République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Electronique et d'Informatique
Département Informatique

Master Systèmes Informatiques intelligents

Module : Conception et Complexité des Algorithmes

Rapport de projet de TP

Réalisé par :
BENBACHIR Mohamed Amir, 191932021049
BOUCHOUL Bouchra, 191931081317
KHEMISSI Maroua, 191935007943
MEDJKOUNE Roumaissa, 191931081005

Année universitaire : 2022 / 2023

Table des matières

1 Introduction **2**

2 Les algorithmes utilisées **3**
2.1 Algorithme A1 3
2.2 Algorithme A2 4
2.3 Algorithme A3 5
2.4 Algorithme A4 6
2.5 Algorithme A5 7
2.6 Algorithme A6 8
2.7 L'evaluation theorique de la complexite 8

3 Environnement experimental **9**

4 Mesure du temps d'exécution des nombres premiers ayant au plus 12 chiffres **10**

5 Mesure du temps d'exécution des 20 nombres premiers ayant la meme taille **12**
5.1 Experimentation 12
5.2 Conclusion 13

6 Mesure du temps d'exécution apres 30 fois **14**

7 Conclusion **16**

8 Annexe **17**
8.1 Repartition des taches 17
8.2 main.c 18

9 Bibliographie **22**

Chapitre 1

Introduction

Un nombre premier est un nombre sans diviseurs autres que 1 et lui-même, par exemple : 2, 3, 5, 7, 11, 13, 17, 19.

Savoir rapidement si un nombre entier est un nombre premier ou non est un problème posé depuis des millénaire. De très nombreux algorithmes avancés ont été conceptualisés certains sont d'une complexité polynomiale.

Dans ce travail nous allons tester 6 différents algorithmes et mesurer la difficulté de chacun par étudier ses deux complexités théorique et expérimentale en évaluant le temps nécessaire pour l'effectuer sur un ordinateur en fonction de la taille des données.

Chapitre 2

Les algorithmes utilisées

2.1 Algorithme A1

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.

Nous pouvons le représenter via le pseudo code suivant :

Fonction A1(Entrée :n : entier ;)

```
Variables :
i : entier;
premier : bool;
début
    premier ← vrai;
    i ← 2;
    tant que premier ← vrai ET i ≤ n − 1 faire
        si n mod 2 == 0 alors
            premier ← false;
        sinon
            i++;
        fin si
    fin tq
fin
```

2.2 Algorithme A2

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.

Nous pouvons représenter une optimisation de A1 via le pseudo code suivant :

Fonction A2(Entrée :n : entier ;)

```
Variables :
i : entier;
premier : bool;
début
    premier ← vrai;
    i ← 2;
    tant que premier ← vrai ET i ≤ n/2 faire
        si n mod 2 == 0 alors
            premier ← false;
        sinon
            i++;
        fin si
    fin tq
fin
```

2.3 Algorithme A3

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.

Nous pouvons représenter cette amélioration de A2 via le pseudo code suivant :

Fonction A3(Entrée :n : entier ;)

```
Variables :
i : entier;
premier : bool;
début
    premier ← vrai;
    i ← 2;
    tant que premier ← vrai ET i ≤ √n faire
        si n mod 2 == 0 alors
            premier ← false;
        sinon
            i++;
        fin si
    fin tq
fin
```

2.4 Algorithme A4

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

Nous pouvons le représenter via le pseudo code suivant :

Fonction A4(Entrée :n : entier ;)

```
Variables :
i : entier;
premier : bool;
début
    premier ← vrai;
    si n <> 2 ET n mod 2 = 0 alors
        premier ← false;
    fin si
    sinon
        si n <> 2 alors
            i ← 3;
            tant que premier ← vrai ET i ≤ n − 2 faire
                si n mod i == 0 alors
                    premier ← false;
                sinon
                    i ← i + 2;
                fin si
            fin tq
        fin si
    fin si
fin
```

2.5 Algorithme A5

Une hybridation des algorithmes A2 et A4.

Nous pouvons le représenter via le pseudo code suivant :

Fonction A5(Entrée :n : entier ;)

Variables :

i : entier;

premier : bool;

début

$premier \leftarrow vrai$;

si $n <> 2 \ \ ET \ \ n \, mod \, 2 = 0$ **alors**

$premier \leftarrow false$;

fin si

sinon

si $n <> 2$ **alors**

$i \leftarrow 3$;

tant que $premier \leftarrow vrai \ \ ET \ \ i \ \leq n/2$ **faire**

si $n \ \ mod \ \ i == 0$ **alors**

$premier \leftarrow false$;

sinon

$i \leftarrow i + 2$;

fin si

fin tq

fin si

fin si

fin

2.6 Algorithme A6

Une hybridation des algorithmes A3 et A4.

Nous pouvons le représenter via le pseudo code suivant :

Fonction A6(Entrée :n : entier ;)

Variables :

i : entier;

premier : bool;

début

- $premier \leftarrow vrai$;
- **si** $n <> 2 \ \ ET \ \ n \, mod \, 2 = 0$ **alors**
  - $premier \leftarrow false$;
- **fin si**
- **sinon**
  - **si** $n <> 2$ **alors**
    - $i \leftarrow 3$;
    - **tant que** $premier \leftarrow vrai \ \ ET \ \ i \leq \sqrt{n}$ **faire**
      - **si** $n \ mod \ i == 0$ **alors**
        - $premier \leftarrow false$;
      - **sinon**
        - $i \leftarrow i + 2$;
      - **fin si**
    - **fin tq**
  - **fin si**
- **fin si**

fin

2.7 L'evaluation theorique de la complexite

Algorithme	Nombre maximum d'itération (en fonction de n)	Complexité théorique	Nombre réel d'itération avec n=69524913
A1	n-2	O(n)	69524913
A2	[n/2]-1	O(n)	34762455
A3	$\sqrt{n} - 1$	O($\sqrt{n}$)	8337
A4	[n/2]-1	O(n)	34762455
A5	[n/4]-1	O(n)	17381227
A6	$[\sqrt{n}/2] - 1$	O($\sqrt{n}$)	4168

Chapitre 3

Environnement experimental

Afin d'éviter de laisser transparaître les différences de puissances de nos machines respectives , on a utilisee une seule machine tout au long du TP avec comme des specification :

RAM : 8GO

Processeur : Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz

Type de Systeme : Système d'exploitation Windows 10 64 bits

Environement de développement : Visual Studio Code

Langage de programmation : C

Chapitre 4

Mesure du temps d'exécution des nombres premiers ayant au plus 12 chiffres

Le tableau suivant représente le temps d'exécutions des six algorithmes pour des nombres ayant des longueurs différentes (6,7,8,9,10,11,12)

Nombre premier	A1	A2	A3	A4	A5	A6
314159	0.010000	0.001000	0.000000	0.001000	0.000000	0.000000
4480649	0.066000	0.022000	0.000000	0.020000	0.012000	0.000000
50943779	0.440000	0.271000	0.000000	0.218000	0.129000	0.000000
999999937	8.502600	4.378000	0.000633	7.151200	2.185066	0.001267
5915587277	8.392000	5.146000	0.001000	5.180000	2.375000	0.000000
41996139943	371.614000	208.341000	0.003000	199.646000	100.515000	0.003000
100123456789	912.519000	502.751000	0.005000	464.866000	311.032000	0.003000

La figure suivante (voir Figure 4.1) représente les résultats d'executions des six algoritmes pour ces nombres.

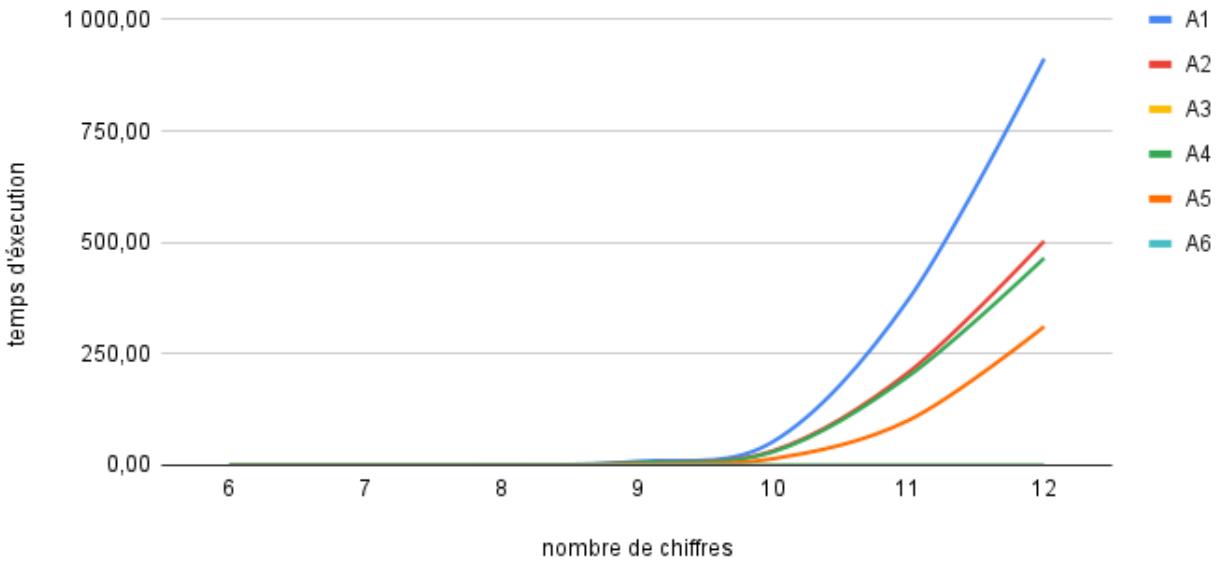

FIGURE 4.1 – Temps d'exécution des six algorithmes sur 7 nombres de chiffres différents

Chapitre 5

Mesure du temps d'exécution des 20 nombres premiers ayant la meme taille

5.1 Experimentation

Le tableau suivant représente les temps d'exécution en nanoseconde de l'algorithme selon la variation des nombres premiers x1..x2 en ordre croissante.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1500450271	18790481837	1979339339	2030444287	2860486313	3267000013	3367900313	3628273133	4093082899	4392489041
x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
5038465463	5463458053	5555333227	5754853343	5915587277	6668999101	7896879971	8278487999	9471413089	9576890767

x	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
A1(s)	16.113	20.109	19.987	20.75	44.815	59.697	48.246	55.874	68.331	70.699
A2(s)	8.206	9.687	10.146	15.69	26.938	28.324	24.693	31.044	37.252	34.662
A3(s)	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0	0.001
A4(s)	8.198	10.035	13.046	17.019	14.282	20.343	20.192	22.26	22.948	25.089
A5(s)	4.429	5.178	5.594	5.062	7.355	8.195	8.504	9.544	10.064	11.134
A6(s)	0	0	0	0	0.001	0.001	0.001	0.001	0.001	0.001

	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
A1(s)	85.505	59.221	84.891	68.61	89.415	94.345	97.925	93.204	114.139	152.4
A2(s)	28.492	48.4	48.453	50.096	38.766	34.916	42.29	48.979	56.765	106.045
A3(s)	0.002	0.001	0.002	0.001	0.001	0.002	0.004	0.004	0.003	0.002
A4(s)	29.859	34.864	35.669	34.387	33.805	36.295	44.952	44.042	51.325	52.909
A5(s)	12.632	13.641	14.342	14.516	14.809	20.352	21.77	21.964	26.168	24.905
A6(s)	0.001	0.001	0	0.001	0.002	0.001	0.001	0	0.001	0.001

La figure suivante (voir Figure 6.1) représente l'évolution du temps d'exécution selon l'algorithme utilisee.

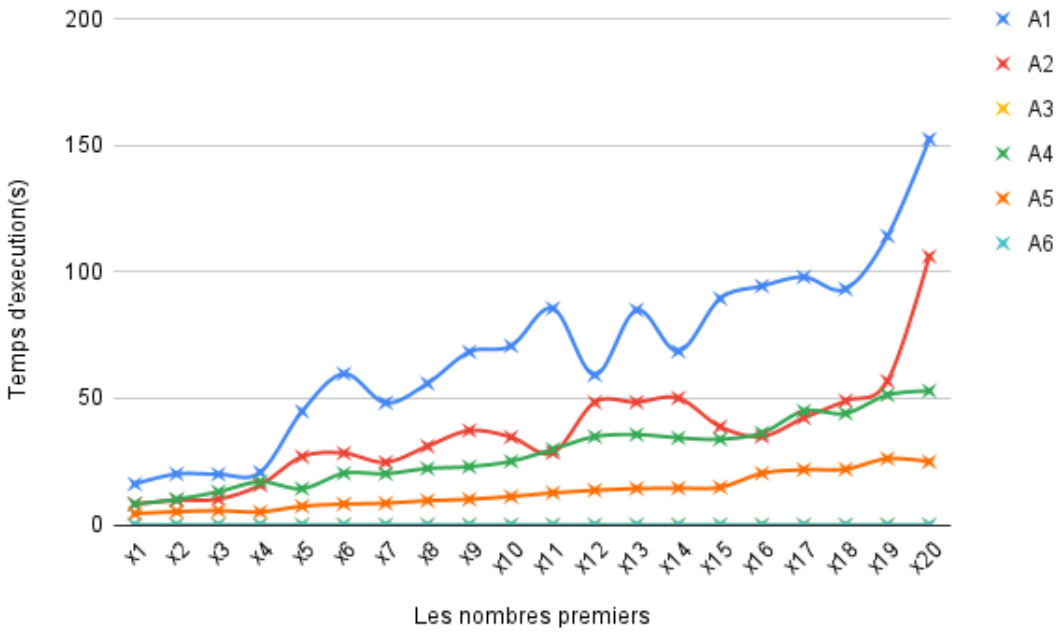

FIGURE 5.1 – Temps d'exécution du programme selon l'algorithme utilisee

Depuis le graphe, les courbes sont sous forme des arcs ascendants, on observe que le temps d'exécution évolue de manière presque linéarithmique avec l'augmentation des nombres premiers , avec A6 et A3 comme des meilleurs temps d'execution qui correspond bien à la complexité théorique calculée auparavant , On a pas obtenu une droite linéaire car les testes étaient peu et aléatoires.

5.2 Conclusion

Les algorithmes A6 , A3 proposent une complexite optimale pour les tests de primalites quelque soit la taille de nombre. et pour ameliorer les resultats on veut tester les algorithmes plusieurs fois ce qu'on va voir dans le chapitre suivant .

Chapitre 6

Mesure du temps d'exécution apres 30 fois

Le tableau suivant représente les temps d'exécution des six algorithmes 30 fois d'itérations sur des nombres ayant des longueurs différentes.

Nombre premier	A1	A2	A3	A4	A5	A6
314159	0.003033	0.001400	0.000000	0.002200	0.000900	0.000033
4480649	0.040700	0.019367	0.000033	0.023600s	0.005822	0.000033
50943779	0.479900	0.231167	0.000200	0.332567	0.011132	0.000500
999999937	8.502600	4.378000	0.000633	7.151200	2.185066	0.001267
5915587277	49.763900	27.214133	0.001567	25.666167	12.863833	0.001767
41996139943	795.541433	186.160400	0.003767	247.778366	92.497733	0.004233
100123456789	957.670200	519.229133	0.006333	466.633330	268.685000	0.007533

La figure suivante (voir Figure 6.1) représente l'évolution du temps d'exécution selon l'algorithme utilisee.

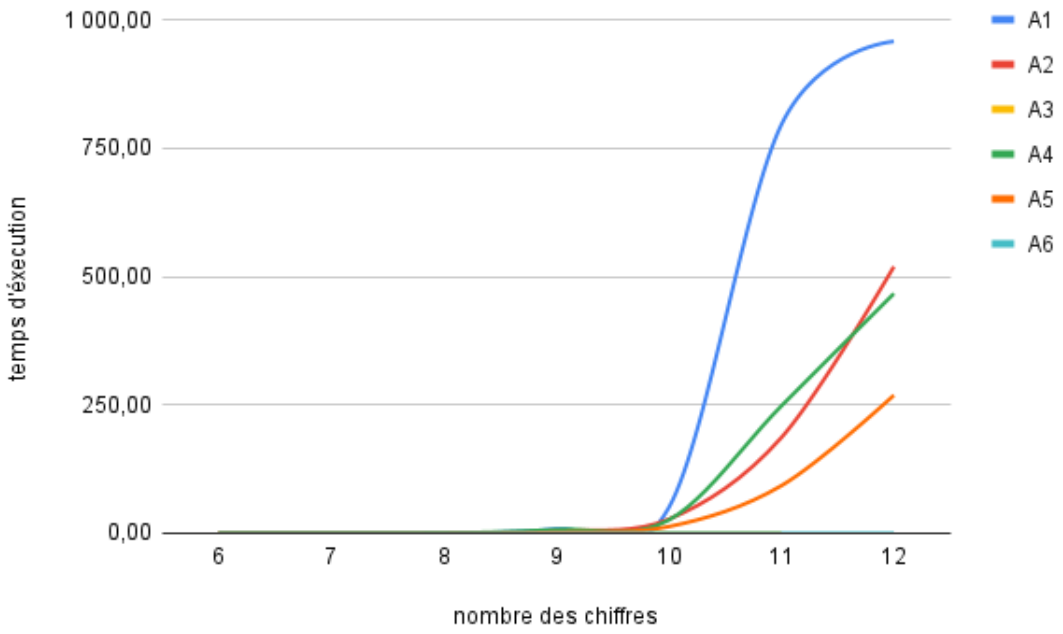

FIGURE 6.1 – Temps d'exécution des six algorithmes aprés 30 exécutions sur des nombres de longueurs différentes

A partir du graphe ci dessus, on constate qu'il y a une relation de correlation directe entre le nombre de chiffres et le temps d'éxecution moyen (plus que le nombre de chiffres auguemente plus que le temps d'éxecution auguemente)ce qui est le cas pour les algorithmes A1,A2,A4,A5.
Pour le cas des algorithmes A3 et A6, on remarque que le résultat d'exécution tend vers 0 malgré l'augmentation du nombre de chiffres.

Chapitre 7

Conclusion

Après une mise à l'échelle des graphes afin d'éviter de laisser transparaître les différences de puissances de nos machines respectives, nous constatons que il y a une relation de correlation directe entre le nombre de chiffres et la moyenne du temps d'éxecution (plus que le nombre de chiffres augemente plus que le temps d'éxecution augemente) qui est le cas pour les algorithmes A1,A2,A4,A5. par contre les algorithmes A6 , A3 proposent une complexite optimale pour les tests de primalites quelque soit la taille de nombre le temps d'execution est toujours minimal.

Chapitre 8

Annexe

8.1 Repartition des taches

Member	Tache
BEBNBACHIR Mohamed Amir	Analyse du temps d'exécution des nombres premiers ayant au plus 12 chiffres
BOUCHOUL Bouchra	Execution et mesure de temps d'execution , preparation des donnees
KHEMISSI Maroua	Analyse du temps d'exécution des 20 nombres premiers ayant la meme taille
MEDJKOUNE Roumaissa	Analyse du temps d'execution apres 30 tests
en collaboration de tous les membres	implémenter les algorithmes et rédige le rapport

8.2 main.c

```c
#include <math.h>
#include <stdio.h>
#include <time.h>
//Sol1 On essaie les divisions par 2,3..n-1 et on s'arrête au premier diviseur
int A1(unsigned long long int n) {
  for (unsigned long long int i = 2; i <= n-1; i++) {
    if (n % i == 0) {
      return 0;
    }
  }
  return 1;
}
//Sol2 pour accélérer la recherche, on essaie les divisions par 2,3..n/2 et on s'arrête au premier diviseur ou n/2
int A2(unsigned long long int n) {
  for (unsigned long long int i = 2; i <= (n / 2); i++) {
    if (n % i == 0) {
      return 0;
    }
  }
  return 1;
}
//Sol3 pour accélérer encore la recherche on essaie les divisions par 2,3..n et on s'arrête au premier diviseur ou n
int A3(unsigned long long int n) {
  for (unsigned long long int i = 2; i <= sqrt(n); i++) {
    if (n % i == 0) {
      return 0;
    }
  }
  return 1;
}

//Sol4 On élimine les multiples de 2 et on essaie pas les nombres pairs ,on s'arrête au premier diviseur
int A4(unsigned long long int n) {
  if (n != 2 && n % 2 == 0) {
    return 0;
```

```
  }
  if (n != 2) {
    for (unsigned long long int i = 3; i <= n - 2; i += 2) {
      if (n % i == 0) {
        return 0;
      }
    }
  }
  return 1;
}
//Sol5 On élimine les multiples de 2 et on essaie pas les nombres pairs ,on s'arrête au premier diviseur ou n/2
int A5(unsigned long long int n) {
  if (n != 2 && n % 2 == 0) {
    return 0;
  }
  if (n != 2) {
    for (unsigned long long int i = 3; i <= n / 2; i += 2) {
      if (n % i == 0) {
        return 0;
      }
    }
  }
  return 1;
}
//Sol6 On élimine les multiples de 2 et on essaie pas les nombres pairs ,on s'arrête au premier diviseur ou n
int A6(unsigned long long int n) {
  if (n != 2 && n % 2 == 0) {
    return 0;
  }
  if (n != 2) {
    for (unsigned long long int i = 3; i <= sqrt(n); i += 2) {
      if (n % i == 0) {
        return 0;
      }
    }
  }
  return 1;
}
```

```
74  int main(void) {
75    unsigned long long int T[7] = {314159,      4480649,    50943779,
76                                   999999937,   5915587277, 41996139943,
77                                   100123456789};
78    unsigned long long int T2[20] = {1500450271,
79  1879048183,
80  1979339339,
81  2030444287,
82  2860486313,
83  3267000013,
84  3367900313,
85  3628273133,
86  4093082899,
87  4392489041,
88  5038465463,
89  5463458053,
90  5555333227,
91  5754853343,
92  5915587277,
93  6668999101,
94  7896879971,
95  8278487999,
96  9471413089,
97  9576890767};
98    unsigned long long int big = 1276478784635844147 1U;
99    clock_t t1, t2;
100   double delta, s;
101   int r = 0;
102   FILE *fptr, *f;
103   f= fopen("data.txt","w");
104   printf("A1\n");
105   for (int j = 0; j <20; j++ ){
106     s = 0;
107     for (int k = 0; k < 1; k++) {
108       t1 = clock();
109       r += A6(T2[j]);
110       t2 = clock();
111       delta = (double)(t2 - t1) / CLOCKS_PER_SEC;
112       s += delta;
113     }
```

```
114       fprintf(f, "%f\n", s);
115       printf("x%i %f s\n", j + 1, s);
116     }
117     fclose(f);
118     fptr = fopen("A5test.txt", "w");
119     printf("A5test1\n");
120     for (int i = 0; i < 7; i++) {
121       s = 0;
122
123       for (int k = 0; k < 1; k++) {
124         t1 = clock();
125         r += A5(T[i]);
126         t2 = clock();
127         delta = (double)(t2 - t1) / CLOCKS_PER_SEC;
128         s += delta;
129       }
130       fprintf(fptr, "%i digit number %f s\n", i + 6, s);
131       printf("%i digit number %f s\n", i + 6, s);
132     }
133     t1 = clock();
134     r += A6(big);
135     t2 = clock();
136     delta = (double)(t2 - t1) / CLOCKS_PER_SEC;
137     fprintf(fptr, "%i digit number %f s\n", 20, delta);
138     printf("%i digit number %f s %i\n", 20, delta, r);
139     fclose(fptr);
140     return 0;
141 }
```

Chapitre 9

Bibliographie

[1] https ://www.pourlascience.fr/sd/mathematiques/savoir-si-un-nombre-est-premier-facile-4919.php