Calculs de complexité d'une multiplication de 2 entiers de n chiffres

Il existe différentes méthodes pour calculer la multiplication de 2 entiers. Nous vous proposons de mettre en place celle utilisant le paradigme *Diviser-pour-Régner* (décomposition) suivant.

Soient X et Y deux entiers à n chiffres. Pour simplifier sans perdre de généralité, nous supposons que n est une puissance de 2. On peut alors décomposer X et Y comme suit :
$X = a10^{n/2} + b$ et $Y = c10^{n/2} + d$.

La **multiplication XY** peut alors être calculée récursivement comme suit :

```
Si X et Y sont des nombres à un chiffre alors retourner XY ;
Sinon
    Partitionner X en $a10^{n/2} + b$; Partitionner Y en $c10^{n/2} + d$;
    retourner $XY = ac10^n + (ad+bc)10^{n/2} + bd$;
Fsi.
```

Remarque importante : la multiplication par les puissances de 10, i.e. 10^n ou $10^{n/2}$, n'est pas récursive.

Questions :

2.1. Donnez l'arborescence complète de décomposition du calcul récursif de 1234x2139. Pensez à vérifier que le résultat obtenu par ce calcul récursif corresponde bien à celui obtenu par le calcul classique que vous connaissez.

2.2. Donnez la complexité temporelle en O de T(n) de cette multiplication récursive. Pensez d'abord à donner la formule récurrente de T(n) avant d'effectuer le calcul explicite de T(n).

Or maintenant, on remarque que si l'on considère $ad+bc=(a+b)(c+d)-ac-bd$, alors la partie inductive de l'algorithme s'écrit $XY = ac10^n + ((a+b)(c+d)-ac-bd)10^{n/2} + bd$.

Remarque importante : les multiplications ac et bd apparaissent 2 fois dans cette seconde formulation.

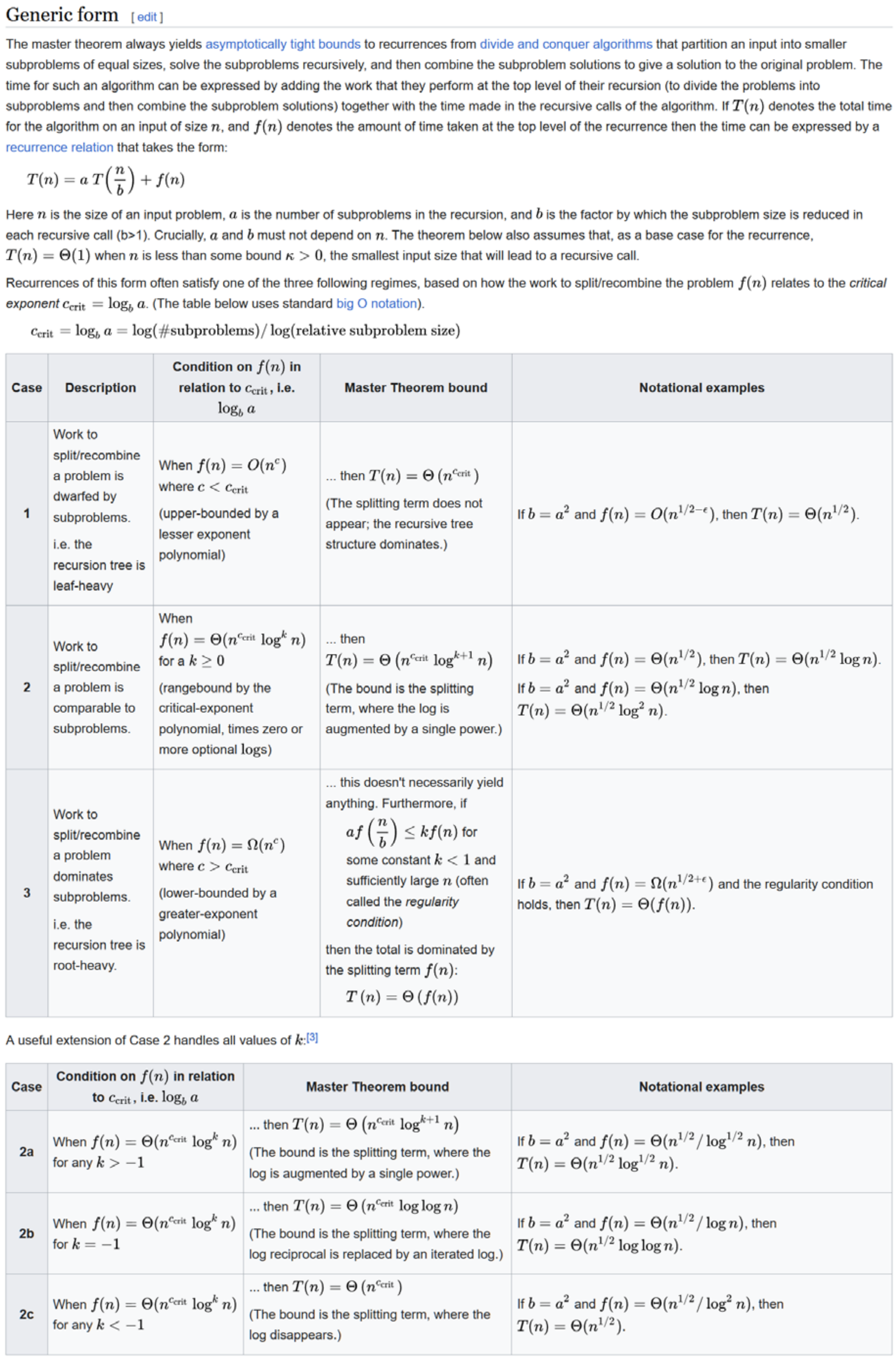
Questions :

2.3. Donnez l'arborescence complète pour cette nouvelle décomposition du calcul récursif de 1234x2139.

2.4. Donnez la complexité temporelle en O de T(n) de cette multiplication récursive. Pensez d'abord à donner la formule récurrente de T(n) avant d'effectuer le calcul explicite de T(n).

2.5. Que déduisez-vous en termes de performance par rapport à la complexité temporelle calculée dans la question 2.2. ?

Soucre : `https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)`



Generic form [edit]

The master theorem always yields asymptotically tight bounds to recurrences from divide and conquer algorithms that partition an input into smaller subproblems of equal sizes, solve the subproblems recursively, and then combine the subproblem solutions to give a solution to the original problem. The time for such an algorithm can be expressed by adding the work that they perform at the top level of their recursion (to divide the problems into subproblems and then combine the subproblem solutions) together with the time made in the recursive calls of the algorithm. If $T(n)$ denotes the total time for the algorithm on an input of size n, and $f(n)$ denotes the amount of time taken at the top level of the recurrence then the time can be expressed by a recurrence relation that takes the form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

Here n is the size of an input problem, a is the number of subproblems in the recursion, and b is the factor by which the subproblem size is reduced in each recursive call (b>1). Crucially, a and b must not depend on n. The theorem below also assumes that, as a base case for the recurrence, $T(n) = \Theta(1)$ when n is less than some bound $\kappa > 0$, the smallest input size that will lead to a recursive call.

Recurrences of this form often satisfy one of the three following regimes, based on how the work to split/recombine the problem $f(n)$ relates to the *critical exponent* $c_{\text{crit}} = \log_b a$. (The table below uses standard big O notation).

$$c_{\text{crit}} = \log_b a = \log(\#\text{subproblems})/\log(\text{relative subproblem size})$$

Case	Description	Condition on $f(n)$ in relation to c_{crit}, i.e. $\log_b a$	Master Theorem bound	Notational examples
1	Work to split/recombine a problem is dwarfed by subproblems. i.e. the recursion tree is leaf-heavy	When $f(n) = O(n^c)$ where $c < c_{\text{crit}}$ (upper-bounded by a lesser exponent polynomial)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$ (The splitting term does not appear; the recursive tree structure dominates.)	If $b = a^2$ and $f(n) = O(n^{1/2-\epsilon})$, then $T(n) = \Theta(n^{1/2})$.
2	Work to split/recombine a problem is comparable to subproblems.	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for a $k \geq 0$ (rangebound by the critical-exponent polynomial, times zero or more optional logs)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2})$, then $T(n) = \Theta(n^{1/2} \log n)$. If $b = a^2$ and $f(n) = \Theta(n^{1/2} \log n)$, then $T(n) = \Theta(n^{1/2} \log^2 n)$.
3	Work to split/recombine a problem dominates subproblems. i.e. the recursion tree is root-heavy.	When $f(n) = \Omega(n^c)$ where $c > c_{\text{crit}}$ (lower-bounded by a greater-exponent polynomial)	... this doesn't necessarily yield anything. Furthermore, if $af\left(\frac{n}{b}\right) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n (often called the *regularity condition*) then the total is dominated by the splitting term $f(n)$: $T\left(n\right) = \Theta\left(f(n)\right)$	If $b = a^2$ and $f(n) = \Omega(n^{1/2+\epsilon})$ and the regularity condition holds, then $T(n) = \Theta(f(n))$.

A useful extension of Case 2 handles all values of k:[3]

Case	Condition on $f(n)$ in relation to c_{crit}, i.e. $\log_b a$	Master Theorem bound	Notational examples
2a	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for any $k > -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log^{1/2} n)$, then $T(n) = \Theta(n^{1/2} \log^{1/2} n)$.
2b	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for $k = -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log \log n\right)$ (The bound is the splitting term, where the log reciprocal is replaced by an iterated log.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log n)$, then $T(n) = \Theta(n^{1/2} \log \log n)$.
2c	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for any $k < -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$ (The bound is the splitting term, where the log disappears.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log^2 n)$, then $T(n) = \Theta(n^{1/2})$.