TP n° 5 : Diviser pour régner

Diviser pour (mieux) régner

— Expression française, d'origine latine sous la forme *divide ut regnes* ou *divide et impera*, souvent attribuée au Sénat romain ou à Philippe de Macédoine, et prônée par Nicolas de Machiavel.

1 La stratégie *diviser pour régner*

En politique et en sociologie, *diviser pour régner* est une stratégie visant à semer la discorde et à opposer pour affaiblir et ainsi régner sur une population qui, si elle était unie, aurait les moyens de faire tomber le pouvoir en question.

En informatique, *diviser pour régner* est une méthode de conception d'algorithmes réduisant récursivement un problème en un ou plusieurs sous-problèmes du même type.

Cette stratégie consiste généralement en trois étapes :

- Découper le problème en sous-problèmes [1] (*diviser*) ;
- Résoudre récursivement les sous-problèmes (*régner*) ;
- Combiner les solutions des sous-problèmes.

Exemple 1 (*Recherche dichotomique pour trouver le zéro d'une fonction*)

Soit $f : [a,b] \to \mathbb{R}$ une fonction continue avec $f(a) < 0$ et $f(b) > 0$. On cherche à déterminer un point $x \in [a,b]$ tel que $f(x) = 0$.

Vous avez étudié, ou allez bientôt le faire, ce problème en informatique en tronc commun. Une idée est de découper l'intervalle $[a,b]$ en deux intervalles plus petits, par exemple $[a, \frac{a+b}{2}]$ et $[\frac{a+b}{2}, b]$ (*diviser*). Il suffit ensuite de chercher un zéro de f sur l'un de ces deux intervalles (*régner*), récursivement, puisque l'on peut se ramener au même problème.

Question 1

Justifier que l'on se ramène bien au même problème (avec des arguments différents) et expliquer comment choisir le nouvel intervalle.

Dans cet exemple simple, il n'y a qu'un seul sous-problème et l'étape de combinaison n'est pas nécessaire. Nous verrons bientôt un exemple plus complet.

1. De taille significativement plus petite. Par exemple, si la taille du problème est n, les sous-problèmes seront de taille $\Theta(\alpha n)$ avec $\alpha < 1$. Découper un problème de taille n en un problème de taille $n-1$ et un problème de taille 1 n'est pas une approche *diviser pour régner*.

Question 2

Quel est le cas de base qui conditionne l'arrêt des appels récursifs (attention, il y a un piège) ?

Question 3

Implémenter la fonction de recherche dichotomique du zéro d'une fonction. Elle sera du type `(float -> float) -> float -> float -> float` et son appel devra être `dichotomie f a b epsilon;;`.

2 Exponentiation rapide

Question 4

Réécrire la fonction `expo_rapide : int -> int -> int` d'exponentiation rapide vue en cours.

☞ **Remarque 1**

Il faut impérativement savoir écrire cette fonction sans même avoir à réfléchir.

Question 5

Pourquoi cet algorithme utilise-t-il le paradigme *diviser pour régner* ? Quelles sont les étapes *diviser*, *régner* et *combiner* ? Il y a encore une particularité : les deux sous-problèmes sont identiques !

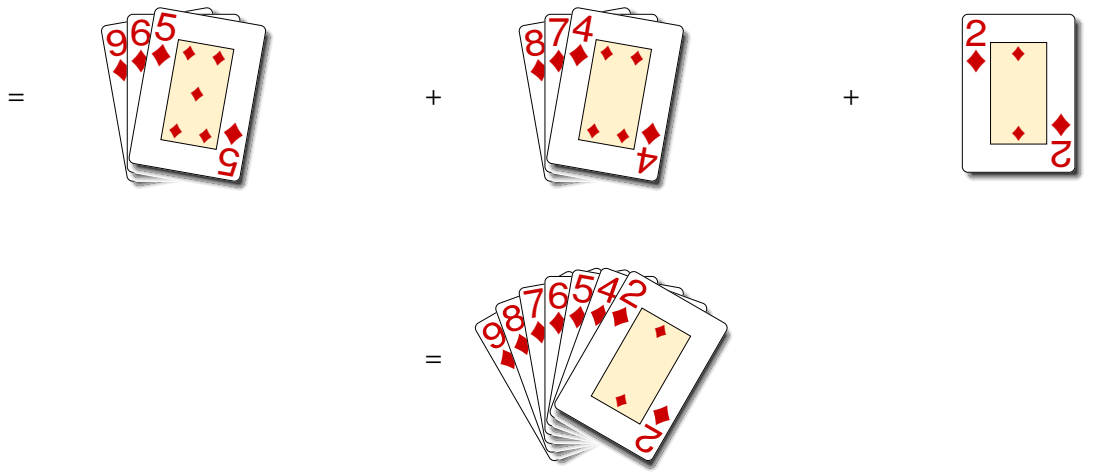
3 Tri fusion

On propose dans cette section un tri plus efficace (ce que l'on montrera la semaine prochaine) en utilisant le paradigme *diviser pour régner*. Illustrons cette idée à l'aide d'un jeu de cartes. S'il y a plus d'une carte (sinon, c'est simple) :

- On divise sa main en deux paquets de même taille (ou presque). Pour cela il suffit par exemple de faire deux tas (initialement vides), de prendre deux cartes de sa main et d'en mettre une dans chaque tas, jusqu'à épuisement de sa main. S'il reste une seule carte en main, on la met arbitrairement dans l'un des deux tas.

- On trie récursivement les deux nouveaux tas, de taille approximativement moitié.

- On fusionne les deux mains. Ceci est relativement simple car elles sont toutes les deux triées. On peut donc comparer la première carte de chaque main et choisir la plus petite, puis recommencer de même avec ce qui suit.



Question 6

Reconnaissez-vous les trois étapes de l'approche *diviser pour régner* ? Cet exemple est-il convaincant ?

Question 7

Implémenter une fonction `division : 'a list -> 'a list * 'a list` qui divise une liste passée en argument en deux sous-listes de même taille, à un près. On rappelle que l'on peut filtrer sur les deux premiers éléments d'une liste en CAML.

Question 8

Implémenter une fonction `fusion : 'a list -> 'a list -> 'a list` qui prend en argument deux listes, supposées triées, et qui renvoie la liste triée formée des éléments des deux listes (avec doublons).

Question 9

Implémenter une fonction `tri_fusion : 'a list -> 'a list` qui trie la liste reçue en argument.

Question 10

Avez-vous bien vérifié la validité de toutes les fonctions écrites jusqu'ici en testant sur des exemples simples et en vérifiant sur des cas particuliers (par exemple ici une liste vide, une liste à un élément, une liste déjà triée, une liste triée par ordre décroissant, etc.) ?

4 Racine carrée

Question 11

Écrire une fonction `racine : int -> int` qui calcule $\lfloor \sqrt{n} \rfloor$ en testant un par un les entiers dans l'ordre croissant jusqu'à trouver le bon. Quelle est sa complexité ?

Question 12

Écrire une nouvelle version `racine_dichotomique` utilisant le principe *diviser pour régner* pour résoudre ce problème. Montrer que la complexité vérifie la relation de récurrence $T(n) = T(\lceil \frac{n}{2} \rceil) + \Theta(1)$. Nous verrons que cette récurrence peut se résoudre en $T(n) = \Theta(\log n)$.