|  |  |
| --- | --- |
|  | **COMPLEXITÉ** |
|  | **Master 1 IL**  **Groupe 2**  **2018** |

|  |
| --- |
| **[ Algorithmique avancÉe et complexitÉ ]** |
| E-mail : geronimotoutcourt@gmail.com |

Rapport de TP N°2COMPLEXITÉ : TEST DE PRIMALITÉ

**BOUDOUR Mehdi / 201500008386/ TP: Test de Primalité**



*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;**

1

**RETOURNER (VRAI);**

**FIN;**

# 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)= +1 = (N-1 - 2 + 1) +1 = N-1 ∼ 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;**

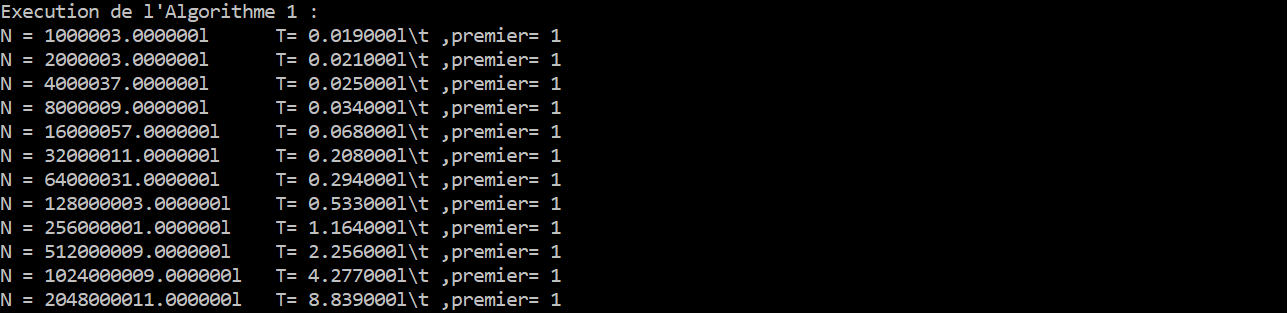
**}**

**return 1;**

**}**

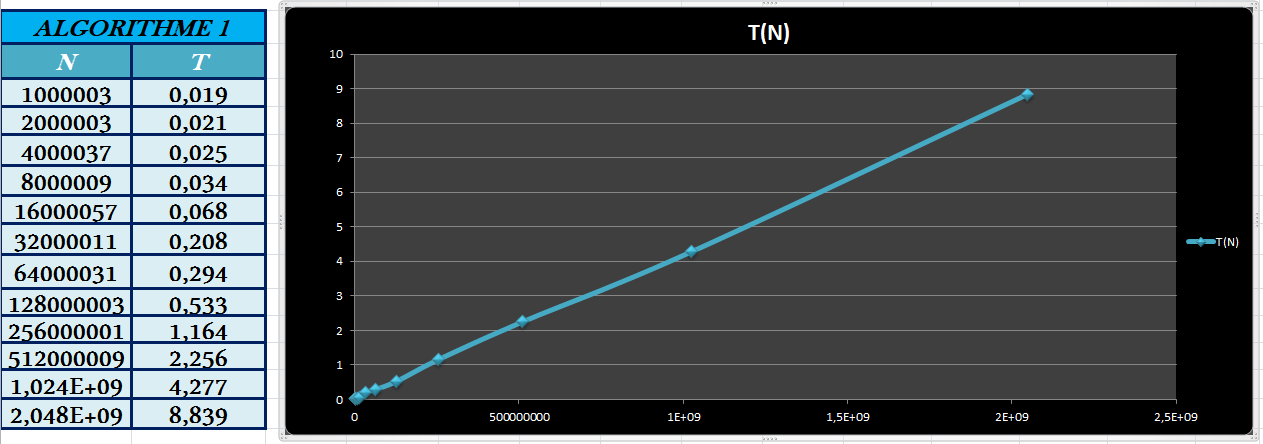
# Exécution :

# *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).*

****

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*

****

# 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;**

1

**RETOURNER (VRAI);**

**FIN;**

# 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)= +1 = ([N/2] - 2 + 1) +1 = [N/2] ∼ O(N)

# Implémentation : En langage C

**int Algorithme2(int N)**

**{**

**int i;**

**for(i=2 ; i<= N/2 ; i++)**

**{**

**if(N%i == 0) return 0;**

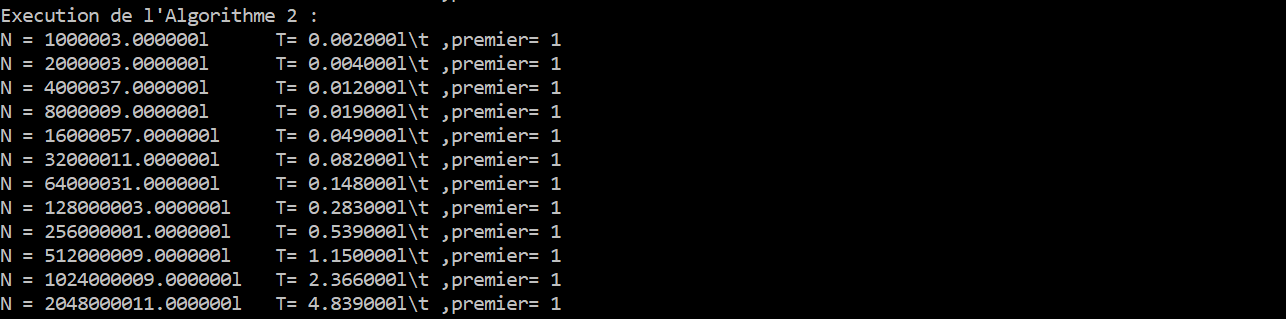
**}**

**return 1;**

**}**

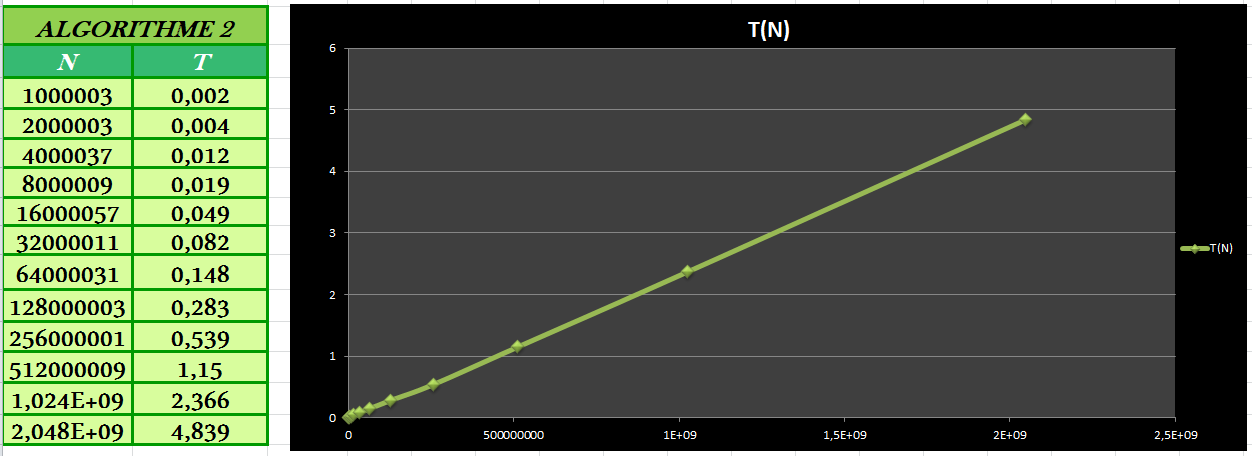
# Exécution :

# *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).*

****

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*



# III. Algorithme 3(A3) :

Utilisons la propriété selon laquelle la moitié des diviseurs d’un nombre ≤ et l’autre ≥ .

# Algorithme :

**FONCTION ALGORITHME3(N:ENTIER) : BOOLÉEN**

**I: ENTIER;**

**R: RÉEL;**

**DEBUT**

**R := SQRT(N);**

1

**I :=2 ;**

**TANT QUE (I <= R) FAIRE**

**SI ( N MOD I = 0 ) ALORS**

**RETOURNER (FAUX);**

**FIN SI;**

**I := I+1 ;**

**FIN TANTQUE;**

1

**RETOURNER (VRAI);**

**FIN;**

# Complexité :

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

T(N)= +1+ +1= +1+(- 2+1)+1=2+1 ∼ O()

# Implémentation : En langage C

**int Algorithme3(int N)**

**{**

**int i=2;**

**float r = sqrt(N) ;**

**while(i<= r )**

**{**

**if(N%i == 0) return 0;**

**i++;**

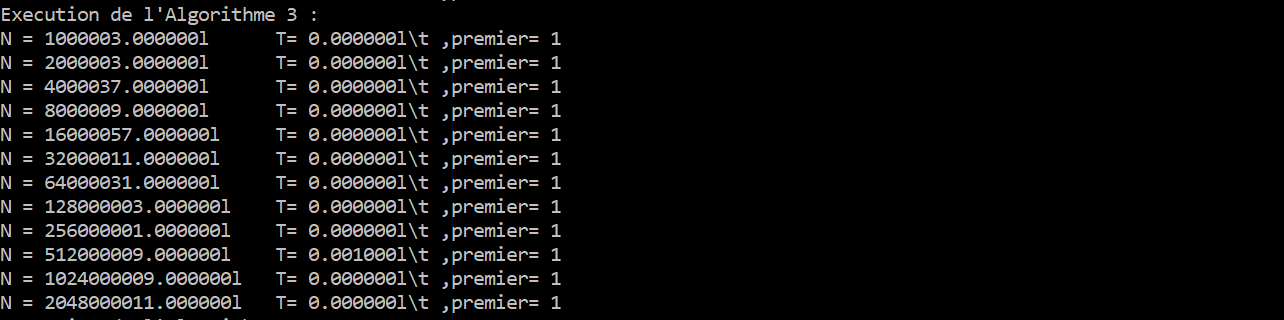
**}**

**return 1;**

**}**

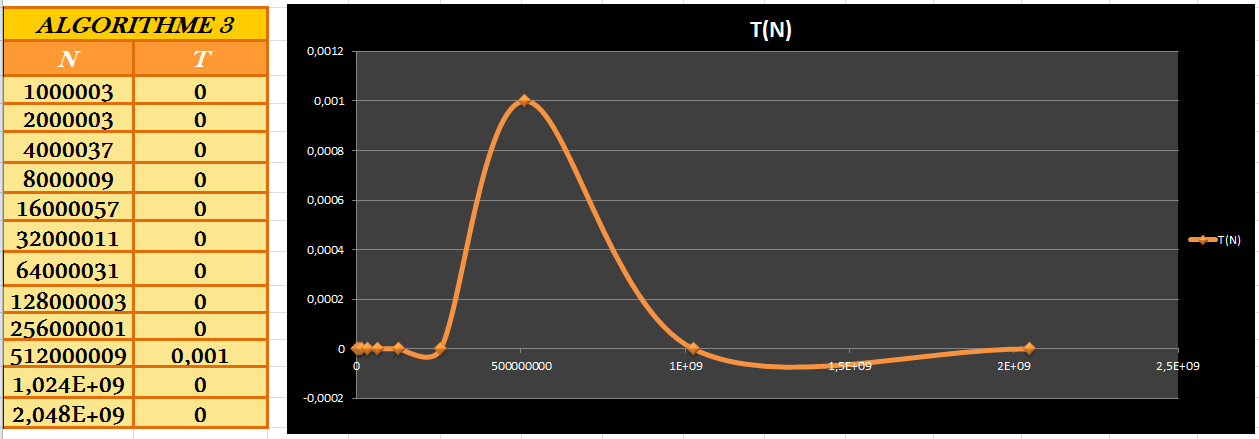
# Exécution :

# *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).*

****

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*

****

# 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;**

1 +

**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;**

1 (pire cas)

**RETOURNER (VRAI);**

**FIN;**

# Complexité :

**Au pire des cas :** Le nombre ***N*** est impair et premier ainsi la boucle s’itérera jusqu’à ***i=[]***  ([]= partie entière de ) car le test ne trouvera aucun nombre ***i*** ∈ { 𝑥 , 𝑥∈ ℕ / ***2*** ≤ 𝑥 ≤ } 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+ +1 = 1+ (- 2 + 1) +1 = + ∼ O()

# Implémentation : En langage 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;**

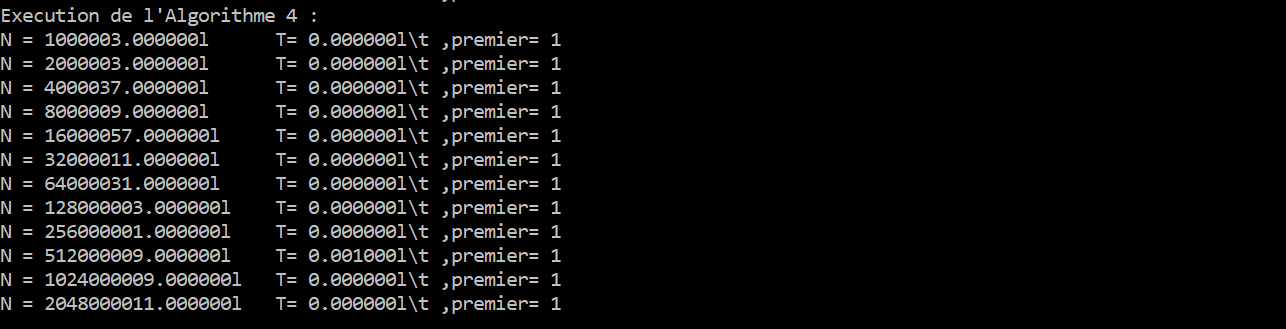
**}**

**return 1;**

**}**

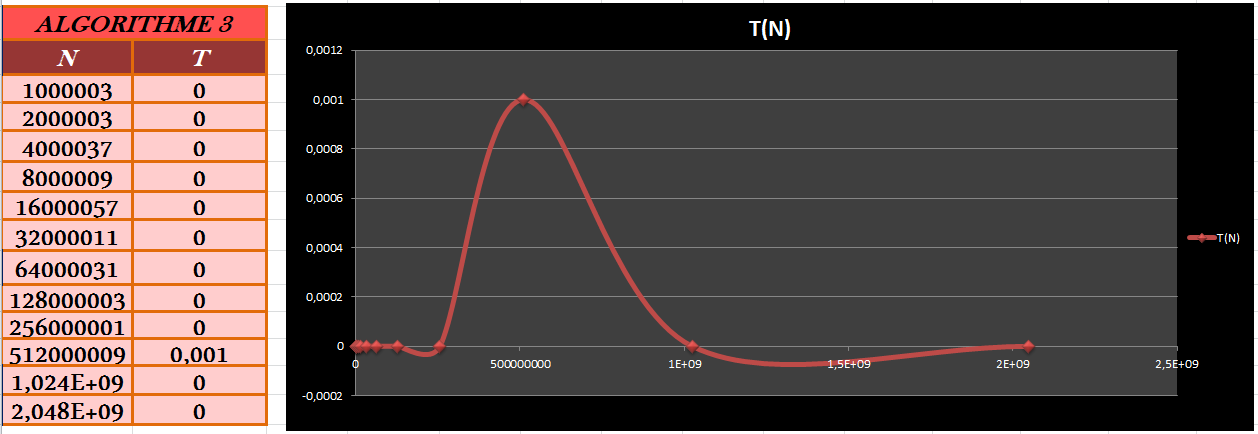
# Exécution :

# *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).*

****

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*

****

# (\*)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;**

**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;**

**}**