 : **Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à (Sqrt[n]/2)-1. La complexité est donc en O(**$\sqrt{n}$).

0.2 Mesure du temps d'exécution.

0.2.1 Question 01

Mesurer les temps d'exécution T pour chacun des six algorithmes en faisant des tests sur des nombres premiers ayant au plus 12 chiffres.

Réponse 01 :

Table des résultats d'exécution :

Nmbr Premiers	Algo 1(ms)	Algo 2(ms)	Algo 3(ms)	Algo 4(ms)	Algo 5	Algo 6
24512053	0.000	0.000	0.000	0.000	0.00ms	0.00ms
1705483033	17.000	9.000	8.000	4.000	0.00ms	0.00ms
333250198343	2488.000	1912.000	1621.000	848.000	0.00ms	0.00ms

TABLE 1 – Résultats d'exécution des nombres premiers d'une longueur inférieur ou égal à 12 avec les six algorithmes

Représentation graphique :

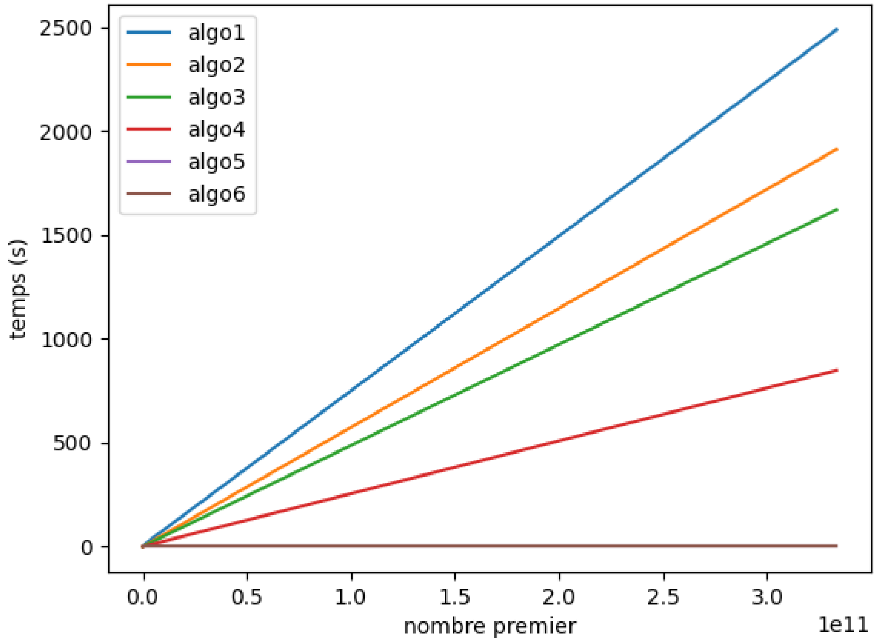

FIGURE 7 – Représentation graphique des résultats d'exécution des nombres premiers d'une longueur inférieur ou égal à 12 avec les six algorithmes :

0.2.2 Question 02

Pour une même longueur de nombres (20 chiffres par longueur), dresser la table des temps d'exécution que vous obtenez, puis donner un graphique traduisant les résultats obtenus. Que pouvez-vous conclure ?[2]

Réponse 02 :

Table des temps d'exécution :

Nbr Premiers	Algo 1 (s)	Algo 2 (s)	Algo 3 (s)	Algo 4 (s)	Algo 5	Algo 6
1054421323	2607.000113	1315.999985	1312.000036	647.000015	0.000s	0.000s
1073494831	2456.000090	1205.000043	1195.999980	587.000012	0.000s	0.000s
1101498319	2460.999966	1236.999989	1213.000059	611.000001	0.000s	0.000s
1106169257	2483.000040	1220.000029	1233.999968	611.999989	0.000s	0.000s
1143400393	2572.000027	1259.999990	1271.999955	662.999988	0.000s	0.000s
1175435993	2642.999887	1319.000006	1345.999956	670.000017	0.000s	0.000s
1233532831	2779.000044	1409.000039	1368.000031	697.000027	0.000s	0.000s
1245417659	2813.999891	1388.000011	1399.999976	699.999988	0.000s	0.000s
1327525747	2971.999884	1460.999966	1519.000053	759.999990	0.000s	0.000s
1338832499	3016.000032	1519.999981	1524.999976	787.000000	0.000s	0.000s
1350670283	3144.999981	1541.000009	1500.000000	762.000024	0.000s	0.000s
1412231461	3181.999922	1588.999987	1628.000021	781.000018	0.000s	0.000s
1429113289	3203.000069	1625.000000	1616.000056	819.000006	0.000s	0.000s
1541727167	3496.999979	1748.000026	1750.000000	874.000013	0.000s	0.000s
1581108167	3619.999886	1766.999960	1764.999986	898.999989	0.000s	0.000s
1582662937	3555.999994	1805.999994	1782.999992	908.999979	2.000s	0.000s
1611260639	3638.999939	1797.000051	1833.999991	902.000010	0.000s	0.000s
1662421357	3723.000050	1860.999942	1871.999979	940.999985	0.000s	0.000s
1820443931	4214.000225	2167.999983	2094.000101	1044.999957	0.000s	0.000s
1885404947	4293.000221	2141.999960	2151.000023	1047.000051	0.000s	0.000s

Table 2 – Résultats d'exécution en prenant 20 nombres premiers de même longueur avec les six algorithmes

Représentation graphique :

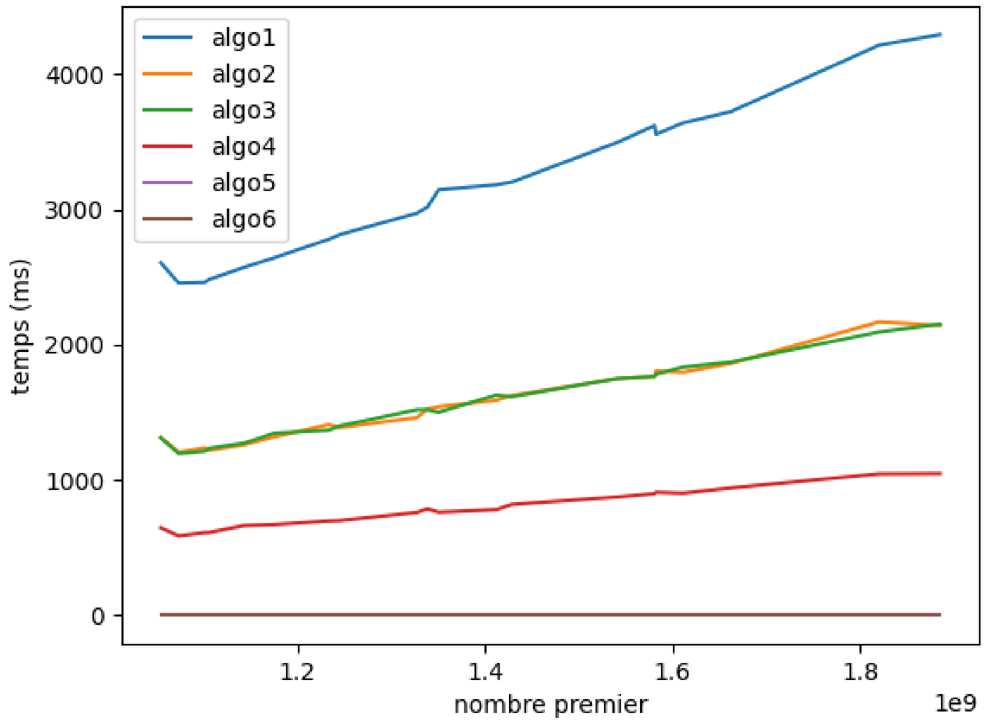

FIGURE 8 – Représentation graphique des résultats d'exécution en prenant 20 nombres premiers de même longueur avec les six algorithmes

Analyse :

D'abord, on remarque à travers ce dernier graphe [Figure 8] que pour les nombres de même longueur [la longueur est quelconque par exemple pour ces résultats d'exécution on a choisit des nombres d'une longueur = 10 chiffres] les courbes des algorithmes 1, 2, 3 et 4 sont très similaires. Et même pour les algorithmes 5 et 6 sont totalement identiques.

Ensuite, on remarque que pour les algorithmes 1, 2, 3 et 4... plus les nombres données augmentent plus les temps d'exécution augmentent et donc on peut dire qu'ils sont dans une relation de corrélation directe avec la longeur des nombres, mais par rapport aux algorithmes 5 et 6 dans notre cas [Des nombres de 10 chiffres] leur temps d'exécution se rapproche de 0.00000 s.

L'analyse a révélé que les Algorithmes 5 et 6 vue à leur temps d'exécution sont très rapides par rapport les autres algorithmes et surtout algorithme 1 qui est le plus long a cause du grands nombres d'itérations exécutées. Et donc les Algorithmes 5 et 6 beaucoup plus optimaux et efficaces.

0.2.3 Question 03

Pour des longueurs différentes de nombres, allant de 6 à 12, exécuter les 6 programmes 50 fois (si possible) et reporter la moyenne du temps d'exécution. Dresser la table des résultats numériques puis le graphique correspondant. Que pouvez-vous conclure ?

Réponse 03 :

Table des résultats numériques :

Nmbr Premiers	Algo 1 (s)	Algo 2 (s)	Algo 3 (s)	Algo 4 (s)	Algo 5 (s)	Algo 6 (s)
24512069	**0.00000**	**0.00000**	**0.00000**	**0.00000**	**0.00000**	**0.00000**
615478553	**6.66667**	**7.00000**	**3.00000**	**1.00000**	**0.00000**	**0.00000**
1170530411	**10.33333**	**10.33333**	**5.00000**	**2.00000**	**0.00000**	**0.00000**
61547854723	**438.33333**	**448.00000**	**833.33333**	**145.66667**	**0.00000**	**0.00000**

Table 3 – Résultats d'exécution des nombres premiers d'une longueur inférieur ou égal à 12, 50 fois, avec les six algorithmes

Représentation graphique :

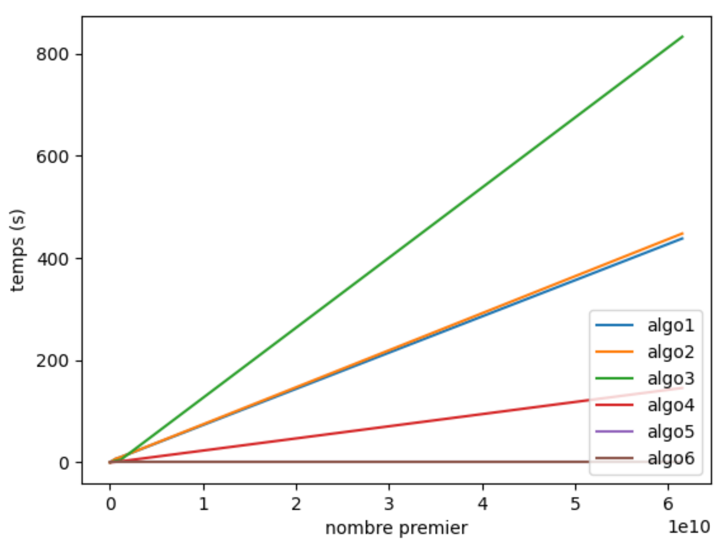

Figure 9 – Représentation graphique des résultats d'exécution des nombres premiers d'une longueur inférieur ou égal à 12, 50 fois, avec les six algorithmes

Analyse :

D'abord, on remarque à travers ce dernier graphe [Figure 9] après le calcul de la moyenne du temps d'exécution pour des longueurs différentes de nombres, allant de 6 à 12, [par exemple pour ces résultats d'exécution on a choisit des nombres d'une longueur = 8, 9, 10 et 11 chiffres] que les courbes des algorithmes 1, 2, 3 et 4 sont très similaires. Et même pour les algorithmes 5 et 6 sont totalement similaires.

Ensuite, on remarque que pour les algorithmes 1, 2, 3 et 4... plus les nombres données augmentent plus les temps d'exécution augmentent et donc on peut dire qu'ils sont dans une relation de corrélation directe, mais par rapport aux algorithmes 5 et 6 dans notre cas [Des nombres de 10 chiffres] leur temps d'exécution se rapproche de 0.00000 s.

L'analyse a révélé que l'exécution de ces algorithmes plusieurs fois n'affecte pas la complexite de chacun des algorithmes, et les Algorithmes 5 et 6 vue à leur temps d'exécution sont très rapides par rapport les autres algorithmes et surtout algorithme 1 qui est le plus long a cause du grands nombres d'instructions exécutées. Et donc les Algorithmes 5 et 6 beaucoup plus optimaux et efficaces.

0.3 Environnement expérimental

Caractéristiques de la machine	Système d'exploitation	CPU	RAM	Version compilateur C
BENKOUITEN Aymen	**Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.**	**Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz**	**8,00 GO**	**codeblocks-20.03mingw**
KENAI Imad Eddine	**Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.**	**Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz**	**16,00 GO**	**codeblocks-8.1.0mingw**
MALKI Omar Chouaab	**Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.**	**Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz**	**8,00 GO**	**codeblocks-20.03mingw**
MEKKAOUI Mohamed	**Système d'exploitation 64 bits, processeur x64. Windows 11 Home V-22H2.**	**Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz**	**16,00 GO**	**codeblocks-20.03mingw**

TABLE 4 – Environnement expérimental

0.4 Répartition des tâches :

- **Écriture du code source : KENAI Imad.**
- **La rédaction du rapport et son organisation avec Latex : BENKOUITEN Aymen.**
- **Exécution des taches :MEKKAOUI Mohamed et MALKI Omar Chouaab.**
- **NB : Tous ce travail est fait durant un meet entre le team et toutes les réponses et les analyses effectuer sur les graphes sont bien été discuter avant la rédaction de rapport.**

Conclusion Générale

Les résultats de notre analyse des résultats obtenus ainsi que le calcul théorique de la complexité temporelle des six algorithmes qui permis de vérifier si un nombre est premier ou pas, montrent d'abord que la machine a aucune affectation a la complixité d'un algorithme et aussi l'analyse nous a permis de comparer l'efficacité des six algorithmes et établir lequel d'entre eux est le optimal et le plus rapide.

On conclut que plus la complexité tend vers l'exponentielle plus le temps d'exécution augmentent d'une façon plus rapide et donc on peut dire que plus la complexité tend vers l'exponentielle que l'exécution devient très pénible et très défficile à réaliser.

Annexe

0.5 Code source des six d'algorithmes :

Algorithme 01 :

```c
int algo1 (unsigned long long int nbr){

        for (unsigned long long int i=2 ; i<=nbr-1; i++){
            if(nbr % i == 0) return 0;
        }
        //printf("premier   ") ;
        return 1;
}
```

FIGURE 10 – code source d'algorithme 1

Algorithme 02 :

```c
int algo2 (unsigned long long int nbr){

        for ( unsigned long long int i=2 ; i<=nbr/2; i++){
            if(nbr % i == 0) return 0;
        }
        //printf("premier   ") ;
return 1;
}
```

FIGURE 11 – code source d'algorithme 2

Algorithme 03 :

```c
int algo3 (unsigned long long int nbr){

	if((nbr != 2) && (nbr%2 == 0))
		return 0;
	else{
		if (nbr != 2){
			for ( unsigned long long int i=3 ; i<=nbr-2; i+=2){
				if(nbr % i == 0) return 0;
			}
		}
	}
//printf("premier   ") ;
return 1;
}
```

FIGURE 12 – code source d'algorithme 3

Algorithme 04 :

```c
int algo4 (unsigned long long int nbr){

	if((nbr != 2) && (nbr%2 == 0))
		return 0;
	else{
		if (nbr != 2){
			for (  unsigned long long int i=3 ; i<=nbr/2; i+=2){
				if(nbr % i == 0) return 0;

			}
		}
	}
//printf("premier   ") ;
return 1;
}
```

FIGURE 13 – code source d'algorithme 4

Algorithme 05 :

```c
int algo5 (unsigned long long int nbr){

        for (unsigned long long int i=2 ; i<=sqrt(nbr); i++){
                if(nbr % i == 0) return 0;
        }
//printf("premier   ") ;
return 1;
}
```

FIGURE 14 – code source d'algorithme 5

Algorithme 06 :

```c
int algo6 (unsigned long long int nbr){

        if((nbr != 2) && (nbr%2 == 0))
                return 0;
        else{
                if (nbr != 2){
                        for ( unsigned long long int i=3 ; i<=sqrt(nbr); i+=2){
                                if(nbr % i == 0) return 0;
                        }
                }
        }
//printf("premier   ") ;
return 1;
}
```

FIGURE 15 – code source d'algorithme 6

la fonction pour executer tache 1 :

```c
void tache01(){
// {5297,389299,34583291,9934239587,979020560239}
unsigned long long int primes[5]= {7919,104729,24512053,1705483033,333250198343};

double delta;

clock_t t1,t2;


for (int i =0 ;i<5;i++){

printf("------------------------------------\n");
printf("le nombre premier  => %llu\n",primes[i]);
printf("------------------------------------\n\n");
///////////


  ///////////

  printf("algo 1 :");

  t1 =clock();

  algo1(primes[i]);

  t2 = clock();

  delta = (t2-t1)/CLOCKS_PER_SEC;

  printf(" %f \n\n",delta);

  ///////////////
```

FIGURE 16 – code source de la fonction tache 01

la fonction pour executer tache 2 :

On peut dire que c'est le meme code de la tache 01,on change juste les valeurs du tableau primes.

```c
void tache02(){
// {5297,389299,34583291,9934239587,979020560239}
unsigned long long int primes[20]= {1054421323,1073494831,1101498319,1106169257,1143400393,1175435993,1233532831,1245417659,1327525747,
1338832499,1350670283,1412231461,1429113289,1541727167,1581108167,1582662937,1611260639,1662421357,1820443931,1885404947};

double delta;

int t1,t2;

for (int i =0 ;i<20;i++){
printf("-----------------------------------\n");
printf("le nombre premier  => %llu\n",primes[i]);
printf("-----------------------------------\n\n");
//////////

printf("algo 1 :");

t1 =clock();

algo1(primes[i]);

t2 = clock();

delta = (float)(t2-t1)/CLOCKS_PER_SEC;

printf(" %lf \n\n",delta*1000);
```

FIGURE 17 – code source de la fonction tache 02

la fonction pour executer tache 3 :

```c
void tache03(){
// {5297,389299,34583291,9934239587,979020560239}

unsigned long long int primes[1]= {5297,389299,34583291,9934239587,979020560239};

double delta;

clock_t t1,t2;

double moy;


// for (int i =0 ;i<5;i++){
for (int i =0 ;i<5;i++){

printf("-----------------------------------\n");
printf("le nombre premier  => %llu\n",primes[i]);
printf("-----------------------------------\n\n");

//////////

moy = 0;

printf("algo 6 :");

for(int t=0;t<20;t++){
```

FIGURE 18 – code source de la fonction tache 03_1

```
for(int t=0;t<20;t++){

t1 =clock();

algo6(primes[i]);

t2 = clock();

delta = (float)(t2-t1)/CLOCKS_PER_SEC;

moy += delta;

}



printf(" %f \n\n",moy/20);

///////////////
```

FIGURE 19 – code source de la fonction tache 03_2

Bibliographie

[1] Complexité temporelle. `https://info.blaisepascal.fr/nsi-complexite-dun-algorithme`.

[2] Générateur des nombres premiers. `https://fr.numberempire.com/primenumbers.php`.