

Études d'algorithmes classiques

S2-2-2 ALGO-II Algorithmes

Germain Gondor

LYCÉE CARNOT - DIJON, 2023 - 2024

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation

Sommaire

- 1 Indice du maximum
 - Algorithme
 - Invariant de boucle et correction
 - Terminaison
 - Complexité
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation

Algorithme

Algorithm 1 Indice du maximum

entrée: T un tableau de n valeurs

résultat: $iMax$ l'indice du maximum du tableau

indiceMax(T)

```
1:  $n = \text{taille}(T)$ 
2:  $iMax = 0$ 
3:  $\max \leftarrow T[0]$ 
4: pour  $i$  entre 1 et  $n-1$  faire
5:   si  $T[i] > \max$  alors
6:      $iMax \leftarrow i$ 
7:      $\max \leftarrow T[i]$ 
8:   fin si
9: fin pour
10: renvoi:  $iMax$ 
```

Invariant de boucle et correction

La variable `max` contient le maximum de $T[0:i]$ et `iMax` l'indice de sa première apparition.

Terminaison

Absence de boucle conditionnelle *Tant que* . Pas de récursivité. Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

Complexité

- au mieux $n + 1$ (max en première position)
- au pire $3.n$ (tableau strictement croissant).

Complexité

- au mieux $n + 1$ (max en première position)
- au pire $3.n$ (tableau strictement croissant).

Donc $C(n) = \Theta(n)$.

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
 - Algorithme
 - Invariant de boucle et correction
 - Terminaison
 - Complexité
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation

Algorithme

Algorithm 2 Recherche dans un tableau

entrée: T un tableau de n valeurs et x un élément (pas forcément du tableau)

résultat: -1 si $x \notin T$, l'indice de la première occurrence de x dans le tableau T

recherche(T, x)

- 1: $i \leftarrow 0$
- 2: $n \leftarrow \text{taille}(T)$
- 3: **tant que** $i < n$ et $T[i] \neq x$ **faire**
- 4: $i \leftarrow i + 1$
- 5: **fin tant que**
- 6: **si** $i = n$ **alors** $i \leftarrow -1$
- 7: **fin si**
- 8: **renvoi:** i

Invariant de boucle et correction

A chaque itération, x n'apparaît pas dans les $i + 1$ premières cases du tableau.

Terminaison

La suite des valeurs prises par i , entier, est strictement croissante et majorée par n .

Complexité

- au mieux 2 (tableau vide) ou 3 (x en première position)
- au pire $2.n + 2$ (x n'apparaît pas).

Complexité

- au mieux 2 (tableau vide) ou 3 (x en première position)
- au pire $2.n + 2$ (x n'apparaît pas).

Donc $C(n) = O(n)$.

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type**
 - Rappel
 - Algorithme
 - Invariant de boucle et correction
 - Terminaison
 - Complexité
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation

Rappel

Moyenne $Moy(T) = \frac{1}{n} \sum_{i=1}^n T[i]$

Variance $Var(T) = \left(\frac{1}{n} \sum_{i=1}^n T[i]^2 \right) - Moy(T)^2$

Ecart type $\sigma(T) = \sqrt{Var(T)}$

Algorithmme

Algorithm 3 Moyenne et variance

entrée: T un tableau de n valeurs

résultats: Moy , la moyenne de T ; Var , la variance de T ; σ , l'écart type de T

MoyVar(T)

- 1: $n \leftarrow \text{taille}(T)$
- 2: $\text{som} \leftarrow 0$
- 3: $\text{somCar} \leftarrow 0$.
- 4: **pour** i entre 0 et $n-1$ **faire**
- 5: $\text{som} \leftarrow \text{som} + T[i]$
- 6: $\text{somCar} \leftarrow \text{somCar} + T[i]^2$
- 7: **fin pour**
- 8: $\text{moy} \leftarrow \text{som} / n$
- 9: $\text{Var} \leftarrow \text{somCar} / n - \text{moy}^2$
- 10: **renvoi:** $\text{moy}, \text{Var}, \text{racine}(\text{Var})$

Invariant de boucle et correction

A chaque itération i :

$$som = \sum_{j=0}^{i-1} T[j]$$

$$somCar = \sum_{j=0}^{i-1} T[j]^2$$

Terminaison

Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

Complexité

Dans la boucle `Pour`, il faut :

Complexité

Dans la boucle `Pour`, il faut :

- ajouter $T[i]$ à somme : $C+ = 1$

Complexité

Dans la boucle `Pour`, il faut :

- ajouter $T[i]$ à somme : $C+ = 1$
- calculer le carré de $T[i]$: $C+ = 1$

Complexité

Dans la boucle `Pour`, il faut :

- ajouter $T[i]$ à somme : $C+ = 1$
- calculer le carré de $T[i]$: $C+ = 1$
- l'ajouter à `somCar` : $C+ = 1$

Complexité

Dans la boucle `Pour`, il faut :

- ajouter $T[i]$ à somme : $C+ = 1$
- calculer le carré de $T[i]$: $C+ = 1$
- l'ajouter à `somCar` : $C+ = 1$

A chaque itération de la boucle, le coût C est donc de 3. Avec les différentes affectations en début et en fin de programme, on obtient un coût total $C(n)$:

Complexité

Dans la boucle `Pour`, il faut :

- ajouter `T[i]` à `somme` : $C+ = 1$
- calculer le carré de `T[i]` : $C+ = 1$
- l'ajouter à `somCar` : $C+ = 1$

A chaque itération de la boucle, le coût C est donc de 3. Avec les différentes affectations en début et en fin de programme, on obtient un coût total $C(n)$:

$$C(n) = 3.n + 9 = \Theta(n)$$

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié**
 - Algorithme
 - Invariant de boucle et correction
 - Terminaison
 - Correction
 - Complexité
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation

Algorithme

Algorithm 4 Recherche dichotomique dans un tableau trié

entrée: T un tableau de n valeurs et x un élément (pas forcément du tableau)

résultat: -1 si $x \notin T$, l'indice d'un x dans le tableau T

```
1: DichoSearch( $T, x$ )
2:  $n \leftarrow \text{taille}(T)$ 
3:  $\text{min}, \text{max}, \text{mil} \leftarrow 0, n-1, (\text{min} + \text{max}) // 2$ 
4: tant que  $\text{min} < \text{max}$  et  $T[\text{mil}] \neq x$  faire
5:   si  $T[\text{mil}] < x$  alors
6:      $\text{min} \leftarrow \text{mil} + 1$ 
7:   sinon
8:      $\text{max} \leftarrow \text{mil} - 1$ 
9:   fin si
10:   $\text{mil} \leftarrow (\text{min} + \text{max}) // 2$ 
11: fin tant que
12: si  $T[\text{mil}] = x$  alors
13:    $\text{rep} \leftarrow -1$ 
14: sinon
15:    $\text{rep} \leftarrow \text{mil}$ 
16: fin si
17: renvoi:  $\text{rep}$ 
```

Invariant de boucle et correction

Proposition $\mathcal{P}(k)$: *A la fin de l'étape k (donc si x n'a pas été trouvé)*

$$\max - \min < \frac{n}{2^k}$$

et si x est dans le tableau T alors

$$T[\min] \leq x \leq T[\max]$$

Démonstration par récurrence

- A la fin de l'étape 0, donc juste avant la boucle, si x est dans le tableau (trié), on a bien

$$T[0] = T[\min] \leq x \leq T[\max] = T[n-1]$$

$$\max - \min = n - 1 < \frac{n}{2^0} = n$$

- Si $\mathcal{P}(k)$ est vraie à la fin de l'étape k , appelons a la valeur de min , b celle de max tels que $b - a < \frac{n}{2^k}$. Alors mil devient :

$$mil = \left\lfloor \frac{min + max}{2} \right\rfloor = \left\lfloor \frac{a + b}{2} \right\rfloor$$

Si $T[mil] \neq x$, on ne sort pas de la boucle, et

- soit $T[\text{mil}] < x$, alors min prend la valeur $\text{mil} + 1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1$ et max reste à b .

Si x est dans le tableau (trié), on a alors bien

$T[\text{min}] \leq x \leq T[\text{max}]$, et

$$\text{max} - \text{min} = b - \left\lfloor \frac{a+b}{2} \right\rfloor - 1 < b - \left\lfloor \frac{a+b}{2} \right\rfloor = \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

- soit $T[mil] < x$, alors min prend la valeur $mil + 1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1$ et max reste à b .

Si x est dans le tableau (trié), on a alors bien

$T[min] \leq x \leq T[max]$, et

$$max - min = b - \left\lfloor \frac{a+b}{2} \right\rfloor - 1 < b - \left\lfloor \frac{a+b}{2} \right\rfloor = \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

- soit $T[mil] > x$, alors max prend la valeur $mil - 1 = \left\lfloor \frac{a+b}{2} \right\rfloor - 1$ et min reste à a .

Si x est dans le tableau (trié), on a alors bien

$T[min] \leq x \leq T[max]$, et

$$max - min = \left\lfloor \frac{a+b}{2} \right\rfloor - 1 - a < \left\lfloor \frac{a+b}{2} \right\rfloor - a = \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

- soit $T[mil] < x$, alors min prend la valeur $mil + 1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1$ et max reste à b .

Si x est dans le tableau (trié), on a alors bien

$T[min] \leq x \leq T[max]$, et

$$max - min = b - \left\lfloor \frac{a+b}{2} \right\rfloor - 1 < b - \left\lfloor \frac{a+b}{2} \right\rfloor = \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

- soit $T[mil] > x$, alors max prend la valeur $mil - 1 = \left\lfloor \frac{a+b}{2} \right\rfloor - 1$ et min reste à a .

Si x est dans le tableau (trié), on a alors bien

$T[min] \leq x \leq T[max]$, et

$$max - min = \left\lfloor \frac{a+b}{2} \right\rfloor - 1 - a < \left\lfloor \frac{a+b}{2} \right\rfloor - a = \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

d'où la récurrence. CQFD

Terminaison

Comme la différence $max - min = Err$ est à valeur dans $\mathbb{N}$ ($Err \in \mathbb{N}$) et que $max - min < \frac{n}{2^k}$ donc si k assez grand $max - min = 0$.

Correction

Soit on tombe sur x à un moment, soit on arrive à $min = max$, et si x est dans le tableau, $T[min] \leq x \leq T[max]$. Le résultat renvoyé est bien le bon.

Complexité

A chaque étape de la boucle Tant que , il se produit :

- deux tests : $C+ = 2$
- une affectation (cas **si** ou **sinon**) : $C+ = 1$
- une somme + une division+une affectation : $C+ = 3$

Or x se loge dans un intervalle $max - min < \frac{n}{2^k}$ et on s'arrête au plus tard quand $min = max$. Ainsi, au plus, on a k itérations tel que

$$\frac{n}{2^k} < 1 \Rightarrow 2^k > n \Rightarrow k \cdot \ln(2) > \ln(n) \Rightarrow k > \log_2(n)$$

donc au plus $\lfloor \log_2(n) \rfloor + 1$ itérations. Ainsi, le coût pour la boucle est $C(n) = 4 \cdot (\log_2(n) + 1) = O(\ln(n))$.

Recherche dichotomique dans un tableau trié de flottants

Le cas de la recherche dichotomique dans un tableau trié de flottants ressemble beaucoup. Il suffit d'ajouter une précision $\epsilon > 0$ en argument et de remplacer $T[mil] = x$ par $|T[mil] - x| < \epsilon$.

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères**
 - Principe
 - Algorithme
 - Algorithme
 - Invariant de boucle et correction
 - Complexité
- 6 Complément : l'exponentiation

Principe

Pour écrire un algorithme *naïf* de recherche d'un mot dans une chaîne de caractères (`ChercheMot(mot, chaîne)`), on utilise une fonction `TestMot(mot, chaîne, i)` qui teste la coïncidence entre les lettres de `mot` à une position `i` de `chaîne` jusqu'à trouver une lettre différente dans la successions des lettres. Il renvoie alors `vrai` si toutes les lettres de `mot` se succèdent dans `chaîne` à la position `i`; `faux` sinon.

La fonction `ChercheMot(mot, chaîne)` balaye les lettres chaînes en appelant pour chaque lettre la fonction `TestMot` jusqu'à ce que cette dernière renvoie `vrai` ou que le nombre de lettres de `chaîne` restant soit inférieur à la taille de `mot`.

Algorithme

Algorithm 5 Teste la présence d'un mot à partir d'une position dans une chaîne de caractères.

entrée: *mot* et *chaine* deux chaînes de caractères, *i*, un indice

résultat: *vrai* si *mot* correspond aux premiers caractères de *chaine* à partir de l'indice *i*; *faux* sinon

- 1: TestMot(*mot*,*chaine*,*i*)
 - 2: $j, m \leftarrow 0, \text{taille}(\text{mot})$
 - 3: **tant que** $j < m$ et $\text{mot}[j] = \text{chaine}[i+j]$ **faire**
 - 4: $j \leftarrow j+1$
 - 5: **fin tant que**
 - 6: **renvoi:** $j == m$
-

Algorithme

Algorithm 6 Recherche d'un mot dans une chaîne de caractères.

entrée: *mot* et *chaine* deux chaînes de caractères

résultat: l'indice de la première occurrence de *mot* dans *chaine* ;
sinon -1

ChercheMot(*mot*,*chaine*)

- 1: $i, n, m, \text{pos}, \text{pastrouv} \leftarrow 0, \text{taille}(\text{chaine}), \text{taille}(\text{mot}), 0, \text{vrai}$
- 2: **tant que** $i \leq n - m$ et **pastrouv** **faire**
- 3: $\text{pastrouv} \leftarrow \text{non}(\text{TestMot}(\text{mot}, \text{chaine}, i))$
- 4: $i \leftarrow i + 1$
- 5: **fin tant que**
- 6: **si** **pastrouv** **alors**
- 7: $i \leftarrow -1$
- 8: **fin si**
- 9: **renvoi:** i

Invariant de boucle et correction

- pour `Testmot` : les j premières lettres de `mot` se suivent dans chaîne
- pour `ChercheMot` : `mot` ne se trouve pas dans les i premières positions de chaîne

Complexité

En comparaison :

- pour `Testmot` : la boucle est parcourue au plus m fois.
- pour `ChercheMot` : la boucle est parcourue au mieux 1 fois ; au pire, $n - m$ fois. Avec l'appel de `Testmot`
$$C(n) = m.(n - m + 1) = O(m.(n - m))$$

Sommaire

- 1 Indice du maximum
- 2 Recherche dans un tableau
- 3 Moyenne, variance et écart type
- 4 Recherche dichotomique dans un tableau trié
- 5 Recherche d'un mot dans une chaîne de caractères
- 6 Complément : l'exponentiation**
 - Exponentiation « naïve »
 - Exponentiation rapide itérative
 - Exponentiation rapide récursive

Exponentiation « naïve »

Principe

Multiplier x n fois par lui même.

Exponentiation « naïve »

Algorithme

Algorithm 7 Exponentiation « naïve »

entrée: n un entier positif et x un nombre réel

résultat: un nombre réel $r = x^n$

Exponaive(x, n)

```
1: si  $n == 0$  alors
2:   renvoi: 1
3: sinon
4:    $r \leftarrow x$ 
5:   pour  $i$  de 2 à  $n$  faire
6:      $r \leftarrow x.r$ 
7:   fin pour
8: fin si
9: renvoi:  $r$ 
```

Exponentiation « naïve »

Terminaison

Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

Exponentiation « naïve »

Invariant de boucle et correction

A chaque itération i , $r = x^i$.

Exponentiation « naïve »

Complexité

A chaque itération, il faut faire une multiplication $C(n) = \Theta(n)$.

Exponentiation rapide itérative

Algorithme

Algorithm 8 Exponentiation rapide itérative

entrée: n un entier positif et x un nombre réel

résultat: un nombre réel $r = x^n$

ExpoRapide(x, n)

```
1: si  $n == 0$  alors
2:   renvoi: 1
3: sinon
4:    $r \leftarrow 1$ 
5:   tant que  $n > 0$  faire
6:     si  $n \bmod 2 == 1$  alors
7:        $r \leftarrow r \cdot x$ 
8:     fin si
9:      $x \leftarrow x \cdot x$ 
10:     $n \leftarrow n // 2$ 
11:   fin tant que
12:   renvoi:  $r$ 
13: fin si
```

Exponentiation rapide itérative

Principe

Se servir des résultats intermédiaires pour accélérer le processus de calcul de x^n en décomposant n en puissance de 2.

EXEMPLE : calculer x^5 . On calcule x^2 puis x^4 et finalement x^5 .

Exponentiation rapide itérative

Terminaison

La boucle conditionnelle s'arrête dès que n cesse d'être strictement positif. Or n est une suite d'entier (division euclidienne) décroissante puisque n est divisé par 2 à chaque itération.

La suite est strictement décroissante et minorée ; l'algorithme termine.

Exponentiation rapide itérative

Propriété

on appelle r_i , x_i et n_i les valeurs de r , x et n après l'itération i dans la boucle `while`. Avec $i = 0$, on a $r_0 = 1$, $x_0 = x$ et $n_0 = n$.

Proposition $\mathcal{P}(i)$: *à la fin de l'étape i , $x^n = r_i \cdot x_i^{n_i}$*

DÉMONSTRATION :

- au rang 0 : montrons $\mathcal{P}(0)$

$$r_0.x_0^{n_0} = 1.x^n = x^n \text{ la propriété est donc vraie au rang 0}$$

- au rang i : montre que $\mathcal{P}(i-1) \Rightarrow \mathcal{P}(i)$

Supposons $\mathcal{P}(i-1)$ vraie : $x^n = r_{i-1}.x_{i-1}^{n_{i-1}}$, alors :

- soit $n_{i-1} \pmod{2} = 1$ et $r_i = r_{i-1}.x$, $x_i = x_{i-1}^2$ et $n_{i-1} = 2.n_i + 1$

$$\text{d'où } r_i.x_i^{n_i} = r_{i-1}.x.x_{i-1}^{2.n_i} = \frac{x^n}{x_{i-1}^{n_{i-1}}} . x_{i-1}^{2.n_i+1} = x^n$$

- soit $n_{i-1} \pmod{2} = 0$ et $r_i = r_{i-1}$, $x_i = x_{i-1}^2$ et $n_{i-1} = 2.n_i$ d'où

$$r_i.x_i^{n_i} = r_{i-1}.x_{i-1}^{2.n_i} = \frac{x^n}{x_{i-1}^{n_{i-1}}} . x_{i-1}^{2.n_i} = x^n$$

ainsi $\mathcal{P}(i-1) \Rightarrow \mathcal{P}(i)$

Exponentiation rapide itérative

Invariant de boucle et correction

A chaque itération i , $\mathcal{P}(i)$ est vraie. Or pour $k \in \mathbb{N}$ tel que $n_k = 0$, $\mathcal{P}(k)$ étant vraie $x^n = r_k \cdot x_k^{n_k} = r_k$. L'algorithme retournant r_k puisque $n_k = 0$, la valeur renvoyée est donc bien x^n

Exponentiation rapide itérative

Complexité

A chaque itération i , $n_i = \left\lfloor \frac{n_{i-1}}{2} \right\rfloor \leq \frac{n_{i-1}}{2} \leq \frac{n}{2^i}$. Ainsi, si m est le nombre d'itérations permettant de calculer x^n alors :

$$1 = n_{m-1} = \frac{n}{2^{m-1}} \Rightarrow (m-1) \cdot \ln(2) \leq \ln(n) \Rightarrow C(m) = O(\ln(m))$$

Exponentiation rapide récursive

Principe

Se servir des résultats intermédiaires pour accélérer le processus de calcul de x^n en décomposant n en puissance de 2.

Exponentiation rapide récursive

Algorithme

Algorithm 9 Exponentiation rapide récursive

entrée: n un entier positif et x un nombre réel

résultat: un nombre réel $r = x^n$

ExpoRec(x, n)

- 1: **si** $n == 0$ **alors**
 - 2: **renvoi:** 1
 - 3: **sinon**
 - 4: **si** $n \bmod 2 == 0$ **alors**
 - 5: **renvoi:** ExpoRec($x.x, n/2$)
 - 6: **sinon**
 - 7: **renvoi:** $x.$ ExpoRec($x.x, (n-1)/2$)
 - 8: **fin si**
 - 9: **fin si**
-

Exponentiation rapide récursive

Terminaison

n décrit une suite d'entiers strictement décroissante et minorée.
L'algorithme se termine.

Exponentiation rapide récursive

Complexité

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
 $C(n) = C(n/2) + 3$

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
$$C(n) = C(n/2) + 3$$

On suppose l'algorithme logarithmique.

L'algorithme est donc bien de complexité logarithmique.

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
$$C(n) = C(n/2) + 3$$

On suppose l'algorithme logarithmique.

- on pose alors : $C(n) = a \cdot \log_2(n) + c$

L'algorithme est donc bien de complexité logarithmique.

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
$$C(n) = C(n/2) + 3$$

On suppose l'algorithme logarithmique.

- on pose alors : $C(n) = a \cdot \log_2(n) + c$

$$C(n/2) = a \cdot \log_2(n/2) + c = a \cdot \log_2(n) - a \cdot \log_2(2) + c$$

- $$= a \cdot \log_2(n) - a + c$$

$$C(n) = a \cdot \log_2(n) - a + c + 3 = a \cdot \log_2(n) + c$$

L'algorithme est donc bien de complexité logarithmique.

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
 $C(n) = C(n/2) + 3$

On suppose l'algorithme logarithmique.

- on pose alors : $C(n) = a \cdot \log_2(n) + c$

$$C(n/2) = a \cdot \log_2(n/2) + c = a \cdot \log_2(n) - a \cdot \log_2(2) + c$$

- $= a \cdot \log_2(n) - a + c$

$$C(n) = a \cdot \log_2(n) - a + c + 3 = a \cdot \log_2(n) + c$$

- on a donc $a = 3$

L'algorithme est donc bien de complexité logarithmique.

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
 $C(n) = C(n/2) + 3$

On suppose l'algorithme logarithmique.

- on pose alors : $C(n) = a \cdot \log_2(n) + c$

$$C(n/2) = a \cdot \log_2(n/2) + c = a \cdot \log_2(n) - a \cdot \log_2(2) + c$$

- $= a \cdot \log_2(n) - a + c$

$$C(n) = a \cdot \log_2(n) - a + c + 3 = a \cdot \log_2(n) + c$$

- on a donc $a = 3$
- or $3 = C(1) = a \cdot \log_2(1) + c = c$ donc $c = 3$

L'algorithme est donc bien de complexité logarithmique.

Exponentiation rapide récursive

Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division :
 $C(n) = C(n/2) + 3$

On suppose l'algorithme logarithmique.

- on pose alors : $C(n) = a \cdot \log_2(n) + c$

$$C(n/2) = a \cdot \log_2(n/2) + c = a \cdot \log_2(n) - a \cdot \log_2(2) + c$$

- $= a \cdot \log_2(n) - a + c$

$$C(n) = a \cdot \log_2(n) - a + c + 3 = a \cdot \log_2(n) + c$$

- on a donc $a = 3$
- or $3 = C(1) = a \cdot \log_2(1) + c = c$ donc $c = 3$
- $C(n) = 3 \cdot \log_2(n) + 3 = \Theta(\ln(n))$

L'algorithme est donc bien de complexité logarithmique.