Diviser pour régner

Dans ce TP, on s'intéresse à la stratégie *diviser pour régner* dont le but est d'améliorer la complexité d'un algorithme. Le principe est le suivant :
- on divise un problème de taille n en deux sous-problèmes de tailles équivalentes $\lfloor n/2 \rfloor$ et $\lceil n/2 \rceil$;
- on résoud de manière récursive le problème sur une ou plusieurs de ces parties ;
- on combine les résultats afin d'obtenir le résultat du problème initial.

On utilise un résultat de complexité qui sera démontré en cours :
Théorème: On note $T(n)$ le coût maximal pour traiter une donnée de taille n. Si $T(n)$ vérifie la relation de récurrence $T(n) = 2T(n/2) + F(n)$ et s'il existe $p \geq 1$ tel que $F(n) = O(n^p)$, alors on a

$$T(n) = \begin{cases} O(n \log n) & \text{si } p = 1 \\ O(n^p) & \text{sinon.} \end{cases}$$

Exponentiation rapide
Ecrire une fonction `puissance` qui réalise l'exponentiation rapide d'un entier avec une complexité logarithmique.
De même écrire une fonction réalisant l'exponentiation rapide d'une matrice $2{\times}2$ à coefficients entiers.

Maximum
Ecrire une fonction `maximum` selon la méthode diviser pour régner qui détermine le maximum d'un tableau non trié. On pourra utiliser la fonction `Array.sub` qui s'appelle de la manière suivante : `Array.sub v deb long` renvoie un vecteur contenant les éléments du vecteur `v` compris entre `deb` et `deb + long - 1`.
Quelle est la complexité de la fonction `maximum` ? Comment l'améliorer ?

Racine carrée
Ecrire une fonction permettant de calculer $\lfloor \sqrt{n} \rfloor$ pour tout entier $n \in \mathbb{N}$ avec un coût logarithmique. Pour cela, on pourra introduire une fonction auxiliaire `cherche i j` reprenant le principe de la recherche dichotomique telle que i et j vérifient $i^2 \leq n \leq j^2$.

Nombre d'inversions dans un tableau d'entiers
Etant donnée une suite finie d'entiers $x = (x_1, \ldots, x_n)$, on appelle *inversion* de x tout couple (i, j) tel que $i < j$ et $x_i > x_j$. Par exemple, $(2, 3, 1, 5, 4)$ possède 3 inversions : les couples $(1, 3)$, $(1, 2)$ et $(4, 5)$. On s'intéresse au calcul du nombre d'inversions de x.
1) Rédiger l'algorithme naïf et étudier sa complexité. On représentera les suites finies d'entiers par le type `Array vect`.
2) Pour améliorer la complexité de l'algorithme on adopte une stratégie diviser pour régner. L'algorithme se déroule selon les étapes suivantes :
- On sépare le tableau en deux tableaux de taille sensiblement égale.
- On calcule récursivement le nombre d'inversions dans chaque demi-tableau.
- Il reste alors à déterminer les inversions à cheval sur les deux parties.

a) Ecrire une fonction `acheval` de complexité linéaire prenant en paramètre deux **listes triées** et qui renvoie le nombre d'inversions entre les deux listes.
b) Ecrire une fonction `fusion` de complexité linéaire prenant en argument deux listes triées et qui renvoie la liste triée des éléments des deux listes.
c) En déduire la fonction `inversion` et étudier sa complexité.