Université des Sciences et de la Technologie Houari Boumediene

Conception et Complexité des Algorithme

Rapport de Travaux Pratiques N°1
Mesure du temps d'exécution d'un programme.

Quadrinôme
MOHAMMEDI Haroune
HOUACINE Naila Aziza

Professeur
Pr. AMANI Ferhat

19 octobre 2017

1 Partie I : Développement de l'algorithme et du programme pour le problème du calcul de la somme des n premiers nombres entiers naturels.

1 Développement de l'algorithme itératif qui permet de calculer la somme S des n premiers nombres entiers naturels.

L'écriture de ces algorithme sera accompagné de commentaire représentant le nombre de mot mémoire que prend chaque instruction puis celui de tous l'algorithme.

1.1 En utilisant la forme "pour ... faire" :

L'algorithme développé ci dessous, nommé Algorithme_Somme_Iteratif_pour, utilise la forme de répétition : Pour ... faire ... fait.

```
Algorithme_Somme_Iteratif_pour

VAR
i,N,S : entier;                 //3 mots mémoire

Debut
   ecrire("Donner N = ");       //1 mot mémoire
   lire(N);                     //1 mot mémoire

   S = 0;                       //1 mot mémoire

   pour(i=1 ; i <= N ; i++)     //4 mots mémoire
     faire
       S = S + i;               //2 mots mémoire
     fait;

   ecrire("La somme = ",S);     //1 mot mémoire

Fin.
```

Totale de mots mémoire = 13 MM

1.2 En utilisant la forme "tant que ... faire" :

L algorithme développé ci dessous, nommé Algorithme_Somme_Iteratif_tant_que, utilise la forme de répétition : tant que ... faire ... fait.

```
Algorithme_Somme_Iteratif_tant_que

VAR
i,N,S : entier;                 //3 mots mémoire

Debut
   ecrire("Donner N = ");       //1 mot mémoire
   lire(N);                     //1 mot mémoire

   S = 0;                       //1 mot mémoire
   i = 1;                       //1 mot mémoire

   tant que(i <= N)             //1 mot mémoire
     faire
       S = S + i;               //2 mots mémoire
       i = i + 1;               //2 mots mémoire
     fait;
```

```
    ecrire("La somme = ",S);     //1 mot mémoire

Fin.
```

Totale de mots mémoire = 13 MM

1.3 En utilisant la forme "répéter ... jusqu'à" :

L algorithme développé ci dessous, nommé Algorithme_Somme_Iteratif_répéter_jusqu_a, utilise la forme de répétition : répéter ... jusqu'à ...

```
Algorithme_Somme_Iteratif_répéter_jusqu_a

VAR
i,N,S : entier;                 //3 mots mémoire

Debut
    ecrire("Donner N = ");      //1 mot mémoire
    lire(N);                    //1 mot mémoire

    S = 0;                      //1 mot mémoire
    i = 1;                      //1 mot mémoire

    répéter
        S = S + i;              //2 mots mémoire
        i = i + 1;              //2 mots mémoire
      jusqu a(i > N);           //1 mot mémoire

    ecrire("La somme = ",S);    //1 mot mémoire

Fin.
```

Totale de mots mémoire = 13 MM

2 Développement des programmes itératifs en langage C.

2.1 En utilisant la forme "FOR" :

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

int main()
{
    long int i,N,S;

    printf("Donner N = ");
    scanf("%Ld",&N);

    S=0;

    for(i=1 ; i <= N ; i++)
    {
        S = S + i;
    }

    printf("La somme S = %Ld",S);
    return 0;
}
```

2.2 En utilisant la forme "WHILE" :

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

int main()
{
    long int i,N,S;

    printf("Donner N = ");
    scanf("%Ld",&N);

    S=0; i=1;

    while(i <= N)
    {
        S = S + i;
        i = i + 1;
    }

    printf("La somme S = %Ld",S);
    return 0;
}
```

2.3 En utilisant la forme "DO ... WHILE" :

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

int main()
{
    long int i,N,S;

    printf("Donner N = ");
    scanf("%Ld",&N);

    S=0; i=1;

    Do
    {
        S = S + i;
        i = i + 1;
    }while(i > N)

    printf("La somme S = %Ld",S);
    return 0;
}
```

3 Développement de l'algorithme récursif pour le problème.

```
Procedure_Somme_Récursif(N , S : entier)

 VAR

 Debut
```

```
    Si( N <= 0)                                          //1 mot mémoire
        Alors ecrire("La Somme S = ",S);                 //1 mot mémoire
        Sinon   S = S + N;                               //2 mots mémoire
                Algorithme_Somme_Récursif(N - 1,S); //2 mots mémoire
    FinSi;

 Fin.


 Algorithme_Somme_Récursif()

 VAR

 Debut
    N: entier;                                           //1 mot mémoire

    ecrire("Donner N = ");                               //1 mot mémoire
    lire(N);                                             //1 mot mémoire

    Procedure_Somme_Récursif(N , 0);                     //1 mot mémoire
 Fin.
```

Totale de mots mémoire = 13 MM

4 Développement du programme récursif en langage C.

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

void SommeRecursive(long int N,long int S)
{
    if(N <= 0)
    {
        printf("La somme S = %Ld \n",S);
    }else
    {
        S = S + N;
        SommeRecursive(N - 1,S);
    }
}


int main()
{
    long int N;

    printf("Donner N = ");
    scanf("%Ld",&N);

    SommeRecursive(N,0);

    return 0;
}
```

2 Partie II : Mesure du temps d'exécution.

Pour la mesure du temps en langage C nous aurons besoin des fonctions de la gestion du temps que l'on retrouve dans la bibliothèque time.h.

Donc nous devant d'abord inclure la directive #include <time.h> Puis définir deux(2) variables notées début et fin de type clock_t.

A l'aide de la fonction "clock()" : Récupéré le Temps avant l'exécution du programme dans début et récupéré le Temps après la fin de l'exécution du programme dans fin. Calculer la différence entre début et fin dans un variable nommé : Delta de type double. Afin que le temps d'exécution Delta soit donné en secondes, nous divisons ce dernier par le paramètre CLOCKS_PER_SEC.

5 Mesure des temps d'exécution et Représentation par un graphe

Au lieu de mesurer séparément le temps d'exécution pour chaque valeur de n donnée, on automatise le processus des mesures en utilisant un tableau tab contenant toutes les valeurs à tester et affichant en boucle le résultat pour chaque valeur.

5.1 L'algorithme itératif avec boucle FOR

le programme

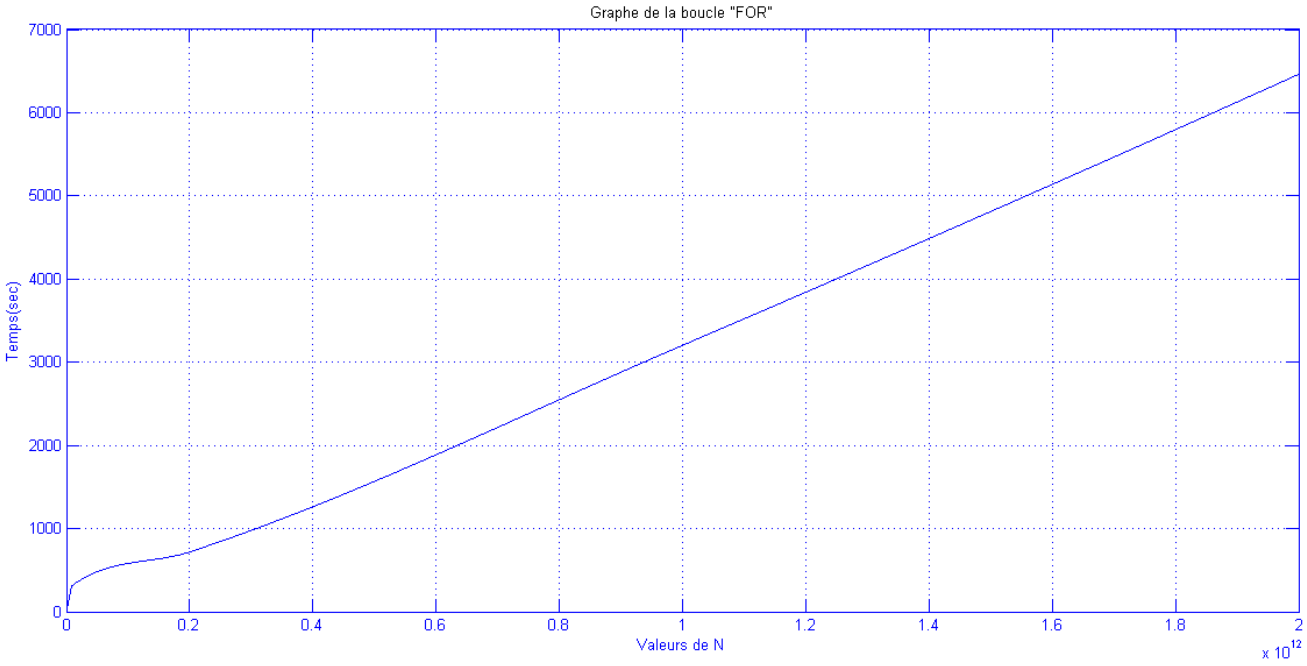
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

int main()
{
   double i,j,N,S;
   clock_t start_t, end_t;
   double total_t;

   double tab[14]={pow(10,6),2*pow(10,6),pow(10,7),2*pow(10,7), pow(10,8), 2*pow(10,8),
      ↪      pow(10,9), 2*pow(10,9), pow(10,10),2*pow(10,10), pow(10,11), 2*pow(10,11)
      ↪ , pow(10,12), 2*pow(10,12)};

   printf("L'algorithme itératif avec boucle for \n\n");
   for(j=0 ; j < 14 ; j++) {

        start_t = clock();          /*debut*/

        S=0; i=1;

        for(i=1 ; i <= tab[(long int)j]; i++)
             S = S + i;

        end_t = clock();            /*Fin*/

        printf("La somme S = %lf \n",S);
        total_t = (double) (end_t - start_t) / CLOCKS_PER_SEC;
        printf("pour %lf Iteration le programme prends %lf\n\n", tab[(long int)j],
           ↪ total_t);
   }
   return 0;
}
```

le tableau

Valeur N :	10^6	$2*10^6$	10^7	$2*10^7$	10^8	$2*10^8$	10^9
Temps d'exe :	0.011177	0.006160	0.031110	0.061784	0.308448	0.615293	3.076619

Valeur N :	$2*10^9$	10^{10}	$2*10^{10}$	10^{11}	$2*10^{11}$	10^{12}	$2*10^{12}$
Temps d'exe :	6.179196	30.920366	69.197337	314.129090	718.460152	3201.630410	6459.747801

le graphe



5.2 L'algorithme itératif avec boucle while

le programme

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

 int main()
 {
    double i,j,N,S;
    clock_t start_t, end_t;
    double total_t;

    double tab[14]={pow(10,6),2*pow(10,6), pow(10,7),2*pow(10,7), pow(10,8), 2*pow(10,8)
    ↪ ,   pow(10,9), 2*pow(10,9), pow(10,10),2*pow(10,10), pow(10,11), 2*pow(10,11)
    ↪ , pow(10,12), 2*pow(10,12)};

    printf("L'algorithme itératif avec boucle while \n\n");

    for(j=0 ; j < 14 ; j++) {

        start_t = clock();      /*debut*/


        S=0; i=1;

        while(i <= tab[(long int)j])
```

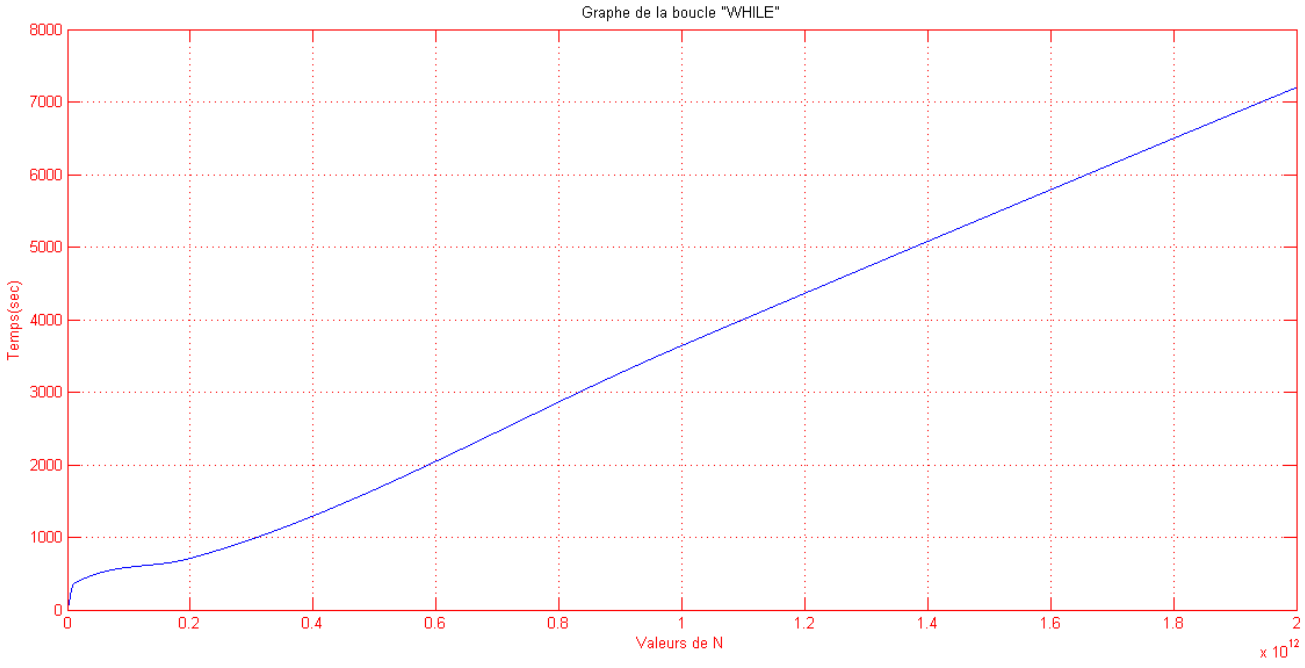
```
        {
            S = S + i;
            i = i + 1;
        }

        end_t = clock();         /*Fin*/

        printf("La somme S = %lf \n",S);
        total_t = (double) (end_t - start_t) / CLOCKS_PER_SEC;
        printf("pour %lf Iteration le programme prends %lf\n\n", tab[(long int)j],
            ↪ total_t);
    }
    return 0;
}
```

le tableau

Valeur N :	10^6	$2*10^6$	10^7	$2*10^7$	10^8	$2*10^8$	10^9
Temps d'exe :	0.003227	0.006472	0.032436	0.064657	0.322734	0.646398	3.228527

Valeur N :	$2*10^9$	10^{10}	$2*10^{10}$	10^{11}	$2*10^{11}$	10^{12}	$2*10^{12}$
Temps d'exe :	6.456017	32.271228	67.204256	353.661353	709.004102	3640.236612	7196.518894

le graphe



5.3 L'algorithme itératif avec boucle Do .. while

le programme

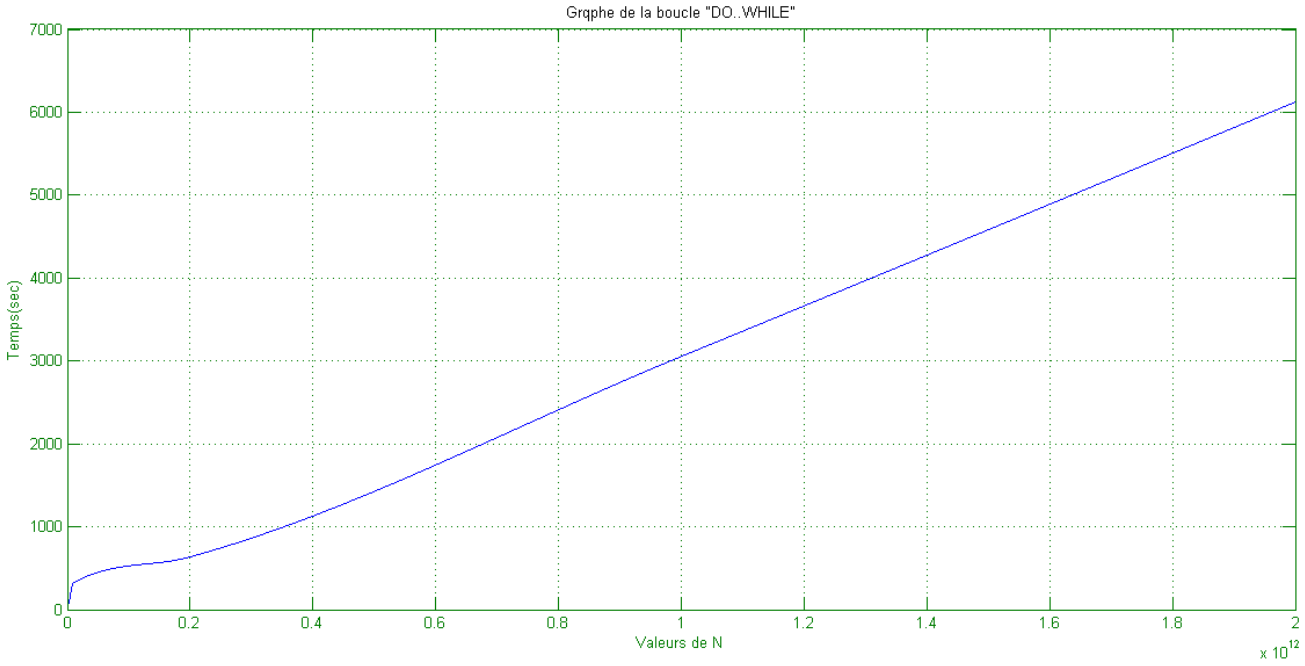
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

 int main()
 {
    double i,j,N,S;
    clock_t start_t, end_t;
    double total_t;

    double tab[14]={pow(10,6),2*pow(10,6),pow(10,7),2*pow(10,7), pow(10,8), 2*pow(10,8),
    ↪  pow(10,9), 2*pow(10,9), pow(10,10),2*pow(10,10), pow(10,11), 2*pow(10,11),
    ↪ pow(10,12), 2*pow(10,12)};

    printf("L'algorithme itératif avec boucle do .. while \n\n");

    for(j=0 ; j < 14 ; j++) {

        start_t = clock();          /*debut*/

        S=0; i=1;

        do
        {
            S = S + i;
            i = i + 1;
        } while(i <= tab[(long int)j]);

        end_t = clock();            /*Fin*/

        printf("La somme S = %lf \n",S);
        total_t = (double) (end_t - start_t) / CLOCKS_PER_SEC;
        printf("pour %lf Iteration le programme prends %lf\n\n", tab[(long int)j],
        ↪ total_t);

    }
    return 0;
}
```

le tableau

Valeur N :	10^6	$2*10^6$	10^7	$2*10^7$	10^8	$2*10^8$	10^9
Temps d'exe :	0.003294	0.006591	0.033344	0.066648	0.341006	0.692939	3.504640

Valeur N :	$2*10^9$	10^{10}	$2*10^{10}$	10^{11}	$2*10^{11}$	10^{12}	$2*10^{12}$
Temps d'exe :	7.179120	35.645556	61.837564	317.164022	635.293616	3052.125339	6121.511236

le graphe



5.4 L'algorithme récursif

le programme

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

void SommeRecursive(long int N, long int S)
{
    if(N > 0)
    {
        S = S + N;
        SommeRecursive(N - 1, S);
    }
    else{
        return;
    }

}


int main()
{

    long int i,j,N,S;
    clock_t start_t, end_t;
    double total_t;

    long int tab[14]={pow(10,6),2*pow(10,6),pow(10,7),2*pow(10,7),pow(10,8),2*pow(10,8),
    pow(10,9),2*pow(10,9),pow(10,10),2*pow(10,10),pow(10,11),2*pow(10,11),pow(10,12),2*
        ↪ pow(10,12)};

    printf("L'algorithme recursif\n");

    for(j=0 ; j < 14 ; j++) {
```

```
        start_t = clock();

        S=0; i=1;

        SommeRecursive(tab[(long int) j],S);

        end_t = clock();

        printf("La somme S = %lf \n",S);
        total_t = (double) (end_t - start_t) / CLOCKS_PER_SEC;
        printf("Pour N = %lf le programme prend %lf\n\n", tab[(long int) j], total_t);
    }


    return 0;
}
```

Remarque : Les valeurs du tableau et le graphe correspondants à ce programme ne sont pas inclus dans ce rapport car les valeurs de N étant trop grandes conduisent à la saturation de la pile responsable de la gestion de la récursivité.

6 Déduction du temps d'exécution moyen

1- On remarque que les valeurs et graphes obtenues avec les 3 algorithmes itératifs sont presque identiques ; donc il suffit d'étudier un seul, nous choisirons le programme utilisant la boucle WHILE.

2- On remarque que les temps d'exécution sont approximativement doublés lorsque N est doublé et qu'ils sont pratiquement multipliés par 10 lorsque N est multiplié par 10.

Exemples :
N $= 10^9 \rightarrow$ T $= 3.228527$
N $= 2 * 10^9 \rightarrow$ T $= 6.456017$

Aussi
N $= 10^9 \rightarrow$ T $= 3.228527$
N $= 10^{10} = 10 * 10^9 \rightarrow$ T $= 32.271228$

3- On peut constater la linéarité du graphe.

On en déduit que le temps d'exécution est proportionnel à N, ce que l'on peut représenter par la formule suivante :

$$T(x * N) = x * T(N) \text{ pour tous } x * N \in [10^6, 2 * 10^{12}]$$

(x étant la tangente d'un point sur le graphe).

Nous ne pouvant pas généraliser car les testes que nous avons fait n'englobent pas toutes les valeurs possibles, la solution est le calcule de la complexité temporelle du programme.