Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon
`dominique.poulalhon@irif.fr`

Université Paris Cité

L2 Informatique & DL Info-Bio, Info-Jap, Math-Info

Année universitaire 2023-2024

Organisation du module

Emploi du temps

- Cours : 2h par semaine, mardi ***14h-16h***, amphi 1A
  (naturellement) aussi obligatoire que les TD et TP
- TD : 2h par semaine *à partir du 29 janvier*
- TP : 2h par quinzaine
  à partir du 29 janvier pour les groupes INFO 1 et 5, BI, JI, MI
  à partir du 5 février pour les groupes INFO 2, 3 et 4

Assiduité :

- appel ou émargement en TD
- rendu de TP systématique : en fin de séance, modifiable pendant quelques jours

Équipe enseignante

Responsable du cours : Dominique Poulalhon

dominique.poulalhon@irif.fr

Chargés de TD-TP

- Groupe INFO 1 : Mónika Csikós — csikos@irif.fr
- Groupe INFO 2 : Mikaël Rabie — mikael.rabie@irif.fr
- Groupe INFO 3 : Dominique Poulalhon
- Groupe INFO 4 : Yan Jurski — jurski@irif.fr
- Groupe INFO 5 : Aymeric Walch — walch@irif.fr
- Groupe MI 1 : Enrique Roman Calvo — calvo@irif.fr
- Groupe MI 2 : Matthieu Picantin — picantin@irif.fr

Communication

Pour nous écrire, toujours mentionner [EA4] dans le sujet (et signer)

Communication

Pour nous écrire, toujours mentionner `[EA4]` dans le sujet (et signer)

Un site Moodle pour les annonces, les énoncés, les rendus de TP, les auto-évaluations (et les évaluations)

Inscrivez-vous dans le bon groupe !

Communication

Pour nous écrire, toujours mentionner `[EA4]` dans le sujet (et signer)

Un site Moodle pour les annonces, les énoncés, les rendus de TP, les auto-évaluations (et les évaluations)

Inscrivez-vous dans le bon groupe !

Un serveur discord (lien sur moodle à venir) pour toutes vos questions – salons séparés pour le cours, les TD, les TP, les (auto-)évaluations

Éditez votre pseudo au format « Prénom Nom (groupe) »

(en cas de message privé, mentionnez à propos de quel cours vous nous écrivez, et précisez votre nom si votre pseudo usuel n'est pas transparent)

Modalités de contrôle des connaissances

Session 1 : Contrôle Continu Intégral, avec a priori :

- un contrôle sur table à mi-semestre,
- deux évaluations via moodle,
- un contrôle final sur table qui comptera pour 50%.

Session 2 : Examen sur table

Thème du cours

algorithmique = « conception et analyse des algorithmes »

Thème du cours

algorithmique = « conception et analyse des *algorithmes* »

Thème du cours

algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

Thème du cours

algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
  (opérations arithmétiques, approximation de π, de $\sqrt{2}$...)

Thème du cours

algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
  (opérations arithmétiques, approximation de π, de $\sqrt{2}$...)
- des constructions géométriques
  (milieu d'un segment, triangle équilatéral, droites parallèles, centre d'un cercle, pentagone régulier...)

Thème du cours

algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

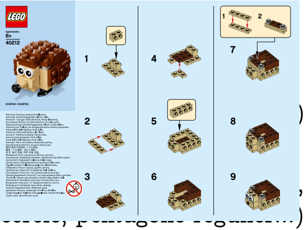
- des algorithmes de calcul
  (opérations arithmétiques, approximation de π, de $\sqrt{2}$...)
- des constructions géométriques
  (milieu d'un segment, triangle les,
  centre d'u ...)
- des recettes de cuisine

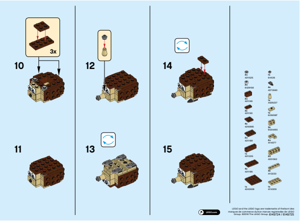
algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs,
ont été décrits bien avant l'invention des ordina

- des algorithmes de calcul
  (opérations arithmétiques, ap)
- des constructions géométriques
  (milieu d'un segment, triangle é ,
  centre d'un)
- des recettes de cuisine
- des manuels de construction…





Thème du cours

algorithmique = « conception et analyse des ***algorithmes*** »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
  (opérations arithmétiques, approximation de π, de $\sqrt{2}$...)
- des constructions géométriques
  (milieu d'un segment, triangle équilatéral, droites parallèles, centre d'un cercle, pentagone régulier...)
- des recettes de cuisine
- des manuels de construction...

mais le concept a pris une importance particulière avec l'apparition de machines capables d'exécuter *fidèlement* et *rapidement* une suite d'opérations prédéfinie

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de résolution d'un problème »

Étymologie : Muhammad Ibn Mūsā al-Khuwārizmī, mathématicien persan du début du 9e siècle

« *Kitābu 'l-mukhtasar fī hisābi 'l-jabr wa'l-muqābalah* » ou « *Abrégé du calcul par la restauration et la comparaison* » : considéré comme le premier manuel d'algèbre, explique comment résoudre les équations du second degré

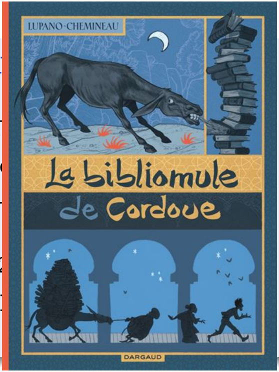
traduit en latin et diffusé en Europe à partir du 12e siècle

(c'est aussi grâce à un de ses livres que se répand la notation positionnelle décimale venue d'Inde)

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de résolution d'un problème »



Étymologie : Muhammad Ibn Mūsā al-Khuwārizmī
du début du 9^{e} siècle

« *Kitābu 'l-mukhtasar fī hisābi 'l-jabr wa'l-muqb*
calcul par la restauration et la comparaison » : c
premier manuel d'algèbre, explique comment résou
second degré

traduit en latin et diffusé en Europe à partir du 12

(c'est aussi grâce à un de ses livres que se répand l
décimale venue d'Inde)

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de résolution d'un problème »

Étymologie : Muhammad Ibn Mūsā al-Khuwārizmī, mathématicien persan du début du 9ᵉ siècle

« *Kitābu 'l-mukhtasar fī hisābi 'l-jabr wa'l-muqb̄alah* » ou « *Abrégé du calcul par la restauration et la comparaison* » : considéré comme le premier manuel d'algèbre, explique comment résoudre les équations du second degré

traduit en latin et diffusé en Europe à partir du 12ᵉ siècle

(c'est aussi grâce à un de ses livres que se répand la notation positionnelle décimale venue d'Inde)

le terme ***algorithme*** est d'abord utilisé pour désigner les méthodes (de calcul) utilisant des chiffres, par opposition au ***calcul*** traditionnel (du latin *calculus*, petit caillou) avec des abaques

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

- preuve de correction

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

- preuve de correction

- étude de l'efficacité

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

- preuve de correction

  un algorithme est correct si, pour chaque entrée, il termine en produisant la bonne sortie

- étude de l'efficacité

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

- preuve de correction

  un algorithme est correct si, pour chaque entrée, il termine en produisant la bonne sortie

- étude de l'efficacité

  les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes
  y a-t-il des techniques générales ?

- preuve de correction
  un algorithme est correct si, pour chaque entrée, il termine en produisant la bonne sortie

- étude de l'efficacité
  les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

Thème du cours

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de ***résolution*** d'un problème »

trois axes d'étude :

- conception des algorithmes

  y a-t-il des techniques générales ?

- preuve de correction

  un algorithme est correct si, pour chaque entrée, il termine en produisant la bonne sortie

- étude de l'efficacité

  les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

(et au passage, on apprendra un peu de Python, parce que c'est un joli langage particulièrement adapté à l'algorithmique)

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline \end{array}$$

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{rcccc} & & & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & & & & 5 \end{array}$$

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$
\begin{array}{ccccc}
 & & 1 & 1 & \\
 & 1 & 3 & 5 & 7 \\
+ & 2 & 4 & 6 & 8 \\
\hline
 & & & 2 & 5
\end{array}
$$

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & 1 & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & & 8 & 2 & 5 \end{array}$$

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & 1 & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & 1 & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

```
def addition(nb1, nb2) :
  # nb1, nb2 : tableaux de chiffres décimaux,
  # représentant des entiers n1 et n2,
  # supposés de même longueur (en commençant par les unités)
  res = []
  retenue = 0
  # parcours parallèle des deux tableaux :
  for (chiffre1, chiffre2) in zip(nb1, nb2) :
    tmp = chiffre1 + chiffre2 + retenue
    retenue = tmp // 10                    # division euclidienne
    res.append(tmp % 10)             # ajout à la fin du tableau
  return res + [retenue]           # concaténation de 2 tableaux
```

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

correction : en montrant l'invariant :

« après i tours de boucle, `res` $\equiv n_1 + n_2$ *modulo* 10^i »

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & 1 & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

correction : en montrant l'invariant :

« après i tours de boucle, $\texttt{res} \equiv n_1 + n_2$ *modulo* 10^i »

complexité en temps : autant d'additions ***élémentaires*** (*i.e.* de chiffres) que de chiffres dans l'écriture décimale des entiers.

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$
\begin{array}{ccccc}
 & & {\scriptstyle 1} & {\scriptstyle 1} & \\
 & 1 & 3 & 5 & 7 \\
+ & 2 & 4 & 6 & 8 \\
\hline
 & 3 & 8 & 2 & 5
\end{array}
$$

correction : en montrant l'invariant :

« après i tours de boucle, $\texttt{res} \equiv n_1 + n_2$ *modulo* 10^i »

complexité en temps : autant d'additions ***élémentaires*** (*i.e.* de chiffres) que de chiffres dans l'écriture décimale des entiers.

$\Longrightarrow$ « complexité ***linéaire*** »

Exemple : les opérations arithmétiques

Addition de deux entiers :

$$\begin{array}{ccccc} & & 1 & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

correction : en montrant l'invariant :

« après i tours de boucle, $\texttt{res} \equiv n_1 + n_2$ *modulo* 10^i »

complexité en temps : autant d'additions ***élémentaires*** (*i.e.* de chiffres) que de chiffres dans l'écriture décimale des entiers.

$\Longrightarrow$ « complexité ***linéaire*** » – sous-entendu « en la taille ℓ des données », ici le nombre de chiffres décimaux de leur représentation : dire que n_1 et n_2 sont de taille (au plus) ℓ signifie que $n_1, n_2 \in O(10^\ell)$, ou encore que $\ell = 1 + \lfloor \max(\log_{10} n_1, \log_{10} n_2) \rfloor$

Exemple : les opérations arithmétiques

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :
  # nb1   tableau de chiffres représentant un entier n1
  # nb2   entier n2 représenté de manière usuelle (type int)
  res = [0] * len(nb1) # tableau de 0 de même longueur que nb1
  for i in range(1, nb2+1) : # de i=1 à i=nb2, donc nb2 tours
    res = addition(res, nb1)
  return res
```

correction : en montrant l'invariant : « après l'étape i, res $\equiv n_1 \times i$ »

Exemple : les opérations arithmétiques

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :
  # nb1   tableau de chiffres représentant un entier n1
  # nb2   entier n2 représenté de manière usuelle (type int)
  res = [0] * len(nb1) # tableau de 0 de même longueur que nb1
  for i in range(1, nb2+1) : # de i=1 à i=nb2, donc nb2 tours
    res = addition(res, nb1)
  return res
```

correction : en montrant l'invariant : « après l'étape i, res $\equiv n_1 \times i$ »

complexité en temps : n_2 additions...

Exemple : les opérations arithmétiques

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :
  # nb1   tableau de chiffres représentant un entier n1
  # nb2   entier n2 représenté de manière usuelle (type int)
  res = [0] * len(nb1) # tableau de 0 de même longueur que nb1
  for i in range(1, nb2+1) : # de i=1 à i=nb2, donc nb2 tours
    res = addition(res, nb1)
  return res
```

correction : en montrant l'invariant : « après l'étape i, res $\equiv n_1 \times i$ »

complexité en temps : n_2 additions... *de (grands) entiers*,

Exemple : les opérations arithmétiques

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :
  # nb1   tableau de chiffres représentant un entier n1
  # nb2   entier n2 représenté de manière usuelle (type int)
  res = [0] * len(nb1) # tableau de 0 de même longueur que nb1
  for i in range(1, nb2+1) : # de i=1 à i=nb2, donc nb2 tours
    res = addition(res, nb1)
  return res
```

correction : en montrant l'invariant : « après l'étape i, res $\equiv n_1 \times i$ »

complexité en temps : n_2 additions... ***de (grands) entiers***, ***chaque*** addition est de coût ***linéaire en la taille du résultat*** – donc en $\log(n_1 n_2) = \log(n_1) + \log(n_2)$

$\Longrightarrow$ complexité en $O(n_2 \times \log(n_1 n_2))$, soit $O(\ell \times 10^{\ell})$ si les deux entiers sont de taille ℓ

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication_par_un_chiffre(nb1, chiffre2) :
  # nb1   tableau de chiffres décimaux représentant un entier n1
  res = []
  retenue = 0
  for chiffre1 in nb1 :
    tmp = chiffre1 * chiffre2 + retenue
    retenue = tmp // 10     # division euclidienne
    res.append(tmp % 10)
  return res + [retenue]
```

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication_par_un_chiffre(nb1, chiffre2) :
  # nb1   tableau de chiffres décimaux représentant un entier n1
  res = []
  retenue = 0
  for chiffre1 in nb1 :
    tmp = chiffre1 * chiffre2 + retenue
    retenue = tmp // 10      # division euclidienne
    res.append(tmp % 10)
  return res + [retenue]
```

correction : en montrant l'invariant :

« après i tours de boucle, res $\equiv n_1 \times \mathrm{chiffre}_2 \mod 10^i$ »

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication_par_un_chiffre(nb1, chiffre2) :
  # nb1   tableau de chiffres décimaux représentant un entier n1
  res = []
  retenue = 0
  for chiffre1 in nb1 :
    tmp = chiffre1 * chiffre2 + retenue
    retenue = tmp // 10      # division euclidienne
    res.append(tmp % 10)
  return res + [retenue]
```

correction : en montrant l'invariant :

« après i tours de boucle, res $\equiv n_1 \times \mathtt{chiffre}_2 \mod 10^i$ »

complexité en temps : un tour de boucle par chiffre de n_1, de coût constant
$\Longrightarrow$ complexité en $O(\log(n_1))$, soit $O(\ell)$ si les nombres sont de taille ℓ

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication(nb1, nb2) :
  # nb1, nb2   tableaux de chiffres représentant des entiers n1, n2
  res = []
  # parcours du tableau nb2 avec itération sur les couples
  # (indice, contenu) de chaque case
  for (i, chiffre2) in enumerate(nb2) :
    tmp = multiplication_par_un_chiffre(nb1, chiffre2)
    res = addition(res, [0] * i + tmp)
  return res
```

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication(nb1, nb2) :
  # nb1, nb2   tableaux de chiffres représentant des entiers n1, n2
  res = []
  # parcours du tableau nb2 avec itération sur les couples
  # (indice, contenu) de chaque case
  for (i, chiffre2) in enumerate(nb2) :
    tmp = multiplication_par_un_chiffre(nb1, chiffre2)
    res = addition(res, [0] * i + tmp)
  return res
```

correction : en montrant l'invariant :

« après l'étape i, $\texttt{res} \equiv n_1 \times n_2 \mod 10^i$ »

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication(nb1, nb2) :
  # nb1, nb2   tableaux de chiffres représentant des entiers n1, n2
  res = []
  # parcours du tableau nb2 avec itération sur les couples
  # (indice, contenu) de chaque case
  for (i, chiffre2) in enumerate(nb2) :
    tmp = multiplication_par_un_chiffre(nb1, chiffre2)
    res = addition(res, [0] * i + tmp)
  return res
```

correction : en montrant l'invariant :

« après l'étape i, res $\equiv n_1 \times n_2 \mod 10^i$ »

complexité en temps : un tour de boucle par chiffre de n_2, chacun de complexité linéaire en la taille du résultat

$\Longrightarrow$ complexité en $O(\ell^2)$ si les nombres sont de taille ℓ

Exemple : les opérations arithmétiques

Multiplication de deux entiers (3) : la méthode du paysan russe

```
def multiplication_russe(nb1, nb2) :
# cette fois, les entiers sont vraiment des entiers
  res = 0
  while nb2 != 0 :
    if nb2%2 == 1 : res += nb1
    nb1 *= 2        # ou : nb1 << 1
    nb2 //= 2       # ou : nb2 >> 1
  return res
```

correction ?

complexité ?

Exemple : les opérations arithmétiques

Puissance (d'un entier par exemple) (1)

```
def puissance_naive(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  res = 1
  for i in range(nb2) :
    res *= nb1
  return res
```

Exemple : les opérations arithmétiques

Puissance (d'un entier par exemple) (1)

```
def puissance_naive(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  res = 1
  for i in range(nb2) :
    res *= nb1
  return res
```

correction : en montrant l'invariant :

« après i tours de boucle, res $= n_1^i$ »

Exemple : les opérations arithmétiques

Puissance (d'un entier par exemple) (1)

```
def puissance_naive(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  res = 1
  for i in range(nb2) :
    res *= nb1
  return res
```

correction : en montrant l'invariant :

« après i tours de boucle, $\texttt{res} = n_1^i$ »

complexité : cet algorithme effectue n_2 multiplications ***entre*** n_1 ***et un très grand entier*** : la taille de $n_1^{n_2}$ est $n_2 \log n_1$. Donc si n_1, n_2 sont de l'ordre de 10^ℓ, les dernières multiplications ont, par la méthode précédente, une complexité en $O(\ell^2 \times 10^\ell)$

$\Longrightarrow$ $O(\ell^2 \times 10^{2\ell})$ si n_1, n_2 entiers de taille ℓ

Exemple : les opérations arithmétiques

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2 // 2)
  carre = tmp * tmp
  if nb2 % 2 == 0 : return carre
  else : return nb1 * carre
```

correction ?

Exemple : les opérations arithmétiques

Puissance (2) : ***l'exponentiation binaire***

```
def puissance(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2 // 2)
  carre = tmp * tmp
  if nb2 % 2 == 0 : return carre
  else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

Exemple : les opérations arithmétiques

Puissance (2) : ***l'exponentiation binaire***

```
def puissance(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2 // 2)
  carre = tmp * tmp
  if nb2 % 2 == 0 : return carre
  else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

complexité ?

Exemple : les opérations arithmétiques

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2 // 2)
  carre = tmp * tmp
  if nb2 % 2 == 0 : return carre
  else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

complexité ? chaque appel récursif nécessite 1 ou 2 multiplications, et le nombre d'appels est égal à $\lfloor \log_2 n_2 \rfloor + 1$,
donc $O(\ell)$ multiplications si n_2 est un entier de taille ℓ

Exemple : les opérations arithmétiques

Puissance (2) : ***l'exponentiation binaire***

```
def puissance(nb1, nb2) :
  # nb1 un élément supportant la multiplication, nb2 un entier
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2 // 2)
  carre = tmp * tmp
  if nb2 % 2 == 0 : return carre
  else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

complexité ? chaque appel récursif nécessite 1 ou 2 multiplications, et le nombre d'appels est égal à $\lfloor \log_2 n_2 \rfloor + 1$,
donc $O(\ell)$ multiplications si n_2 est un entier de taille ℓ

(pour en dire plus, il faut connaître plus précisément la complexité de l'algorithme de multiplication utilisé)