Chapitre 6

Diviser pour régner

Définition 1 :

La méthode du « Diviser pour régner », est une méthode de programmation dans le but d'améliorer la complexité d'un algorithme.
Elle contient 3 temps :

- Diviser : découper un problème initial en 2 sous-problèmes ;
- Régner : résoudre les sous-problèmes ;
- Combiner : calculer une solution au problème initial à partir des solutions des sous-problèmes.

1 Recherche dichotomique

Exercice 1 :

Écrire la fonction `recherche`, qui recherche la valeur d'un entier pris aléatoirement sur l'intervalle $[0, 100]$ et renvoie le nombre d'appel récursif ainsi que le nombre à rechercher.
La valeur `valeur_chercher` est une variable global. La fonction sera donc récursive, avec 3 arguments : la borne inférieur de l'intervalle de recherche, la borne supérieur et un compteur.
Exemple :

```
>>>print(recherche(0,100,1))
>>> (81, 4)
```

2 Exponentiation rapide

Exercice 2 :

Algorithme naïf :

1. Écrire la fonction `puissance_naive`, qui prend en argument un entier a et un entier naturel n et renvoie la valeur de a^n.
2. Déterminer la complexité de la fonction `puissance_naive`.

Méthode du diviser pour régner :
Le but est d'améliorer la complexité de la fonction `puissance`.

- Si $n = 0$ alors `puissance(a, 0)` renvoie 1 ;
- Si $n = 1$ alors `puissance(a, 1)` renvoie `a` ;
- Si $n = 2k$, on pose $\mathtt{a}' =$ `puissance(a, k)` on renvoie alors $\mathtt{a}' \times \mathtt{a}'$;
- Si $n = 2k + 1$, on pose $\mathtt{a}' =$ `puissance(a, k)` on renvoie alors $\mathtt{a} \times \mathtt{a}' \times \mathtt{a}'$;

(attention, il est important de poser `a'` pour ne pas faire deux fois un calcul identique !)

1. Combien de fois l'opérateur $\times$ sera-t-elle utilisée si on calcule a^{10} ? Et pour a^{15} ? Et pour a^{16} ?

2. Montrer la terminaison et la correction de l'algorithme.

3. On note c_n le nombre de $\times$ utilisé pour le calcul de a^n. On note $n = \overline{b_p b_{p-1} ... b_0}^2$ la décomposition de n en base 2.
   Montrer que l'on a :
   $$c_n = c_{\lfloor n/2 \rfloor} + 1 + b_0$$
   En déduire que :
   $$c_n = p + \sum_{k=0}^{p-1} b_k$$

4. Montrer que le nombre d'utilisation de l'opérateur $\times$ dans le calcul de a^n est inférieur à $2\lfloor \log_2(n) \rfloor$.

5. Écrire la fonction `puissance`

3 Tri fusion

Exercice 3 : **Tri fusion**

Le tri repose sur le fait que pour fusionner deux listes (ou tableaux) triées dont la somme des longueurs est n, $n-1$ comparaisons au maximum sont nécessaires.
L'algorithme peut s'effectuer récursivement :

1. On découpe en deux parties à peu près égales les données à trier.

2. On trie les données de chaque partie.

3. On fusionne les deux parties.

La récursivité s'arrête à un moment car on finit par arriver à des listes de 1 élément et alors le tri est trivial.
Exemple de fusion : Fusionner [1 ;2 ;5] et [3 ;4] : On sait que le premier élément de la liste fusionnée sera le premier élément d'une des deux listes d'entrée (soit 1, soit 3), propriété non remarquable sur des listes non triées.
On compare donc 1 et 3 : 1 est plus petit.
[2 ;5] - [3 ;4] -> [1]
On compare 2 et 3 : 2 est plus petit.
[5] - [3 ;4] -> [1 ;2]
On compare 5 et 3 : 3 est plus petit.
[5] - [4] -> [1 ;2 ;3]
On compare 5 et 4 : 4 est plus petit.
[5] -> [1 ;2 ;3 ;4] puis [1 ;2 ;3 ;4 ;5]
Exemple simple, pour la liste [6 ;1 ;2 ;5 ;4 ;7 ;3] : À l'état initial on a les éléments un par un, on les fusionne deux à deux : ([6] [1]) ([2] [5]) ([4] [7]) [3] ; On obtient : ([1 ;6] [2 ;5]) ([4 ;7] [3]) que l'on fusionne deux à deux à nouveau et ainsi de suite : ([1 ;2 ;5 ;6] [3 ;4 ;7]) puis [1 ;2 ;3 ;4 ;5 ;6 ;7].

1. Écrire la fonction fusion de deux listes triées :
   Exemple :

```
>>> fusion([1,5,9,13] , [2,3,5,8,10,12])
[1, 2, 3, 5, 5, 8, 9, 10, 12, 13]
```

   Quelle est le nombre d'itérations de la fonction ?

2. Écrire une fonction fendre qui renvoie pour une liste donnée deux listes de tailles identiques (à une valeur près).
   Exemple :

```
>>> fendre([8,4,1,6,4,9,7,5])
([8, 4, 1, 6], [4, 9, 7, 5])
>>> fendre([8,4,1,6,4,9,7,5,3])
([8, 4, 1, 6], [4, 9, 7, 5, 3])
```

3. Calculer un ordre de grandeur du nombre d'itérations.

4. Écrire la fonction `tri_fusion` :

```
>>> tri_fusion([8,4,1,6,4,9,7,5])
[1, 4, 4, 5, 6, 7, 8, 9]
```

   Estimer le nombre d'itérations nécessaire.