COMPLEXITÉ

Master 1 IL
Groupe 2
2018

Rapport de TP N°1 COMPLEXITÉ : TEST DE PRIMALITÉ

BOUDOUR Mehdi / 201500008386/ TP: Test de Primalité

[ALGORITHMIQUE AVANCÉE ET COMPLEXITÉ]

E-mail : geronimotoutcourt@gmail.com

Ce document présent les solutions en 5 étapes : (1) les algorithmes écris en pseudo-code. (2) le calcul de la complexité au pire des cas. (3) Implémentation de l'algorithme en langage C. (4) capture de l'exécution de l'algorithme. (5) représentation graphique de l'évolution du temps d'exécution en fonction de N. Le programme C complet contenant les détails (affichage, calcul du temps d'exécution,…) d'implémentation est présenté à la fin du document.

I. Algorithme 1(A1) : Approche naïve :

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.

Algorithme :

```
FONCTION ALGORITHME1(N:ENTIER) : BOOLÉEN
    I: ENTIER;
DEBUT
    POUR I = 2 JUSQU'A N-1 FAIRE
        SI ( N MOD I = 0 ) ALORS
            RETOURNER (FAUX);
        FIN SI;
    FIN POUR;
    RETOURNER (VRAI);
FIN;
```

$\sum_{I=2}^{N-1} 1$

1

Complexité :

Au pire des cas : Le nombre ***N*** est premier ainsi la boucle s'itérera jusqu'à ***i=N-1*** car le test ne trouvera aucun nombre ***i*** ∈ { , x∈ ℕ / **2** ≤ x ≤ ***N-1***} qui soit diviseur de ***N***.

$$T(N)= \sum_{2}^{N-1} 1 +1 = (N\text{-}1 - 2 + 1) +1 = N\text{-}1 \sim O(N)$$

Implémentation : En langage C

```
int Algorithme1(int N)
{
    int i;
    for(i=2 ; i<= N-1 ; i++)
    {
        if(N%i == 0) return 0;
    }
    return 1;
}
```

Exécution :

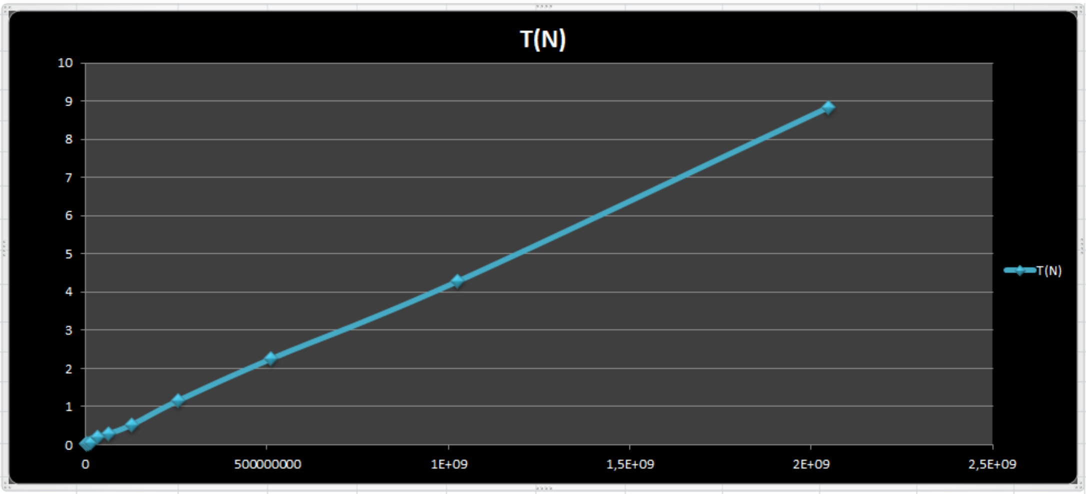
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Algorithme1** premier = Retour de **Algorithme1**).*

```
Execution de l'Algorithme 1 :
N = 1000003.0000001      T= 0.0190001\t ,premier= 1
N = 2000003.0000001      T= 0.0210001\t ,premier= 1
N = 4000037.0000001      T= 0.0250001\t ,premier= 1
N = 8000009.0000001      T= 0.0340001\t ,premier= 1
N = 16000057.0000001     T= 0.0680001\t ,premier= 1
N = 32000011.0000001     T= 0.2080001\t ,premier= 1
N = 64000031.0000001     T= 0.2940001\t ,premier= 1
N = 128000003.0000001    T= 0.5330001\t ,premier= 1
N = 256000001.0000001    T= 1.1640001\t ,premier= 1
N = 512000009.0000001    T= 2.2560001\t ,premier= 1
N = 1024000009.0000001   T= 4.2770001\t ,premier= 1
N = 2048000011.0000001   T= 8.8390001\t ,premier= 1
```

Représentation Graphique :

Graphe du temps d'exécution en fonction de N.

ALGORITHME 1	
N	***T***
1000003	0,019
2000003	0,021
4000037	0,025
8000009	0,034
16000057	0,068
32000011	0,208
64000031	0,294
128000003	0,533
256000001	1,164
512000009	2,256
1,024E+09	4,277
2,048E+09	8,839



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

Améliorons A1 en sachant que N≡0[i] ⟺ i≤N/2 .

Algorithme :

```
FONCTION ALGORITHME2(N:ENTIER) : BOOLÉEN
    I: ENTIER;
DEBUT
    POUR I = 2 JUSQU'A N/2 FAIRE
        SI ( N MOD I = 0 ) ALORS
            RETOURNER (FAUX);
        FIN SI;
    FIN POUR;
    RETOURNER (VRAI);
FIN;
```

$\sum_{I=2}^{[N/2]} 1$

1

Complexité :

Au pire des cas : Le nombre ***N*** est premier ainsi la boucle s'itérera jusqu'à ***i=[N/2]*** ([N/2]= partie entière de N/2) car le test ne trouvera aucun nombre ***i*** ∈ { x , x∈ ℕ / ***2*** ≤ x ≤ ***N/2***} qui soit diviseur de ***N***.

$$T(N)= \sum_{2}^{[\frac{N}{2}]} 1 +1 = ([N/2] - 2 + 1) +1 = [N/2] \sim O(N)$$

Implémentation : En langage C

```c
int Algorithme2(int N)
{
    int i;
    for(i=2 ; i<= N/2 ; i++)
    {
        if(N%i == 0) return 0;
    }
    return 1;
}
```

Exécution :

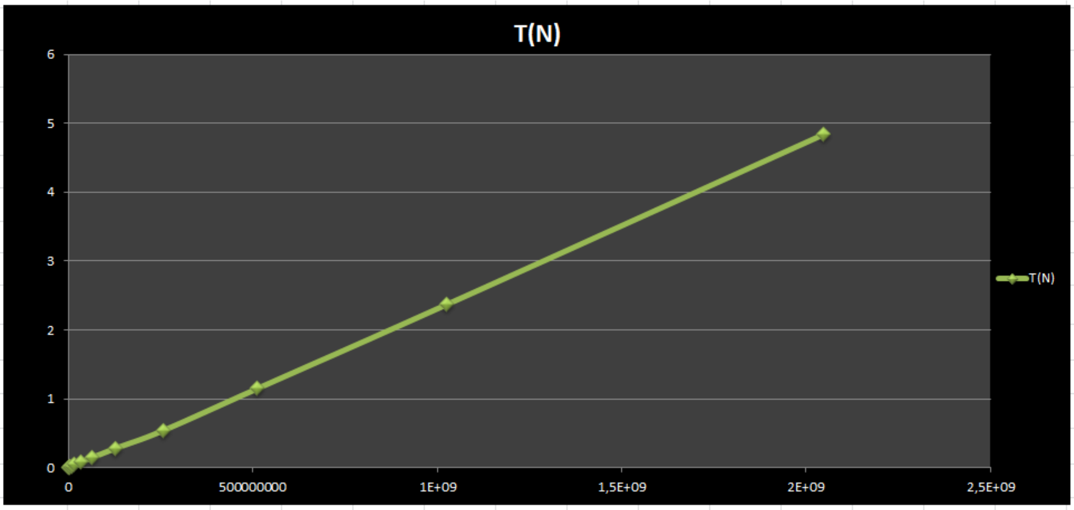
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Algorithme2** premier = Retour de **Algorithme2**).*

```
Execution de l'Algorithme 2 :
N = 1000003.0000001      T= 0.0020001\t ,premier= 1
N = 2000003.0000001      T= 0.0040001\t ,premier= 1
N = 4000037.0000001      T= 0.0120001\t ,premier= 1
N = 8000009.0000001      T= 0.0190001\t ,premier= 1
N = 16000057.0000001     T= 0.0490001\t ,premier= 1
N = 32000011.0000001     T= 0.0820001\t ,premier= 1
N = 64000031.0000001     T= 0.1480001\t ,premier= 1
N = 128000003.0000001    T= 0.2830001\t ,premier= 1
N = 256000001.0000001    T= 0.5390001\t ,premier= 1
N = 512000009.0000001    T= 1.1500001\t ,premier= 1
N = 1024000009.0000001   T= 2.3660001\t ,premier= 1
N = 2048000011.0000001   T= 4.8390001\t ,premier= 1
```

Représentation Graphique :

Graphe du temps d'exécution en fonction de N.

ALGORITHME 2	
N	T
1000003	0,002
2000003	0,004
4000037	0,012
8000009	0,019
16000057	0,049
32000011	0,082
64000031	0,148
128000003	0,283
256000001	0,539
512000009	1,15
1,024E+09	2,366
2,048E+09	4,839



III. Algorithme 3(A3) :

Utilisons la propriété selon laquelle la moitié des diviseurs d'un nombre ≤ $\sqrt{N}$ et l'autre ≥ $\sqrt{N}$.

Algorithme :

```
FONCTION ALGORITHME3(N:ENTIER) : BOOLÉEN
    I: ENTIER;
    R: RÉEL;
DEBUT
    R := SQRT(N);            <--  √N
    I :=2 ;                  <--  1
    TANT QUE (I <= R) FAIRE  <--  }
        SI ( N MOD I = 0 ) ALORS  }
            RETOURNER (FAUX);     }  ∑_{I=2}^{[√N]} 1
        FIN SI;                   }
        I := I+1 ;                }
    FIN TANTQUE;             <--  }
    RETOURNER (VRAI);        <--  1
FIN;
```

Complexité :

Au pire des cas : Le nombre ***N*** est premier ainsi la boucle s'itérera jusqu'à ***i=[$\sqrt{N}$]*** ([$\sqrt{N}$]= partie entière de $\sqrt{N}$) car le test ne trouvera aucun nombre ***i*** ∈ { x , $x \in \mathbb{N}$ / ***2*** ≤ x ≤ $\sqrt{N}$} qui soit diviseur de ***N*** dans la première moitié donc il n'y en aura pas dans l'autre moitié.

$$T(N)= [\sqrt{N}] +1+ \sum_{2}^{[\sqrt{N}]} 1+1=[\sqrt{N}] +1+([\sqrt{N}]- 2+1)+1=2[\sqrt{N}]+1 \sim O(\sqrt{N})$$

Implémentation : **En langage C**

```c
int Algorithme3(int N)
{
     int i=2;
     float r = sqrt(N) ;
    while(i<= r )
    {
        if(N%i == 0) return 0;
        i++;
    }
    return 1;
}
```

Exécution :

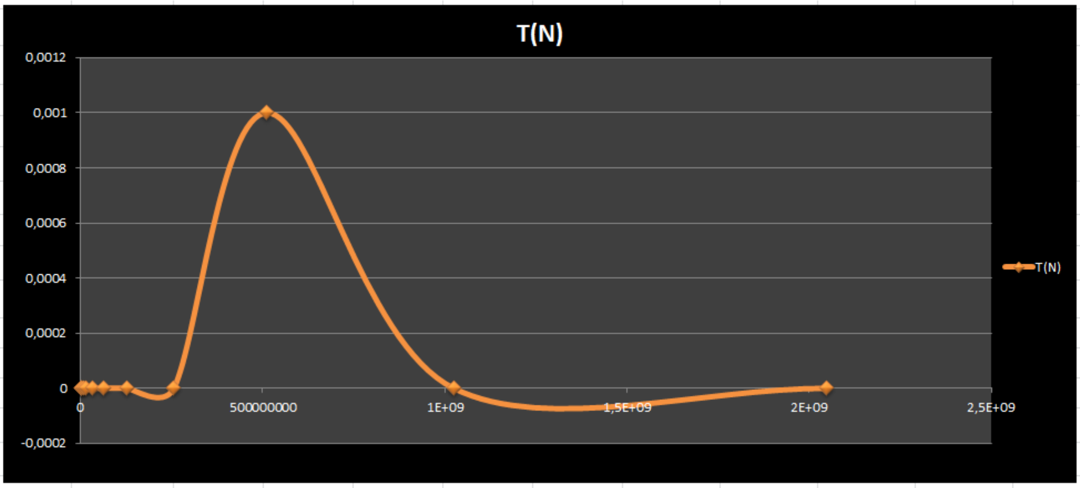
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Algorithme3** premier = Retour de **Algorithme3**).*

```
Execution de l'Algorithme 3 :
N = 1000003.0000001       T= 0.0000001\t ,premier= 1
N = 2000003.0000001       T= 0.0000001\t ,premier= 1
N = 4000037.0000001       T= 0.0000001\t ,premier= 1
N = 8000009.0000001       T= 0.0000001\t ,premier= 1
N = 16000057.0000001      T= 0.0000001\t ,premier= 1
N = 32000011.0000001      T= 0.0000001\t ,premier= 1
N = 64000031.0000001      T= 0.0000001\t ,premier= 1
N = 128000003.0000001     T= 0.0000001\t ,premier= 1
N = 256000001.0000001     T= 0.0000001\t ,premier= 1
N = 512000009.0000001     T= 0.0010001\t ,premier= 1
N = 1024000009.0000001    T= 0.0000001\t ,premier= 1
N = 2048000011.0000001    T= 0.0000001\t ,premier= 1
```

Représentation Graphique :

Graphe du temps d'exécution en fonction de N.

ALGORITHME 3	
N	T
1000003	0
2000003	0
4000037	0
8000009	0
16000057	0
32000011	0
64000031	0
128000003	0
256000001	0
512000009	0,001
1,024E+09	0
2,048E+09	0



IV. 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 pour les nombres impairs.

Algorithme :

```
FONCTION ALGORITHME4(N:ENTIER) : BOOLÉEN
    I,DIVISEURS : ENTIER;
DEBUT
    SI (N MOD 2 = 0) ALORS RETOURNER (FAUX);FIN SI;
        I := 3; R := SQRT(N);
        TANT QUE (I<=R) FAIRE
            SI ( N MOD I = 0 ) ALORS
                RETOURNER (FAUX);
            FIN SI;
            I := I+2;
        FIN TANT QUE;
    RETOURNER (VRAI);
FIN;
```

$1+\sqrt{N}$

$\frac{1}{2}\sum_{I=2}^{[\sqrt{N}]} 1$

1

Complexité :

Au pire des cas : Le nombre ***N*** est impair et premier ainsi la boucle s'itérera jusqu'à ***i=[$\sqrt{N}$]*** ([$\sqrt{N}$]= partie entière de $\sqrt{N}$) car le test ne trouvera aucun nombre ***i*** $\in \{ x , x \in \mathbb{N} \ / \ \boldsymbol{2} \leq x \leq \sqrt{N} \}$ qui soit diviseur de ***N*** dans la première moitié des nombre impairs donc il n'y en aura pas dans l'autre moitié.

$$T(N)=1+\frac{1}{2}\sum_{I=2}^{[\sqrt{N}]} 1 +1 = 1+\frac{1}{2}([\sqrt{N}]- 2 + 1) +1 = \frac{1}{2}[\sqrt{N}]+\frac{3}{2} \sim O(\sqrt{N})$$

Implémentation : **En langage C**

```c
int Algorithme4(int N)
{
    int i;
    if(N%2 == 0)  return 0;
    i=3;
    while(i<= sqrt(N))
    {
        if(N%i == 0) return 0;
        i=i+2;
    }
    return 1;
}
```

Exécution :

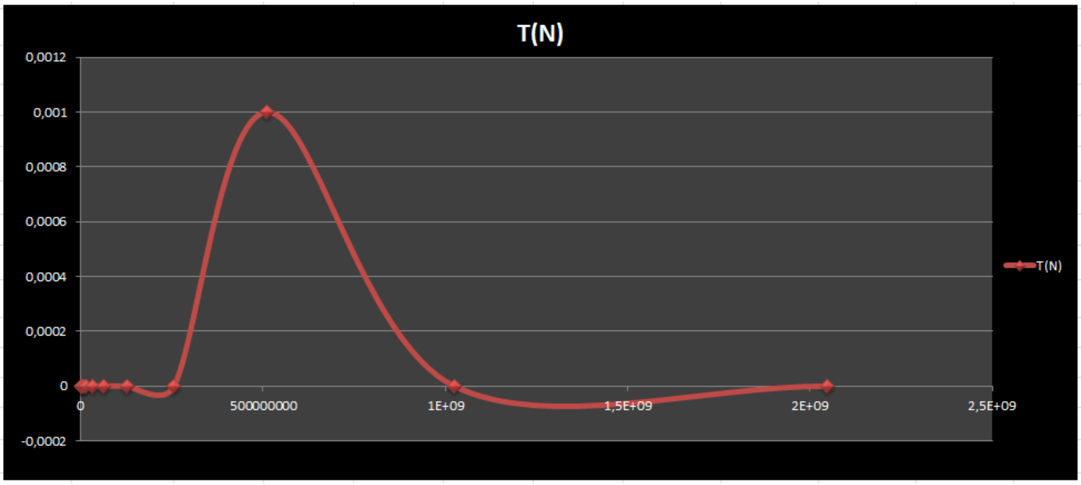
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Algorithme4** premier = Retour de **Algorithme4**).*

```
Execution de l'Algorithme 4 :
N = 1000003.0000001      T= 0.0000001\t ,premier= 1
N = 2000003.0000001      T= 0.0000001\t ,premier= 1
N = 4000037.0000001      T= 0.0000001\t ,premier= 1
N = 8000009.0000001      T= 0.0000001\t ,premier= 1
N = 16000057.0000001     T= 0.0000001\t ,premier= 1
N = 32000011.0000001     T= 0.0000001\t ,premier= 1
N = 64000031.0000001     T= 0.0000001\t ,premier= 1
N = 128000003.0000001    T= 0.0000001\t ,premier= 1
N = 256000001.0000001    T= 0.0000001\t ,premier= 1
N = 512000009.0000001    T= 0.0010001\t ,premier= 1
N = 1024000009.0000001   T= 0.0000001\t ,premier= 1
N = 2048000011.0000001   T= 0.0000001\t ,premier= 1
```

Représentation Graphique :

Graphe du temps d'exécution en fonction de N.

ALGORITHME 3	
N	T
1000003	0
2000003	0
4000037	0
8000009	0
16000057	0
32000011	0
64000031	0
128000003	0
256000001	0
512000009	0,001
1,024E+09	0
2,048E+09	0



(*)Code Source du Programme complet :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int Algorithme1(int N)
{
    int i;
    for(i=2 ; i<= N-1 ; i++)
    {
        if(N%i == 0) return 0;
    }
    return 1;
}


int Algorithme2(int N)
{
    int i;
```

```c
    for(i=2 ; i<= N/2 ; i++)
    {
        if(N%i == 0) return 0;
    }
    return 1;
}


int Algorithme3(int N)
{
    int i=2;
    while(i<= sqrt(N))
    {
        if(N%i == 0) return 0;
       i++;
    }
    return 1;
}


int Algorithme4(int N)
{
    int i;
    if(N%2 == 0)  return 0;
    i=3;
    while(i<= sqrt(N))
    {
        if(N%i == 0) return 0;
        i=i+2;
    }
    return 1;
}



double **Calcul_des_Temps(double **tab , int algorithme)
{
        int j,premier;
        for(j=0 ; j<12 ; j++)
        {
            clock_t begin = clock();
            switch(algorithme)
            {
                case 1: premier = Algorithme1(tab[0][j]); break;
                case 2: premier = Algorithme2(tab[0][j]); break;
                case 3: premier = Algorithme3(tab[0][j]); break;
                case 4: premier = Algorithme4(tab[0][j]); break;
            }
            clock_t end = clock();
            tab[1][j] = (double)(end - begin) / CLOCKS_PER_SEC;
            tab[2][j] = premier;
        }
        return tab;
```

```
}

double **Tableau_de_Valeurs(void)
{
    int i ;
    double **tab;
    tab = (double **)malloc(3*sizeof(double *));
    for(i=0 ; i<3 ; i++) tab[i] = (double *)malloc(12*sizeof(double));
    tab[0][0]=1000003;
    tab[0][1]=2000003;
    tab[0][2]=4000037;
    tab[0][3]=8000009;
    tab[0][4]=16000057;
    tab[0][5]=32000011;
    tab[0][6]=64000031;
    tab[0][7]=128000003;
    tab[0][8]=256000001;
    tab[0][9]=512000009;
    tab[0][10]=1024000009;
    tab[0][11]=2048000011;
    for(i=0 ; i<12 ; i++)tab[1][i] = 0 ;
    return tab;
}

void Afficher_Tableau_de_Valeurs(double **tab)
{
    int j;
        for(j=0 ; j<12 ; j++)
        {
            printf("N = %fl \t T= %fl \t ,premier= %d
\n",tab[0][j],tab[1][j],(int)tab[2][j]);
        }
}
int main(int argc, char *argv[])
{
    printf("Execution de l'Algorithme 1 :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs(),1));

    printf("Execution de l'Algorithme 2 :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs(),2));

    printf("Execution de l'Algorithme 3 :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs(),3));

    printf("Execution de l'Algorithme 4 :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs(),4));


    getchar();
    return 0;
}
```