Programmation en C

Corrigé TD7

```
 1 /* Manipulation de pointeurs
 2  ------------------------------------------------ */
 3
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6
 7 void echange(int *a, int *b){
 8
 9      int tmp;
10      tmp=*a;
11      *a=*b;
12      *b=tmp;
13 }
14
15 int main (int argc, char *argv[])
16 {
17   int a,b;
18   int *p,*q;
19   a = atoi(argv[1]);
20   b = atoi(argv[2]);
21   printf(" a = %12d\t b = %12d\n
22  p = %12p\t q = %12p\n
23 &a = %12p\t &b = %12p\n
24 *p = %12d\t *q = %12d\n",
25      a,b,p,q,&a,&b,*p,*q);
26
27   p = &a;
28   q = malloc(sizeof(int));
29   printf(" a = %12d\t b = %12d\n
30  p = %12p\t q = %12p\n
31 &a = %12p\t &b = %12p\n
32 *p = %12d\t *q = %12d\n",
33      a,b,p,q,&a,&b,*p,*q);
34
35   *q = b + *p;
36   printf(" a = %12d\t b = %12d\n
```

```
 p = %12p\t q = %12p\n
&a = %12p\t &b = %12p\n
*p = %12d\t *q = %12d\n
q[0] = %12d",
     a,b,p,q,&a,&b,*p,*q,q[0]);
  return 0;
}
```

```
/* Déterminant
 ------------------------------------------------ */

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



int **initialisation(int n) {
  int **M;
  int i,j;
  M = malloc(n*sizeof(int *));
  for (i=0;i<n;i++) {
    M[i] = malloc(n*sizeof(int));
    for (j=0;j<n;j++)
      M[i][j] = rand() % 100;
  }
  return M;
}

void affiche(int **M, int n) {
  int i,j;
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      printf(" %8d",M[i][j]);
    printf("\n");
  }
  printf("\n");
}

int ** addition(int **A, int **B, int n){
  int ** C = initialisation(n);
  int i,j;

  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      C[i][j] = A[i][j] + B[i][j];
```

```
39
40   return C;
41 }
42
43 int ** multiplication(int **A, int **B, int n){
44   int ** C = initialisation(n);
45   int i,j,k;
46
47   for (i=0; i<n; i++)
48     for (j=0; j<n; j++){
49       C[i][j]= 0;
50       for(k=0;k<n;k++)
51     C[i][j] += A[i][k]*B[k][j];
52     }
53
54   return C;
55 }
56
57
58 int ** copie(int **M, int n){
59   int i;
60   int **N=malloc(n*sizeof(int *));
61
62   for(i=0; i<n; i++)
63     N[i]=M[i];
64
65   return N;
66 }
67
68 int determinant(int **M, int n) {
69   int **S;
70   int s=1,d=0, k, i;
71   /* Cas simples: n=0 et n=1 */
72   if (n==1) return M[0][0];
73   d=M[n-1][n-1]*determinant(M,n-1);
74   s=1;
75   for (k=n-2; k>=0; k--) {
76     s=-s; S=copie(M,n);
77     for (i=k; i<n; i++)
78       S[i]=S[i+1];
79     d=d+s*M[k][n-1]*determinant(S,n-1);
80     free(S);
81   }
82   return d;
83 }
84
```

```
85 int main (int argc, char *argv[]) {
86   int **A,n;
87   int **B;
88
89   srand(getpid());
90   n = atoi(argv[1]);
91   A = initialisation (n);
92   B = initialisation (n);
93
94   printf("Affiche A\n");
95   affiche(A,n);
96
97   printf("Affiche B\n");
98   affiche(B,n);
99
100  printf("Affiche somme de A et B\n");
101  affiche(addition(A,B,n),n);
102
103  printf("Affiche produit de A et B\n");
104  affiche(multiplication(A,B,n),n);
105
106  printf("Matrice A\n");
107  affiche(A,n);
108  printf("Déterminant de A = %d \n",determinant(A,n));
109
110  return 0;
111 }
```

La complexité de l'algorithme récursif du calcul du déterminant est $n!$.

1 Solution Algorithme de Strassen

1. Le nombre d'addition est $n^2(n-1)$ et le nombre de multiplications n^3.
2. Dans l'algorithme naïf, il y a 8 multiplications et 4 additions, alors que dans l'algorithme de Strassen il y a 7 multiplications et 18 additions.
3. On a m^3 multiplications pour chaque p_i, d'où $Mult(n) = 7m^3 = Tn^3/8$ et pour les additions, il y a les p_i, donc $7m^2(m-1)$ et les 18 matrices auxiliaires au nombre de $18m^2$, soit $Add(n) = 7m^3 + 18m^2 = 7n^3/8 + 18n^2/4$.

   La méthode de Strassen est plus efficace que l'algorithme naïf car une multiplication de matrice $n \times n$ coûte n^3 alors qu'une addition coûte n^2.
4. On a $M(1) = 1$ et $M(n) = 7 \times M(n/2)$ donc $M(n) = 7^{\log_2(n)} = n^{\log_2(7)}$. On a $A(1) = 0$ et $A(n) = 7 \times A(n/2) + 18 \times (n/2)^2$ donc $A(n) = 6 \times (n^{\log_2(7)} - n^2)$.