Algorithmes sur les grands nombres

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

Motivations : pourquoi étudier les algorithmes pour les grands nombres?

- Manipuler des nombres ayant un nombre important de chiffres

Motivations : pourquoi étudier les algorithmes pour les grands nombres?

- Manipuler des nombres ayant un nombre important de chiffres
- Les opérations arithmétiques standards ne se font plus en $O(1)$

Motivations : pourquoi étudier les algorithmes pour les grands nombres?

- Manipuler des nombres ayant un nombre important de chiffres
- Les opérations arithmétiques standards ne se font plus en $O(1)$
- Très utilisé dans de nombreuses applications :

Motivations : pourquoi étudier les algorithmes pour les grands nombres?

- Manipuler des nombres ayant un nombre important de chiffres
- Les opérations arithmétiques standards ne se font plus en $O(1)$
- Très utilisé dans de nombreuses applications :
  - Cryptographie : RSA (par exemple)
  - Signature des documents
  - Fonctions de hachages
  - etc.

Produit vs Somme des entiers

Produit vs Somme

Lorsque l'on manipule des valeurs numériques de type standard (entier, réel etc) alors :

le temps d'exécution des produits de nombres

Produit vs Somme des entiers

Produit vs Somme

Lorsque l'on manipule des valeurs numériques de type standard (entier, réel etc) alors :

le temps d'exécution des produits de nombres
reste proche

Produit vs Somme des entiers

Produit vs Somme

Lorsque l'on manipule des valeurs numériques de type standard (entier, réel etc) alors :

le temps d'exécution des produits de nombres
reste proche
à celui nécessaire pour la réalisation des sommes de nombres

Exemple : Produit vs Somme

```
double produit(double M, int N)
{
    int i;
    for (i=0; i<N; i++)
    {
        M=M*M;
    }
}

double addition(double M, int N)
{
    int i;
    for (i=0; i<N; i++)
    {
        M=M+M;
    }
}
```

Somme vs Produit

Remarque

- Ces deux fonctions sont quasi-identiques.

Somme vs Produit

Remarque

- Ces deux fonctions sont quasi-identiques.
- Dans l'une des fonction l'opération produit "*" est utilisée alors dans l'autre fonction l'opération somme "+" est utilsée.

Somme vs Produit

Remarque

- Ces deux fonctions sont quasi-identiques.
- Dans l'une des fonction l'opération produit "*" est utilisée alors dans l'autre fonction l'opération somme "+" est utilsée.
- Regardons le temps d'exécution de ces deux fonctions avec :

$$M = 9223372036854775807.$$

Somme vs Produit ($N \leq 10^6$)

Temps d'exécution en secondes

Nbre_elements	Temps_produit	Temps_somme
100000	0.000579	0.000426
200000	0.001131	0.000850
300000	0.001697	0.001289
400000	0.002012	0.001427
500000	0.002224	0.001476
600000	0.002362	0.001773
700000	0.002773	0.002065
800000	0.003149	0.002368
900000	0.003544	0.002661
1000000	0.003936	0.002951

Somme vs Produit ($N \leq 10^9$)

Temps d'exécution en secondes

Nbre_elements	Temps_produit	Temps_somme
100000000	0.394204	0.295470
200000000	0.787298	0.590014
300000000	1.180548	0.885297
400000000	1.574570	1.180404
500000000	1.967312	1.476103
600000000	2.365610	1.776160
700000000	2.758570	2.067422
800000000	3.151554	2.362312
900000000	3.545822	2.661512
1000000000	3.940624	2.953201

Somme vs Produit

Remarque

- Pour obtenir une différence d'une seconde, il faudra qu'il y ait au moins 10^9 opérations arithmétiques de base (produit ou somme)

Somme vs Produit

Remarque

- Pour obtenir une différence d'une seconde, il faudra qu'il y ait au moins 10^9 opérations arithmétiques de base (produit ou somme)
- On verra plus tard que ce ne sera plus le cas lorsque les nombres contiennent plusieurs centaines de chiffres.

Pour commencer

Evaluation d'un polynôme

Polynômes

Définition

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

où :

- a_i sont des réels (positifs ou négatifs ou nuls)
- a_n est différent de zéro
- n est appelé le degré du polynôme $p(x)$.

Polynômes

Définition

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

où :

- a_i sont des réels (positifs ou négatifs ou nuls)
- a_n est différent de zéro
- n est appelé le degré du polynôme $p(x)$.

But

Evaluer $p(x)$ lorsque $x = x_0$, c'est-à-dire lorsque x prend une valeur particulière x_0.

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.
- L'alphabet est composé de 43 caractères :

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.
- L'alphabet est composé de 43 caractères :
  - 26 lettres minuscules $\{a, b, ..., z\}$ et

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.
- L'alphabet est composé de 43 caractères :
  - 26 lettres minuscules $\{a, b, ..., z\}$ et
  - 17 lettres accentués et ligature

    {à, â, ä, ç, é, è, ê, ë, î, ï, ô, ö, ù, û, ü, æ, œ}

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.
- L'alphabet est composé de 43 caractères :
  - 26 lettres minuscules $\{a, b, ..., z\}$ et
  - 17 lettres accentués et ligature

    $\{$à, â, ä, ç, é, è, ê, ë, î, ï, ô, ö, ù, û, ü, æ, œ$\}$
- un mot est composé d'une suite de ces 43 caractères

Evaluation des polynômes : Dictionnaire

stocker les mots d'un dictionnaire

- Supposons que nous souhaitons stocker tous les mots d'un dictionnaire (par exemple dictionnaire de la langue française) dans une tableau.
- L'alphabet est composé de 43 caractères :
  - 26 lettres minuscules $\{a, b, ..., z\}$ et
  - 17 lettres accentués et ligature

    $\{$à, â, ä, ç, é, è, ê, ë, î, ï, ô, ö, ù, û, ü, æ, œ$\}$
- un mot est composé d'une suite de ces 43 caractères
- Le mot le plus long contient 25 caractères qui est "anticonstitutionnellement"

Dictionnaire

stocker les mots d'un dictionnaire

- Comme un dictionnaire contient moins d'un million de mots. Il suffit (intuitivement) alors de créer un tableau de un million cases.

Dictionnaire

stocker les mots d'un dictionnaire

- Comme un dictionnaire contient moins d'un million de mots. Il suffit (intuitivement) alors de créer un tableau de un million cases.
- chacune des cases contiendra un mot du dictionnaire.

Dictionnaire

stocker les mots d'un dictionnaire

- Comme un dictionnaire contient moins d'un million de mots. Il suffit (intuitivement) alors de créer un tableau de un million cases.
- chacune des cases contiendra un mot du dictionnaire.
- Il faut cependant transformer chaque mot en entier qui représentera l'indice où le mot est stocké dans le tableau.

Dictionnaire

stocker les mots d'un dictionnaire

- Comme un dictionnaire contient moins d'un million de mots. Il suffit (intuitivement) alors de créer un tableau de un million cases.
- chacune des cases contiendra un mot du dictionnaire.
- Il faut cependant transformer chaque mot en entier qui représentera l'indice où le mot est stocké dans le tableau.
- Une façon d'avoir des indices numériques est de transformer chaque mot du dictionnaire vers un entier.

Dictionnaire

stocker les mots d'un dictionnaire

- Associer à chaque lettre *a* à un entier code(a)=$\{1, ..., 43\}$.
  $a \to 1$, $b \to 1$, ..., $z \to 26$, $à \to 27$, . . . , $œ \to 43$.

Dictionnaire

stocker les mots d'un dictionnaire

- Associer à chaque lettre a à un entier code(a)=$\{1, ..., 43\}$.
  $a \to 1$, $b \to 1$, ..., $z \to 26$, $à \to 27$, . . . , $œ \to 43$.
- Un mot de la forme $x_n x_{n-1} ... x_1 x_0$ est transformée en :

Dictionnaire

stocker les mots d'un dictionnaire

- Associer à chaque lettre a à un entier code(a)=$\{1, ..., 43\}$.
  $a \to 1$, $b \to 1$, ..., $z \to 26$, $à \to 27$, . . . , $œ \to 43$.
- Un mot de la forme $x_n x_{n-1} ... x_1 x_0$ est transformée en :
  $Code(x_n x_{n-1} ... x_1 x_0)$
  $= \sum_{i=1}^{i=n} code(x_i) * 43^i$
  $= code(x_0) + code(x_1) * 43 + code(x_2) * 43^2 + ... + code(x_n) * 43^n$.

Dictionnaire

stocker les mots d'un dictionnaire

- Associer à chaque lettre a à un entier code(a)=$\{1, ..., 43\}$.
  $a \to 1$, $b \to 1$, ..., $z \to 26$, $à \to 27$, . . . , $œ \to 43$.
- Un mot de la forme $x_n x_{n-1} ... x_1 x_0$ est transformée en :
  $Code(x_n x_{n-1} ... x_1 x_0)$
  $= \sum_{i=1}^{i=n} code(x_i) * 43^i$
  $= code(x_0) + code(x_1) * 43 + code(x_2) * 43^2 + ... + code(x_n) * 43^n$.
- Le mot "a" a un indice 1
- Le mot "lens" a un indice 963950.

Dictionnaire

stocker les mots d'un dictionnaire

- Un mot de la forme $x_n x_{n-1} \ldots x_1 x_0$ est transformée en :
  $Code(x_n x_{n-1} \ldots x_1 x_0)$
  $= \sum_{i=1,\ldots,n} code(x_i) * 43^i$
  $= code(x_0) + code(x_1) * 43 + code(x_2) * 43^2 + \ldots + code(x_n) * 43^n$.

Dictionnaire

stocker les mots d'un dictionnaire

- Un mot de la forme $x_n x_{n-1} \ldots x_1 x_0$ est transformée en :
  $Code(x_n x_{n-1} \ldots x_1 x_0)$
  $= \sum_{i=1,\ldots,n} code(x_i) * 43^i$
  $= code(x_0) + code(x_1) * 43 + code(x_2) * 43^2 + \ldots + code(x_n) * 43^n$.

- Ce problème revient à évaluer un polynôme :

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

où :

  - $a_i = code(x_i)$
  - p(x) est évalué pour $x = 43$.

Représentatation d'un polynôme

Tableau

Un polynôme :

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

sera tout simplement représenté par un tableau de réels :

a_0	a_1	a_2	....	a_{n-1}	a_n
0	1	2	....	$n-1$	n

Evaluation d'un polynôme

Exercice

- Ecrire une fonction qui évalue un polynôme :

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

  en un point x_0.
- Evaluer sa complexité.

Version naïve de l'évaluation des polynômes

```
double poly_naif(double A[], double x_0, int N)
{
    int i, j;
    double res=0, puissance;
    for (i=0; i<=N; i++)
    {
        puissance=1;
        for (j=0; j<i; j++)
        {
            puissance=puissance*x_0;
        }
        res=res+A[i]*puissance;

    }
    return res;
}
```

Version naïve de l'évaluation des polynômes

Question

Quelle est la complexité de la version naïve de l'évaluation des polynômes?

Version naïve de l'évaluation des polynômes

Analysons l'algorithme qui commence par une affectation :

```
2 double poly_naif(double A[], double x_0, int N)
3 {
4     int i, j;

2     double res=0, puissance;

      ...
2 }
```

A ce niveau là, nous avons : 1 instruction.

Version naïve de l'évaluation des polynômes

```
double poly_naif(double A[], double x_0, int N)
{
.

   for (i=0; i<=N; i++)
   {
      A(i) instructions
   }

.
}
```

Le nombre d'instructions pour la réalisation de cette boucle est :

- On a une instruction pour l'initialisation de $i = 0$
- A chaque étape de la boucle on fait :
  - Une incrémentation plus une comparaisaon
  - Et A(i) instructions en fonction de la valeur de i

Version naïve de l'évaluation des polynômes

Question

Quelle est la complexité de la version naïve de l'évaluation des polynômes?

Version naïve de l'évaluation des polynômes

Question

Quelle est la complexité de la version naïve de l'évaluation des polynômes?

Réponse

$O(n^2)$

Evaluation d'un polynôme

Exercice

- Ecrire une fonction, en $O(n * log_2(n))$, qui évalue un polynôme :

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

en un point x_0.

Version naïve de l'évaluation des polynômes

Question

Peut-on faire mieux?

Version naïve de l'évaluation des polynômes

Question

Peut-on faire mieux?

Réponse

Oui. Utiliser une fonction de puissance plus efficace (celle vue en TD).

Version améliorée de l'évaluation des polynômes

```
2  double poly_amelioree(double A[], double x_0, int N)
3  {
4      int i;
5      double res=0, puissance;
6      for (i=0; i<=N; i++)
7      {
8          puissance=puissance_rapide(x_0,i);
9          res=res+A[i]*puissance;
0      }
1      return res;
2  }
```

Version améliorée de l'évaluation des polynômes

Question

Quelle est la complexité de la version améliorée de l'évaluation des polynômes?

Version améliorée de l'évaluation des polynômes

Question

Quelle est la complexité de la version améliorée de l'évaluation des polynômes?

Réponse

$O(n * log_2(n))$

Evaluation des polynômes

Remarque

Dans la version naïve et améliorée, le calcul de la puissance se fait plusieurs fois.

Evaluation des polynômes

Remarque

Dans la version naïve et améliorée, le calcul de la puissance se fait plusieurs fois.

Evaluation d'un polynôme

Exercice

- Ecrire une fonction, en $O(n)$, qui évalue un polynôme :

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

en un point x_0.

Version améliorée de l'évaluation des polynômes

```
2  double poly_efficace(double A[], double x_0, int N)
3  {
4      int i;
5      double p=0, x=1.;
6      for (i=0; i<=N; i++)
7      {
8          p = p + A[i]*x;
9          x=x*x_0;
0      }
1      return p;
2  }
```

Algorithme de Horner

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Redéfinissons les nouvelles constantes :

$$y_0 = a_n$$

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Redéfinissons les nouvelles constantes :

$$\begin{aligned} y_0 &= a_n \\ y_1 &= a_{n-1} + y_0 * x_0 \end{aligned}$$

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Redéfinissons les nouvelles constantes :

$$\begin{aligned} y_0 &= a_n \\ y_1 &= a_{n-1} + y_0 * x_0 \\ y_2 &= a_{n-2} + y_1 * x_0 \end{aligned}$$

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Redéfinissons les nouvelles constantes :

$$\begin{aligned} y_0 &= a_n \\ y_1 &= a_{n-1} + y_0 * x_0 \\ y_2 &= a_{n-2} + y_1 * x_0 \\ &\dots \end{aligned}$$

Version améliorée de l'évaluation des polynômes

Principe

Il suffit de ré-écrire le polynôme

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + a_n x_0)\cdots))$$

Redéfinissons les nouvelles constantes :

$$\begin{aligned} y_0 &= a_n \\ y_1 &= a_{n-1} + y_0 * x_0 \\ y_2 &= a_{n-2} + y_1 * x_0 \\ &\cdots \\ y_n &= a_0 + y_{n-1} * x_0 \end{aligned}$$

Et à la fin $y_n = p(x_0)$

Version efficace de l'évaluation des polynômes

```
2  double poly_horner(double A[], double x_0, int N)
3   {
4     int i;
5     double y=0;
6     for (i=N; i>=0; i--)
7     {
8        y = y*x_0 + A[i];
9     }
0     return y;
    }
```

Version efficace de l'évaluation des polynômes

Question

Quelle est la complexité de la version efficace de l'évaluation des polynômes?

Version efficace de l'évaluation des polynômes

Question

Quelle est la complexité de la version efficace de l'évaluation des polynômes?

Réponse

$O(n)$

Retour sur l’exemple des dictionnaires

stocker les mots d’un dictionnaire

- Le problème se pose avec des mots plus longs :
  - Le mot ”anticonstitutionnellement” a un indice (40 chiffres):
    2 134 105 435 261 400 063 822 702 181 021 025 337 279
  - Et le mot ”œœœœœœœœœœœœœœœœœœœœœœœœœ” (le mot le plus long imaginable dans le vocabulaire) a comme indice (41 chiffres):
    70 295 990 495 133 038 898 032 360 278 502 608 251 643
- Ces nombres ne peuvent plus être stockés avec des types ”standards”

Représentation d'un grand nombre

Représentation d'un grand nombre

- On ne manipule que des entiers positifs dans ce cours
- Un tableau de taille "Taille"
- Chaque case représente un chiffre
- L'indice "Taille-1" représente les unités, "Taille-2" représente les dizaines
- Le chiffre de poids le plus fort est donné dans la case "0

La base et la complexité

- Nombre de chiffres dans un entier n est donné approximativement par $log_{10}(n)$ (ou tout simplement $log(n)$ ou $lg(n)$)
- Le calcul de complexité se fait toujours en $log(n)$
- Souvent des calculs se font en base 2.
- On rappelle que donner la complexité en base 2 ou base 10 (ou toute autre base) est équivalent. Par exemple,

$$O((log_2(n))^2) = O((log_{10}(n))^2)$$

Addition de deux grands nombres

- Soit T1 et T2 deux tableaux qui codent deux grands nombres
- Chaque case représente un chiffre

Exercice

- Ecrire un algorithme qui calcule la somme de deux grands nombres (addition de deux tableaux de chiffres).
- Evaluer sa complexité.

Addition de deux grands nombres

```
int *addition (int A[Taille], int B[Taille])
{
    int i;
    int *C=malloc(Taille*sizeof(int));
    int ret=0;
    for (i=(Taille-1); i>=0; i--)
    {
        if ((A[i]+B[i]+ret) >= 10)
                {C[i]=(A[i]+B[i]+ret)-10; ret=1;}
        else
                {C[i]=A[i]+B[i]+ret; ret=0;}
    }
    return C;
}
```

Soustraction de deux grands nombres

- Soit T1 et T2 deux tableaux qui codent deux grands nombres
- Chaque case représente un chiffre

Exercice

- Ecrire un algorithme qui calcule la différence de deux grands nombres (soustraction de deux tableaux de chiffres). On suppose que le premier nombre est plus grand que le second
- Evaluer sa complexité.

Soustraction de deux grands nombres

```
int *soustraction (int A[Taille], int B[Taille])
{

    int i, diff;
    int *C=malloc(Taille*sizeof(int));
    int ret=0;
    for (i=(Taille-1); i>=0; i--)
    {
        diff=A[i]-(B[i]+ret);
        if (diff>=0)
            {C[i]=diff;
            ret=0;}
        else
            {C[i]=10+diff;ret=1;}
    }
    return C;
}
```

Produit d'un grand nombre par un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre

Produit d'un grand nombre par un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre

Exercice

- Ecrire une fonction qui calcule le produit d'un grand nombre par un chiffre
- Evaluer la complexité de cet algorithme

Produit d'un grand nombre par un chiffre

```
int *produit_un_chiffre (int A[Taille], int B)
{
    int i;
    int *C=malloc(Taille*sizeof(int));
    int ret=0;
    for (i=(Taille-1); i>=0; i--)
    {
        C[i]=(A[i]*B +ret) % 10;
        ret=(A[i]*B+ret) / 10;
    }
    return C;
}
```

Comparaison de deux grands nombres

Données

- Deux tableaux qui codent deux grands nombres

Comparaison de deux grands nombres

Données

- Deux tableaux qui codent deux grands nombres

Exercice

- Ecrire une fonction qui compare deux grands nombres
- Evaluer la complexité de cet algorithme

Comparaison de deux grands nombres

```
int comparer (int A[Taille], int B[Taille])
{
    int i=0;
    while ((A[i]==B[i]) && (i<(Taille-1)))
    {
        i++;
    }
    return (A[i]>=B[i]);

}
```

Egalité entre deux grands nombres

Données

- Deux tableaux qui codent deux grands nombres

Egalité entre deux grands nombres

Données

- Deux tableaux qui codent deux grands nombres

Exercice

- Ecrire une fonction qui rend vraie si les deux grands nombres sont égaux
- Evaluer la complexité de cet algorithme

Egalité entre deux grands nombres

```
int Egalite (int A[Taille], int B[Taille])
{
    return (Comparer (A,B) && Comparer (B,A));
}
```

Décalage d'un tableau et insertion d'un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre

Décalage d'un tableau et insertion d'un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre

Exercice

- Ecrire une fonction qui décale le tableau à gauche et qui insère un chiffre *c* à la position "Taille-1"
- Evaluer la complexité de cet algorithme

Décalage d'un tableau et insertion d'un chiffre

Ce qui est demandé :

```
int * Decale_un_chiffre (int A[Taille], int B)
{
    Un decalage vers la gauche
    Inserer B

}
```

Décalage d'un tableau et insertion d'un chiffre

```
2  int * Decale_un_chiffre (int A[Taille], int B)
3  {
4      int i;
5      int *C=malloc(Taille*sizeof(int));
6      for (i=(Taille-2); i>=0; i--)
7      {
8          C[i]=A[i+1];
9      }
0      C[Taille-1]=B;
1      return C;
2  }
```

Décalage de plusieurs positions et insertion d'un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre à insérer
- nombre de décalage

Décalage de plusieurs positions et insertion d'un chiffre

Données

- Un tableau qui code un grand nombre
- un chiffre à insérer
- nombre de décalage

Exercice

- Ecrire une fonction qui décale le tableau à gauche de plusieurs position et qui insère un chiffre à la position "Taille-1"
- Evaluer la complexité de cet algorithme

Décalage de plusieurs positions et insertion d'un chiffre

```
2  int *Decale_plus_chiffres (int A[], int nbre_de_chiffres, int B)
3  {
4
5     Decalage vers la gauche de Nbre_de_chiffres positions
6     Inserer B a la position 'Taille-1'
7  }
```

Décalage de plusieurs positions et insertion d'un chiffre

```
int *Decale_plus_chiffres (int A[], int nbre_de_chiffres, int B)
{
   int i;
   int *C=malloc(Taille*sizeof(int));
   for (i=(Taille-1-nbre_de_chiffres); i>=0; i--)
   {
      C[i]=A[i+nbre_de_chiffres];
   }
   for (i=(Taille-nbre_de_chiffres); i<=(Taille-2); i++)
   {
      C[i]=0;
   }
   C[Taille-1]=B;
   return C;
}
```

Produit de deux grands nombres

- Soit T1 et T2 deux tableaux qui codent deux grands nombres
- Chaque case représente un chiffre

Exercice

- Ecrire un algorithme qui calcule le produit de deux grands nombres (addition de deux tableaux de chiffres).
- Evaluer sa complexité.

Produit de deux grands nombres

```
int *Produit_deux_grands_nombres (int A[Taille], int B[Taille])
{
  int i,j;
  int *C=malloc(Taille*sizeof(int));
  int *inter=malloc(Taille*sizeof(int));
  for (i=(Taille-1); i>=0; i--) { C[i]=0;}
  for (i=(Taille-1); i>=0; i--)
  {
    inter=produit_un_chiffre(A,B[i]);
    inter=Decale_plusieurs_chiffres (inter,(Taille-1)-i,0);
    C=addition(C,inter);
  }
  return C;
}
```

Division de

deux grands nombres