COMPLEXITÉ

Master 1 IL
Groupe 2
2018

Rapport de TP N°5 COMPLEXITÉ : Algorithmes itératifs vs récursifs

BOUDOUR Mehdi / 201500008386/ TP: Algorithmes itératifs vs récursifs

[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 *Suite de Fibonacci*:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-2} + F_{n-1} \quad \text{pour } n \geq 2.$$

Ecrire une fonction Fibo_Rec qui calcul récursivement le nième terme de la suite de Fibonacci.

Algorithme : *Réccursif*

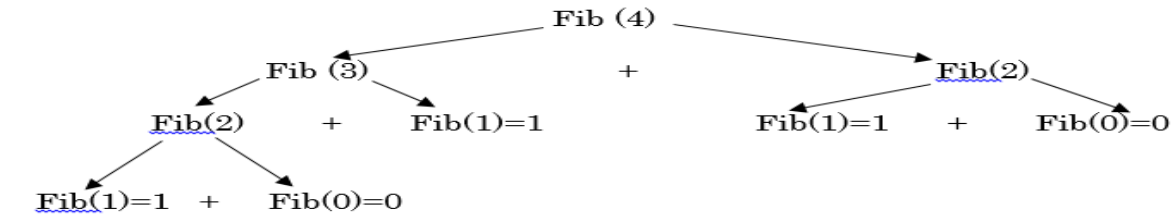
```
FONCTION FIBO_REC(E/ N:ENTIER) :ENTIER
DEBUT
        SI F(N=0)ALORS
             RETOURNER 0;
        SINON
             SI (N=1) ALORS
                  RETOURNER 1;
             SINON
                  SI(N>=2) ALORS
                       RETOURNER FIBO_REC(N-1)+FIBO_REC(N-2);
                  FIN SI;
             FIN SI;

        FIN SI;
FIN;
```

Complexité :

Arbre récursif :

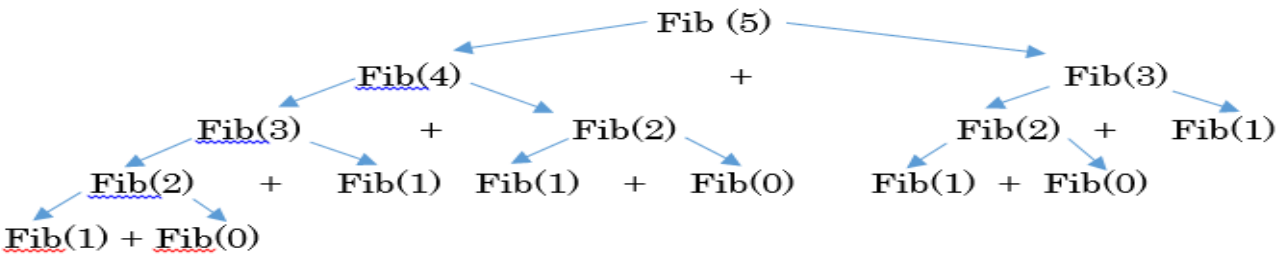
F(4)= ?

L'exécution de cette fonction récursive pour n=4 nous donne l'arbre suivant :



Fib(4)=3 (9 appels pour arriver au résultat)

F(5) = ?



Pour n=5 Fib(5)=5 (15 appels pour arriver au résultat)

$$T(N) \sim O(2^n) \text{ Exponentielle}$$

Implémentation : En langage C

```
long Fibo_Rec(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
        else
            if(n>=2)
```

```
                return Fibo_Rec(n-1)+Fibo_Rec(n-2);
}
```

Exécution :

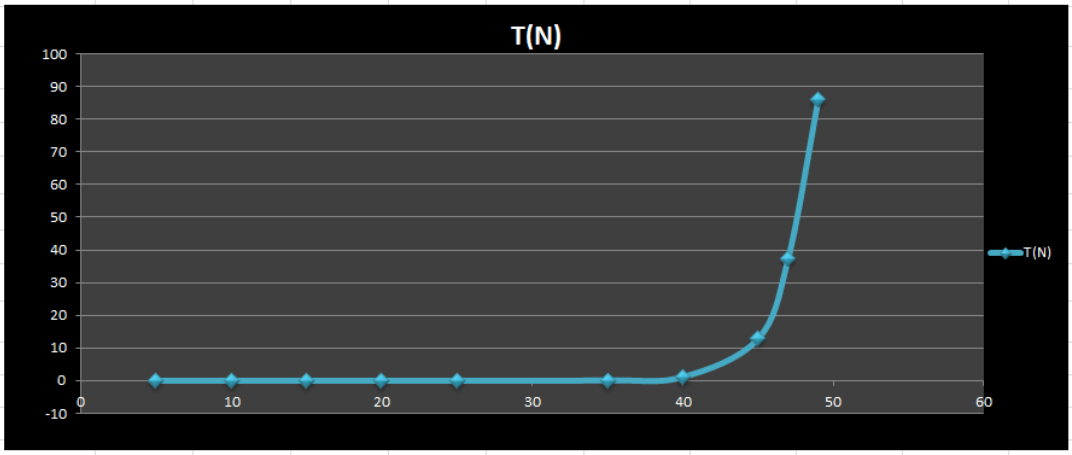
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Fibo_Rec**).*

```
Execution de Fibo_Rec :
N = 5.000000     T= 0.000000
N = 10.000000    T= 0.000000
N = 15.000000    T= 0.000000
N = 20.000000    T= 0.000000
N = 25.000000    T= 0.001000
N = 35.000000    T= 0.099000
N = 40.000000    T= 1.108000
N = 45.000000    T= 12.819000
N = 47.000000    T= 37.381000
N = 49.000000    T= 85.971000
```

Représentation Graphique :

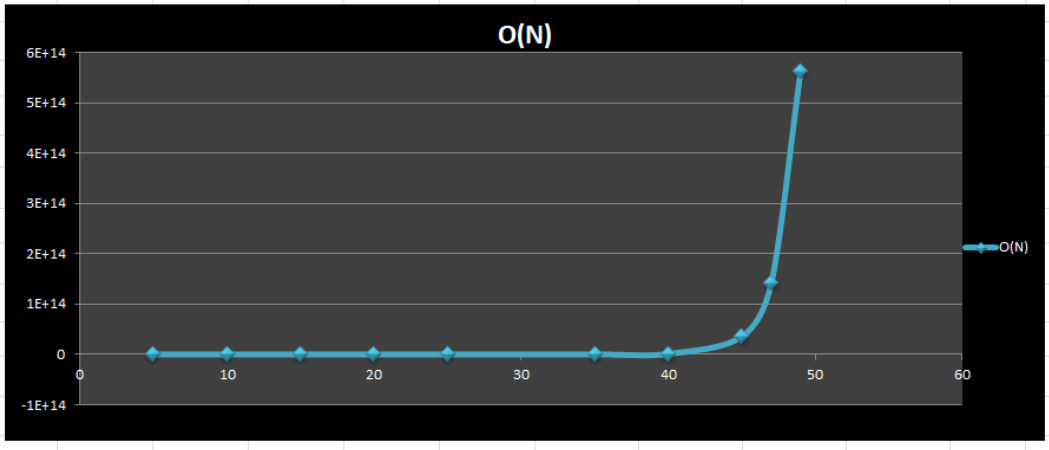
Graphe du temps d'exécution en fonction de N.

Fibo_Rec	
N	*T(N)*
5	0
10	0
15	0
20	0
25	0,001
35	0,099
40	1,108
45	12,819
47	37,381
49	85,971



Graphe de la complexité théorique en fonction de N.

Fibo_Rec	
N	*O(N)*
5	32
10	1024
15	32768
20	1048576
25	33554432
35	3,44E+10
40	1,1E+12
45	3,52E+13
47	1,41E+14
49	5,63E+14



Algorithme : *Itératif*

```
FONCTION FIBO_ITER(E/ N:ENTIER):ENTIER
    F1,F2,F:ENTIER;
DEBUT
```

```
        SI (N=0) ALORS
            RETOURNER 0;
        SINON
            SI(N=1) ALORS
                RETOURNER 1;
            FIN SI;
        FIN SI;
        F1=0,F2=1;
        TANT QUE (N>=2) FAIRE
            F  = F1 + F2;
            F1 = F2;
            F2 = F;
            N=N-1;
        FIN TANT QUE;
        RETOURNER F;
FIN;
```

1

2

$\sum_2^N 4$

1

Complexité :

$$T(N)= \sum_2^N 4 +1+2+1 = 4(N-2 + 1) +4 = 4N \sim O(N) \text{ Linéaire}$$

Implémentation : En langage

```
long Fibo_iter(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
    long F1=0,F2=1,F;
    for(; n>=2 ; n--)
    {
        F  = F1 + F2;
        F1 = F2;
        F2 = F;
    }
    return F;
}
```

Exécution :

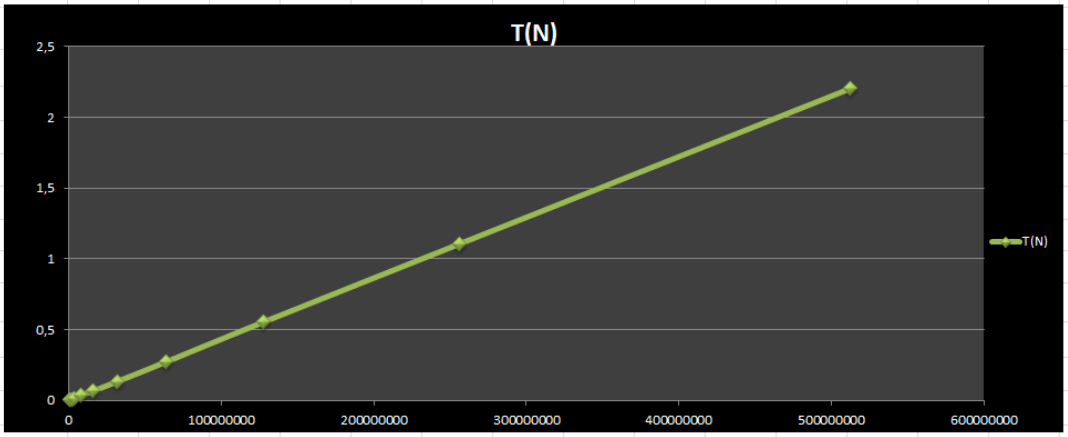
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Fibo_iter**).*

```
Execution de Fibo_iter :
N = 1000003.000000      T= 0.004000
N = 2000003.000000      T= 0.008000
N = 4000037.000000      T= 0.017000
N = 8000009.000000      T= 0.035000
N = 16000057.000000     T= 0.068000
N = 32000011.000000     T= 0.135000
N = 64000031.000000     T= 0.274000
N = 128000003.000000    T= 0.556000
N = 256000001.000000    T= 1.106000
N = 512000009.000000    T= 2.203000
```

Représentation Graphique :

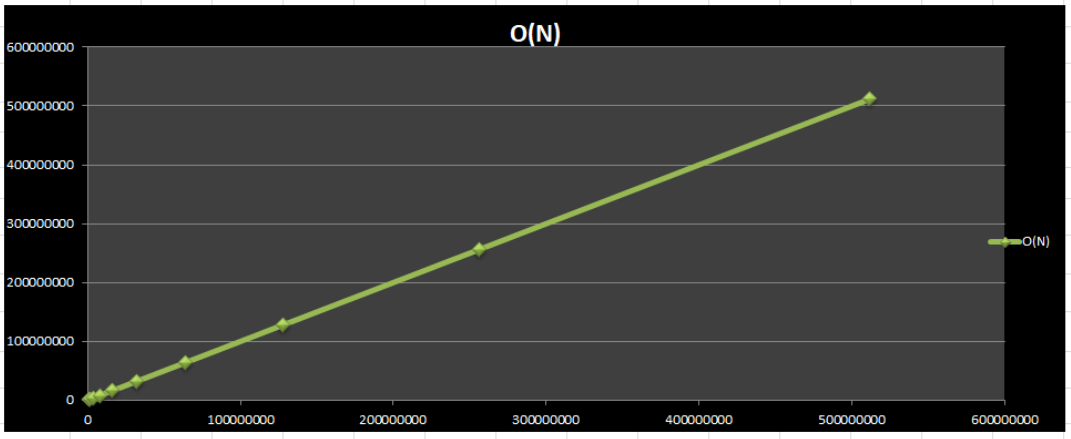
Graphe du temps d'exécution en fonction de N.

Fibo_iter	
N	*T(N)*
1000003	0,004
2000003	0,008
4000037	0,017
8000009	0,035
16000057	0,068
32000011	0,135
64000031	0,274
128000003	0,556
256000001	1,106
512000009	2,203



Graphe de la complexité théorique en fonction de N.

Fibo_iter	
N	*O(N)*
1000003	1000003
2000003	2000003
4000037	4000037
8000009	8000009
16000057	16000057
32000011	32000011
64000031	64000031
128000003	128000003
256000001	256000001
512000009	512000009



I. Algorithme *Phi*:

On peut monter que la suite (Un), $n \in$ N vérifie $Un \sim \phi$, où ϕ est le nombre d'or. Ainsi lim Un /Un-1 = ϕ quand $n \to +\infty$. Faites une copie de la fonction Fibo_iter et modifiez-la pour tester cette propriété (utilisez le type double). Qui calcul ϕ.

Remarque : $\phi = 1+\sqrt{5} \cong 1{,}618033988749894848204586834365 6$

Algorithme :

```
FONCTION PHI(E/ N:ENTIER : REEL
       F1,F2,F:REEL
DEBUT
         SI (N=0) ALORS
             RETOURNER 0;
         SINON
             SI (N=1) ALORS
                 RETOURNER 1;
             FIN SI;
         FIN SI;
         F1=0,F2=1;
         TANT QUE (N>=2) FAIRE
             F  = F1 + F2;
             F1 = F2;
             F2 = F;
             N=N-1;
         FIN TANT QUE;
         RETOURNER  F2/F1;
FIN;
```

1

2

$\sum_2^N 4$

1

Complexité :

$$T(N)= \sum_2^N 4 +1+2+1 = 4(N-2 + 1) +4 = 4N \sim O(N) \text{ Linéaire}$$

Implémentation : En langage C

```c
double Phi(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
    double F,F1=0,F2=1;
    for(; n>=2 ; n--)
    {
        F  = F1 + F2;
        F1 = F2;
        F2 = F;
    }
    return (double) F2/F1;
}
```

Exécution :

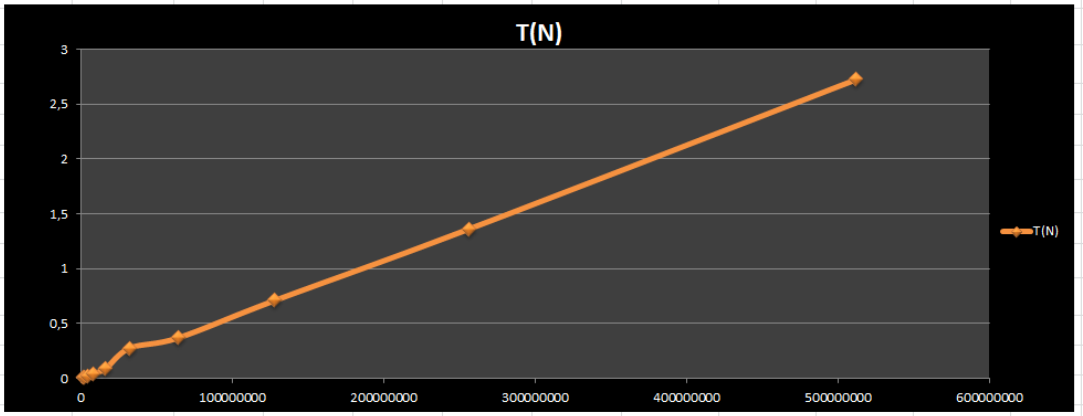
*Affichage du temps d'exécution de l'algorithme pour chaque valeur de N (T = le temps d'exécution calculé pour chaque exécution de la fonction **Phi**).*

```
Execution de Phi:
N = 1000003.000000      T= 0.005000
N = 2000003.000000      T= 0.011000
N = 4000037.000000      T= 0.022000
N = 8000009.000000      T= 0.043000
N = 16000057.000000     T= 0.088000
N = 32000011.000000     T= 0.275000
N = 64000031.000000     T= 0.368000
N = 128000003.000000    T= 0.710000
N = 256000001.000000    T= 1.360000
N = 512000009.000000    T= 2.724000
```

Représentation Graphique :

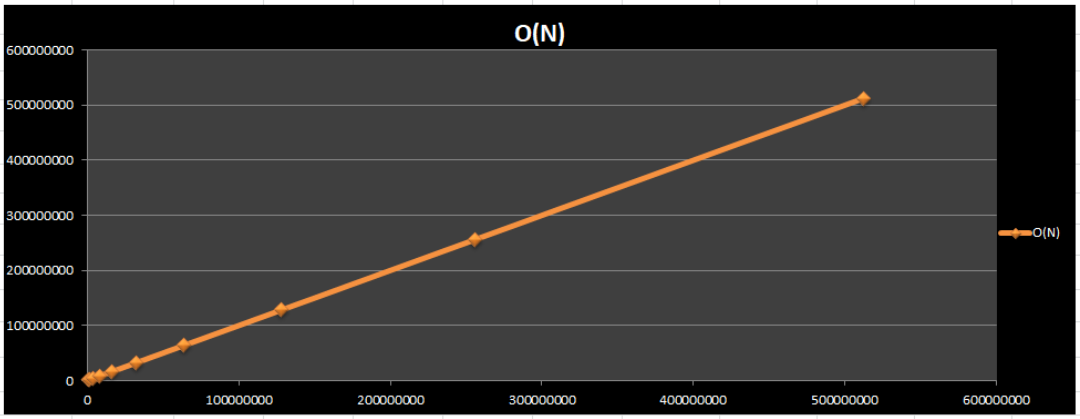
Graphe du temps d'exécution en fonction de N.

Phi	
N	*T(N)*
1000003	0,005
2000003	0,011
4000037	0,022
8000009	0,043
16000057	0,088
32000011	0,275
64000031	0,368
128000003	0,71
256000001	1,36
512000009	2,724



Graphe de la complexité théorique en fonction de N.

Phi	
N	*O(N)*
1000003	1000003
2000003	2000003
4000037	4000037
8000009	8000009
16000057	16000057
32000011	32000011
64000031	64000031
128000003	128000003
256000001	256000001
512000009	512000009



(*)Code Source du Programme complet :

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

```c
#include <math.h>

long Fibo_Rec(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
        else
            if(n>=2)
                return Fibo_Rec(n-1)+Fibo_Rec(n-2);
}

long Fibo_iter(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
    long F1=0,F2=1,F;
    for(; n>=2 ; n--)
    {
        F  = F1 + F2;
        F1 = F2;
        F2 = F;
    }
    return F;
}

double Phi(long n)
{
    if(n==0) return 0;
    else
        if(n==1) return 1;
    double F,F1=0,F2=1;
    for(; n>=2 ; n--)
    {
        F  = F1 + F2;
        F1 = F2;
        F2 = F;
    }
    return (double) F2/F1;
}

double **Calcul_des_Temps(double **tab , long algorithme)
{
```

```
        long j;
        double resultat;
        for(j=0 ; j<11 ; j++)
        {
            clock_t begin = clock();
            switch(algorithme)
            {
                case 1: resultat = Fibo_Rec(tab[0][j]); break;
                case 2: resultat = Fibo_iter(tab[0][j]); break;
                case 3: resultat = Phi(tab[0][j]); break;
            }
            clock_t end = clock();
            tab[1][j] = (double)(end - begin) / CLOCKS_PER_SEC;
            tab[2][j] = resultat;
        }
        return tab;
}


double **Tableau_de_ValeursFibRec(void)
{
    long i ;
    double **tab;
    tab = (double **)malloc(3*sizeof(double *));
    for(i=0 ; i<3 ; i++) tab[i] = (double
*)malloc(11*sizeof(double));
    tab[0][0]=5;
    tab[0][1]=10;
    tab[0][2]=15;
    tab[0][3]=20;
    tab[0][4]=25;
    tab[0][5]=35;
    tab[0][6]=40;
    tab[0][7]=45;
    tab[0][8]=47;
    tab[0][9]=49;
    tab[0][10]=50;
    for(i=0 ; i<11 ; i++)tab[1][i] = 0 ;
    return tab;
}

double **Tableau_de_ValeursFibIter(void)
{
    long i ;
```

```
    double **tab;
    tab = (double **)malloc(3*sizeof(double *));
    for(i=0 ; i<3 ; i++) tab[i] = (double
*)malloc(11*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;

    for(i=0 ; i<10 ; i++)tab[1][i] = 0 ;
    return tab;
}

void Afficher_Tableau_de_Valeurs(double **tab)
{
    long j;
        for(j=0 ; j<10 ; j++)
        {
            printf("N = %lf \t T= %lf \t ,resultat = %lf
\n",tab[0][j],tab[1][j],tab[2][j]);
        }
}

int main(int argc, char *argv[])
{

    printf("Execution de Fibo_iter :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
FibIter(),2));

    printf("Execution de Phi:\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
FibIter(),3));

    printf("Execution de Fibo_Rec :\n");
    Afficher_Tableau_de_Valeurs(Calcul_des_Temps(Tableau_de_Valeurs
FibRec(),1));
```

```
    getchar();
    return 0;
}
```