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

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

Rapport de TP N°3 COMPLEXITÉ : Complexité polynomiale

**BOUDOUR Mehdi / 201500008386/ TP: Complexité polynomiale**



*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 *ProduitMatriciel* :

# Ecrire le programme C qui permet de calculer le produit de 2 matrices A et B :

# Algorithme :

**FONCTION PRODUITMATRICIEL(E/ A:TABLEAU[1..N][1..M] D'ENTIER;**

**E/ B:TABLEAU[1..M][1..P] D'ENTIER ;**

**E/ N, M,P:ENTIER) : TABLEAU[1..N][1..P] D' ENTIER**

**I,J,K : ENTIER;**

**C : TABLEAU[1..N][1..P] D' ENTIER**

**DEBUT**

**POUR I=1 JUSQU'A N FAIRE**

**POUR J=0 JUSQU'A P FAIRE**

**C[I][J]=0;**

3 Boucles Imbriquées

**POUR(K=0 JUSQU'A M FAIRE**

**C[I][J]= C[I][J] +**

**A[I][K]\*B[K][J];**

**FIN POUR ;**

**FIN POUR;**

**FIN POUR;**

1

**RETOURNER C;**

**FIN;**

# Complexité :

Il y’a **3 boucles** à nombres d’itérations explicites.

Il n’est pas difficile de constater que la complexité de l’algorithme donné est déterminée par celle de la boucle externe (sur i).

Le corps de la **boucle interne (sur k)** est en **O(1)** car ne contenant qu’un nombre constant d’instructions élémentaires. Comme cette boucle est itérée ***M*** fois, sa complexité sera donc en **O(p).** La boucle du milieu **(sur j)** est répétée ***P*** fois. Sa complexité est donc en **O(m\*p)**.

La complexité de **la boucle externe (sur i)** est ***N*** fois celle de son corps ; c’est à dire en **O(N\*P\*M)**. Par conséquent, la complexité de tout le l’algorithme est en **O(N\*P\*M).**

T(N,P,M)= +1 = N\*P\*M +1 =∼ O(N\*P\*M)

# Implémentation : En langage C

**long \*\*ProduitMatriciel(long \*\*A,long \*\*B,long n,long m,long p)**

**{**

**long i,j,k;**

**long \*\*C =(long \*\*)malloc(n\*sizeof(long \*));**

**for(i=0;i<n;i++)**

**{   \*(C+i)=(long \*)malloc(p\*sizeof(long));**

**for(j=0;j<p;j++)**

**{**

**C[i][j]=0;**

**for(k=0;k<m;k++)**

**{C[i][j]= C[i][j] + A[i][k]\*B[k][j];}**

**}**

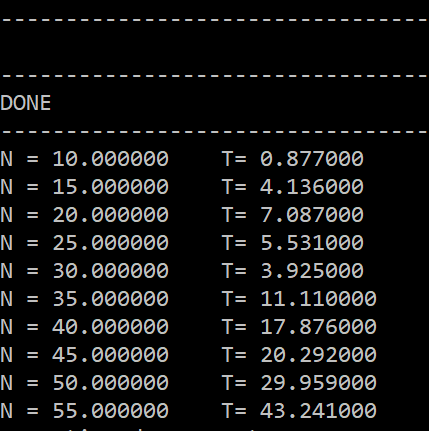
**}**

**return C;**

**}**

# 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 ProduitMatriciel).*

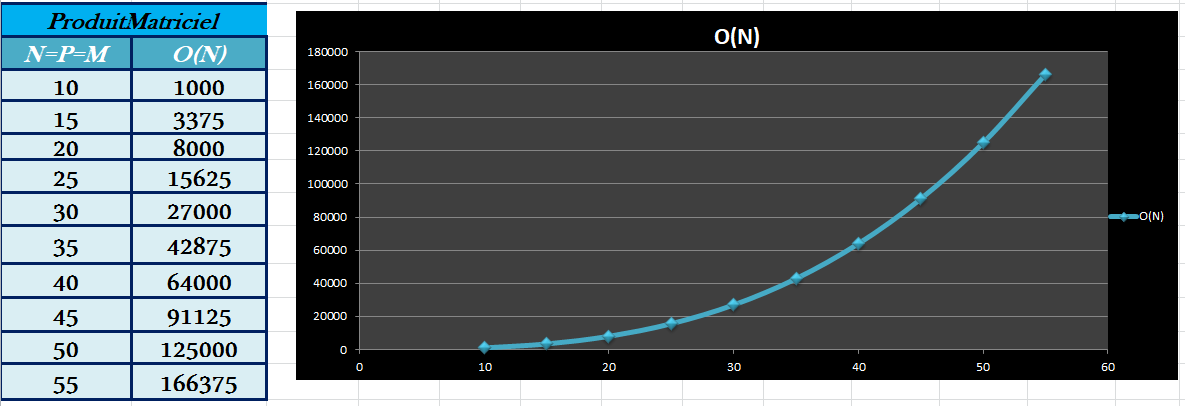


# Représentation Graphique :

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

# C:\Users\Moflawer\AppData\Local\Microsoft\Windows\INetCache\Content.Word\tn1.png

# *Graphe de la complexité théorique en fonction de N.*



# II. Algorithme *sousMat1*:

Soit (𝑛, 𝑚), (𝑛’, 𝑚’) deux tableaux à deux dimensions tel que 𝑛’ < 𝑛 et 𝑚’ < 𝑚. Il s’agit de rechercher le tableau 𝐵 dans le tableau 𝐴. En supposant que les éléments de 𝐴 et 𝐵 ne sont pas triés, écrire une fonction sousMat1 qui retrouve 𝐵 dans 𝐴.

# Algorithme :

**FONCTION SOUSMAT1(E/ A:TABLEAU[1..NA][1..MA] D' ENTIER;**

**E/ B:TABLEAU[1..NB][1..MB] D' ENTIER ;**

**E/ NA,MA,NB,MB:ENTIER): BOOLEEN**

**I,J,K,L:ENTIER;**

**DEBUT**

**POUR I=0 JUSQU'A NA -(NB-1) FAIRE**

**POUR J=0 JUSQU'A MA -(MB-1) FAIRE**

**SI (B[0][0] = A[I][J]) ALORS**

**POUR K=0 JUSQU'A NB FAIR**

**POUR L=0 JUSQU'A MB FAIRE**

**SI (B[K][L]<>A[I+K][J+L]) ALORS**

**/\*SORTIE DES 2 BOUCLES\*/**

**4 Boucles Imbriquées**

**K=NB+1; L=MB+1;**

**SINON**

**SI(K== NB ET L== MB ) ALORS**

**RETOURNER VRAI;**

**FIN SI;**

**FIN SI;**

**FIN POUR;**

**FIN POUR;**

**FIN SI;**

**FIN POUR;**

**FIN POUR;**

**RETOURNER FAUX;**

1

**FIN;**

# Complexité :

**Au pire cas  :** la matrice B est à chaque parcours de ligne sous-matrice de A au dernière élément près ainsi il y a double parcours tous les éléments de la matrice A à quelque élément près (nombre = E = cste) .

T()= +1 =

()() ∼ O(\*\* )

# Implémentation : En langage C

**long sousMat1(long \*\*A,long na,long ma,long \*\*B,long nb,long mb)**

**{**

**long i,j,k,l;**

**for(i=0;i+(nb-1)<na;i++)**

**{**

**for(j=0;j+(mb-1)<ma;j++)**

**{**

**if(B[0][0] == A[i][j])**

**{**

**for(k=0;k<nb;k++)**

**for(l=0;l<mb;l++)**

**{**

**if(B[k][l]!=A[i+k][j+l])**

**{**

**/\*sortie des 2 boucles\*/k=nb; break;**

**}**

**else if(k== nb-1 && l== mb-1 ) return 1;**

**}**

**}**

**}**

**}**

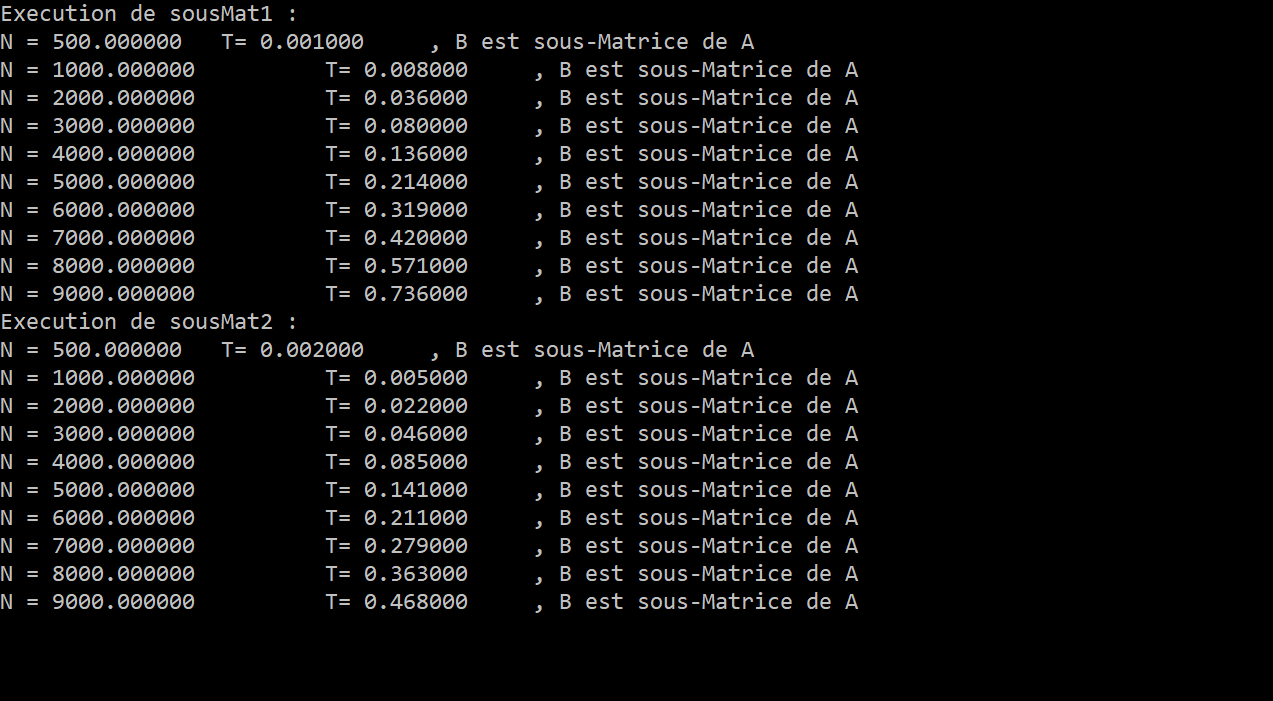
**return 0;**

**}**

# Exécution :

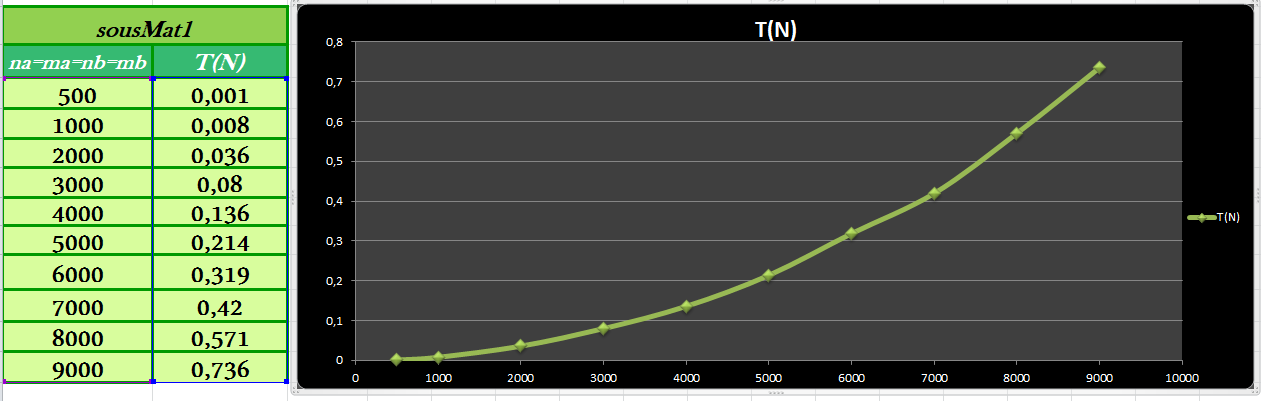
# *Affichage du temps d’exécution de l’algorithme pour chaque valeur de N (T = le temps d’exécution calculé pour chaque exécution de la fonction sousMat1).*

# *Les Matrices choisis sont tel que B est de dimensions (1,1) et en dernière position de A càd : A[n][m]=B[0][0].*

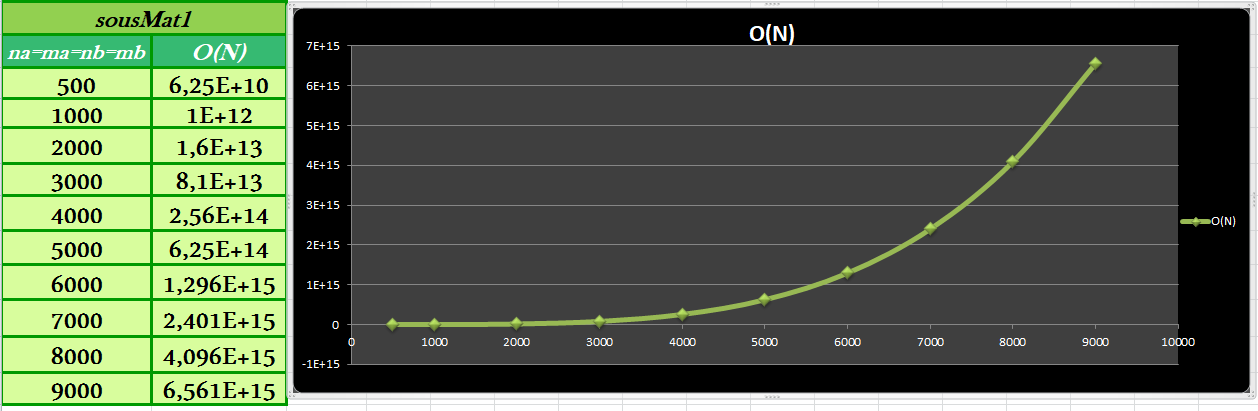


# Représentation Graphique :

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



# *Graphe de la complexité théorique en fonction de N.*



# III. Algorithme *sousMat2*:

# En supposant que chacune des lignes de 𝐴 et 𝐵 est triée par ordre croissant (voir figure), écrire une fonction sousMat2 non naïve de complexité minimale pour trouver 𝐵 dans 𝐴.

# Algorithme :

**FONCTION SOUSMAT2(E/ A:TABLEAU[1..NA][1..MA] D' ENTIER;**

**E/ B:TABLEAU[1..NB][1..MB] D' ENTIER ;**

**E/ NA,MA,NB,MB:ENTIER): ENTIER**

**I,J,K,L:ENTIER;**

**DEBUT;**

**POUR I=0 JUSQU'A NA -(NB-1) FAIRE**

**SI(B[0][0]>=A[I][0] ET B[0][MB-1]<=A[I][MA-1]) ALORS**

**O(log(MA)**

**J = RECHELETS\_DICHO(A[I],MA,B[0][0]);**

**SI (J>=0) ALORS**

**POUR K=0 JUSQU'A NB FAIR**

**POUR L=0 JUSQU'A MB FAIRE**

**SI (B[K][L]<>A[I+K][J+L]) ALORS**

**/\*SORTIE DES 2 BOUCLES\*/**

**K=NB+1; L=MB+1;**

**3 Boucles Imbriquées**

**SINON**

**SI (K== NB-1 ET L== MB-1 ) ALORS**

**RETOURNER J;**

**FIN SI;**

**FIN SI;**

**FIN POUR;**

**FIN POUR;**

**FIN SI;**

**FIN SI;**

**FIN POUR;**

**RETOURNER 0;**

**FIN;**

# Complexité :

**Au pire cas  :** la matrice B est à chaque parcours de ligne sous-matrice de A au dernière élément près ainsi il y a double parcours tous les éléments de la matrice A à quelque élément près (nombre = E = cste) .

T()= +1 =

()( **)**

∼ O()+\* )

# Implémentation : En langage C

**long sousMat2(long \*\*A,long na,long ma,long \*\*B,long nb,long mb)**

**{**

**long i,j,k,l;**

**for(i=0;i+(nb-1)<na;i++)**

**{**

**if(B[0][0]>=A[i][0] && B[0][mb-1]<=A[i][ma-1])**

**{**

**j = rechElets\_Dicho(\*(A+i),ma,B[0][0]);**

**if(j>=0)**

**{**

**for(k=0;k<nb;k++)**

**for(l=0;l<mb;l++)**

**{**

**if(B[k][l]!=A[i+k][j+l])**

**{**

**/\*sortie des 2 boucles\*/k=nb; break;**

**}**

**else if(k== nb-1 && l== mb-1 ) return j;**

**}**

**}**

**}**

**}**

**return 0;**

**}**

# 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 sousMat2).*

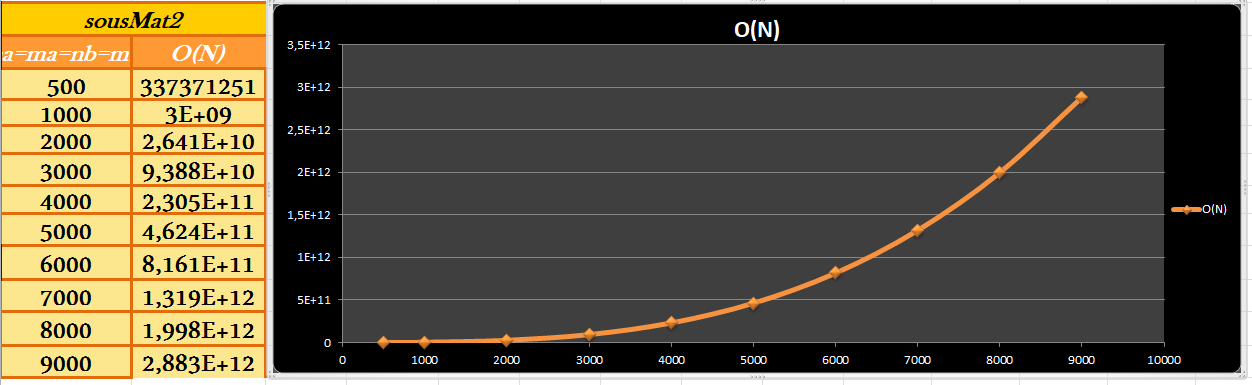
# *Les Matrices choisis sont tel que B est de dimensions (1,1) et en dernière position de A càd : A[n][m]=B[0][0].*

# C:\Users\Moflawer\Desktop\Dol_Gul_Dur\WorkShop_Tree\C\Almost_Done\TP_COMPLEXITÉ\TP3\sousmat.PNG

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.* C:\Users\Moflawer\AppData\Local\Microsoft\Windows\INetCache\Content.Word\tn3.png

# *Graphe de la complexité théorique en fonction de N.*



(\*)Code Source du Programme complet :

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#include <math.h>**

**long rechElets\_Dicho(long \*T,long N , long x)**

**{**

**if(T[0]==x) return 0;**

**else if(T[N-1]==x) return N-1;**

**else if(x<T[0] || x>T[N-1]) return -1;**

**long d=1,f=N-1,m;**

**for(;d<f;m=(d+f)/2)**

**{**

**if(T[m]==x) return m;**

**else**

**if(x<T[m]) f=m;**

**else d=m;**

**}**

**return -1;**

**}**

**long \*\*ProduitMatriciel(long \*\*A,long \*\*B,long n,long m,long p)**

**{**

**long i,j,k;**

**long \*\*C =(long \*\*)malloc(n\*sizeof(long \*));**

**for(i=0;i<n;i++)**

**{   \*(C+i)=(long \*)malloc(p\*sizeof(long));**

**for(j=0;j<p;j++)**

**{**

**C[i][j]=0;**

**for(k=0;k<m;k++){C[i][j]= C[i][j] + A[i][k]\*B[k][j];**

**}**

**}**

**}**

**return C;**

**}**

**//T(N) = O(n\*p\*m) / O(n^3)**

**//S(n,m,p) = Mem(A)+Mem(B)+Mem(C)+Mem(i)+Mem(j)+Mem(k)**

**long sousMat1(long \*\*A,long na,long ma,long \*\*B,long nb,long mb)**

**{**

**long i,j,k,l;**

**for(i=0;i+(nb-1)<na;i++)**

**{**

**for(j=0;j+(mb-1)<ma;j++)**

**{**

**if(B[0][0] == A[i][j])**

**{**

**for(k=0;k<nb;k++)**

**for(l=0;l<mb;l++)**

**{**

**if(B[k][l]!=A[i+k][j+l])**

**{**

**/\*sortie des 2 boucles\*/k=nb; break;**

**}**

**else if(k== nb-1 && l== mb-1 ) return 1;**

**}**

**}**

**}**

**}**

**return 0;**

**}**

**long sousMat2(long \*\*A,long na,long ma,long \*\*B,long nb,long mb)**

**{**

**long i,j,k,l;**

**for(i=0;i+(nb-1)<na;i++)**

**{**

**if(B[0][0]>=A[i][0] && B[0][mb-1]<=A[i][ma-1])**

**{**

**j = rechElets\_Dicho(\*(A+i),ma,B[0][0]);**

**if(j>=0)**

**{**

**for(k=0;k<nb;k++)**

**for(l=0;l<mb;l++)**

**{**

**if(B[k][l]!=A[i+k][j+l])**

**{**

**/\*sortie des 2 boucles\*/k=nb; break;**

**}**

**else if(k== nb-1 && l== mb-1 ) return j;**

**}**

**}**

**}**

**}**

**return 0;**

**}**

**void AfficherMatrice(long \*\*M,long n,long m)**

**{**

**long i,j;**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++)**

**printf("%d\t",M[i][j]);**

**printf("\n");**

**}**

**}**

**long \*\*MatrixToPolonger(long \*M,long rows ,long cols)**

**{**

**long i,j,\*\*R=(long \*\*)malloc(rows\*sizeof(long \*));**

**for (i = 0; i < rows; i++) {**

**\*(R+i)=(long \*)malloc(cols\*sizeof(long ));**

**for (j = 0; j < cols; j++) {**

**R[i][j]= \*(M + i \* cols + j);**

**}**

**}**

**return R;**

**}**

**//Matrice Triï¿½ Ordre dï¿½croissant**

**long \*\*PireCas(long n)**

**{**

**long num=n\*n;**

**long i,j,\*\*T=(long \*\*)malloc(n\*sizeof(long \*));**

**for(i=0;i<n;i++)**

**{**

**T[i] = (long \*)malloc(n\*sizeof(long ));**

**for(j=0;j<n;j++)**

**{T[i][j]=num; num =num-1;}**

**}**

**return T;**

**}**

**//Matrice Triï¿½ Ordre Croissant**

**long \*\*MeilleurCas(long n)**

**{**

**long num=1;**

**long i,j,\*\*T=(long \*\*)malloc(n\*sizeof(long \*));**

**for(i=0;i<n;i++)**

**{**

**T[i] = (long \*)malloc(n\*sizeof(long ));**

**for(j=0;j<n;j++)**

**{T[i][j]=num; num++;}**

**}**

**return T;**

**}**

**double \*\*Calcul\_des\_Temps(double \*\*tab , long algorithme)**

**{**

**long j,verdict,\*\*M;**

**for(j=0 ; j<12 ; j++)**

**{**

**clock\_t begin = clock();**

**switch(algorithme)**

**{**

**case 1: ProduitMatriciel(PireCas(tab[0][j]),MeilleurCas(tab[0][j]),tab[0][j],tab[0][j],tab[0][j]); break;**

**case 2: verdict = sousMat1(PireCas(tab[0][j]),tab[0][j],tab[0][j],MeilleurCas(1),1,1); break;**

**case 3: M = MeilleurCas(1) ; M[0][0]=(long) (tab[0][j]\*(tab[0][j] - 1) +tab[0][j]/2) ;**

**verdict = sousMat2(MeilleurCas(tab[0][j]),tab[0][j],tab[0][j],M,1,1); break;**

**}**

**clock\_t end = clock();**

**tab[1][j] = (double)(end - begin) / CLOCKS\_PER\_SEC;**

**tab[2][j] = verdict;**

**}**

**return tab;**

**}**

**double \*\*Tableau\_de\_ValeursPoduit(void)**

**{**

**long i ;**

**double \*\*tab;**

**tab = (double \*\*)malloc(4\*sizeof(double \*));**

**for(i=0 ; i<4 ; i++) tab[i] = (double \*)malloc(10\*sizeof(double));**

**tab[0][0]=10;**

**tab[0][1]=15;**

**tab[0][2]=20;**

**tab[0][3]=25;**

**tab[0][4]=30;**

**tab[0][5]=35;**

**tab[0][6]=40;**

**tab[0][7]=45;**

**tab[0][8]=50;**

**tab[0][9]=55;**

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

**return tab;**

**}**

**double \*\*Tableau\_de\_ValeursSousMat(void)**

**{**

**long i ;**

**double \*\*tab;**

**tab = (double \*\*)malloc(4\*sizeof(double \*));**

**for(i=0 ; i<4 ; i++) tab[i] = (double \*)malloc(10\*sizeof(double));**

**tab[0][0]=500;**

**tab[0][1]=1000;**

**tab[0][2]=2000;**

**tab[0][3]=3000;**

**tab[0][4]=4000;**

**tab[0][5]=5000;**

**tab[0][6]=6000;**

**tab[0][7]=7000;**

**tab[0][8]=8000;**

**tab[0][9]=9000;**

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

**return tab;**

**}**

**void Afficher\_Tableau\_de\_Valeurs(double \*\*tab)**

**{**

**long j,verdict;**

**for(j=0 ; j<10 ; j++)**

**{**

**verdict = (int)tab[2][j];**

**if(verdict) printf("N = %f \t T= %f \t , B est sous-Matrice de A \n",tab[0][j],tab[1][j]);**

**else printf("N = %f \t T= %f \t , B n'est pas sous-Matrice de A \n",tab[0][j],tab[1][j]);**

**}**

**}**

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

**{**

**//printf("Execution de ProduitMatriciel :\n");**

**//Afficher\_Tableau\_de\_Valeurs(Calcul\_des\_Temps(Tableau\_de\_Valeurs(),1));**

**printf("Execution de sousMat1 :\n");**

**Afficher\_Tableau\_de\_Valeurs(Calcul\_des\_Temps(Tableau\_de\_ValeursSousMat(),2));**

**printf("Execution de sousMat2 :\n");**

**Afficher\_Tableau\_de\_Valeurs(Calcul\_des\_Temps(Tableau\_de\_ValeursSousMat(),3));**

**/\*  int a[5][4] ={**

**{0 ,1 ,2 ,3 },**

**{4 ,5 ,6 ,7 },**

**{8 ,9 ,10,11},**

**{12,13,14,15},**

**{16,17,18,19}**

**},**

**b[2][2]={**

**{9 ,10},**

**{13,14}**

**};**

**int \*\*A=MatrixToPointer(&a[0][0],5,4),**

**\*\*B=MatrixToPointer(&b[0][0],2,2);**

**printf("A = \n");**

**AfficherMatrice(A,5,4);**

**printf("B = \n");**

**AfficherMatrice(B,2,2);**

**printf("Resultat = %d",sousMat2(A,5,4,B,2,2));**

**\*/**

**/\***

**int Mat1[2][3] = {{1,2,0},{4,3,-1}},Mat2[3][2] = {{5,1},{2,3},{3,4}};**

**int \*\*A=MatrixToPointer(&Mat2[0][0],3,2),**

**\*\*B=MatrixToPointer(&Mat1[0][0],2,3);**

**printf("A = \n");**

**AfficherMatrice(A,3,2);**

**printf("B = \n");**

**AfficherMatrice(B,2,3);**

**printf("C = \n");**

**AfficherMatrice(ProduitMatriciel(A,B,3,2,3),3,3);**

**\*/**

**getchar();**

**return 0;**

**}**