# TD d'Algorithmique et structures de données
# Diviser pour régner

Équipe pédagogique Algo SD

## 1 Elément majoritaire

Un tableau possède un élément majoritaire si plus de la moitié (strictement) de ses entrées sont identiques. Etant donné un tableau $A$ à $n$ éléments, on se propose de concevoir un algorithme efficace qui permet de déterminer s'il possède un élément majoritaire, et dans l'affirmative de l'identifier. Les éléments du tableau ne sont pas forcément comparables comme des entiers ou des caractères alphabétiques et il n'est pas possible d'effectuer des comparaisons de la forme A[i] > A[j] (on peut penser par exemple à des images). Les éléments ne sont pas hashables. Par contre, on peut répondre à des questions de la forme « A[i] = A[j] ? » en temps constant.

On peut commencer à remarquer que s'il existe, un élément majoritaire est nécessairement unique...

1. Écrire un algorithme qui calcule le nombre d'occurrences d'un élément donné dans un tableau.
   En déduire un algorithme simple pour trouver l'élément majoritaire.
   Evaluer son coût.

### 1.1 Diviser pour régner

Une approche un peu plus sophistiquée consiste à utiliser le paradigme "diviser pour régner" : couper le tableau $A$ en deux tableaux $A1$ et $A2$ de même taille (dans un premier temps, on pourra supposer que la taille du tableau initial est une puissance de 2).

S'il existe un élément majoritaire dans $A$ alors il est majoritaire dans au moins un des deux sous-tableaux.

On en déduit un algorithme récursif qui calcule les éléments majoritaires dans chacun des deux sous-tableaux (s'ils existent) et leurs nombres d'occurrences.

1. Détailler cet algorithme.
2. Quel est son coût ? On donnera l'équation de récurrence et on explicitera comment la résoudre.

### 1.2 Variante

On se propose maintenant d'essayer une autre solution en groupant les éléments par paires (deux par deux). On examine alors les $\frac{n}{2}$ paires ainsi formées. Si les deux éléments d'une paire sont différents, on les élimine. S'ils sont identiques, on n'en retient qu'un seul.

1. Supposons une entrée de taille paire. Montrer qu'après une étape de cette procédure simple il ne reste au plus que n/2 éléments. Montrer que si un élément est majoritaire dans le tableau initial, alors il l'est également dans le tableau obtenu après une étape.
2. Conclure en donnant un algorithme complet et calculer son coût en pire cas.

## 2 Multiplication des grands entiers – Algorithme de Karatsuba

On considère la représentation en binaire de deux entiers (longs) $A$ et $B$, de taille $n = 2^k$ pour un entier positif $k$.

$$A = \{a_n a_{n-1} \dots a_1\} \quad \text{et} \quad B = \{b_n b_{n-1} \dots b_1\}$$

L'objectif ici est de calculer la représentation binaire de $A \times B$. On rappelle que l'algorithme simple (celui que l'on a appris à l'école primaire) est basé sur le calcul de $n$ produits $a_n a_{n-1} \dots a_1 a_0$ par $b_i$ pour $i = 1, \dots, n$. Sa complexité est en $\mathcal{O}(n^2)$.

## 2.1 Algorithme naïf

1. Etant données les représentations binaires de $A$ and $B$, on demande d'écrire un algorithme diviser-pour-régner pour multiplier ces deux entiers.
2. Ecrire l'équation de récurrence qui calcule le coût de cet algorithme.
3. En déduire le coût asymptotique et le comparer à l'algorithme classique.

## 2.2 Karatsuba

On propose ici une amélioration de la méthode précédente. Pour cela, on va diminuer le nombre de sous-problèmes, c'est-à-dire, le nombre d'appels à des multiplications d'entiers de $n/2$-bits.

1. Montrer que $(A_1 + A_2) \cdot (B_1 + B_2) = A_1 B_1 + A_2 B_2 + (A_1 B_2 + A_2 B_1)$. En déduire un algorithme diviser-pour-régner.
2. Calculer le coût asymptotique de Karatsuba.