Chapitre 1

Diviser pour régner

1.1 Principe

Diviser pour régner (*Divide ut imperes* en latin ou *divide and conquer* en anglais) est une méthode algorithmique reposant sur trois étapes :

Diviser : on divise le problème initial en sous-problèmes plus simples à résoudre.

Régner : on résout les sous-problèmes.

Combiner : on résout le problème initial à partir des solutions des sous-problèmes.

Cette méthode s'applique la plupart du temps grâce à la récursivité mais il est aussi possible de résoudre les sous-problèmes directement s'ils sont assez petits. Elle s'avère intéressante lorsque la résolution de l'ensemble des sosu-problèmes s'avère moins coûteuse en temps et ou en mémoire que le problème initial.

1.2 Exemples

1.2.1 Puissance récursive

On a vu précédemment comment calculer la puissance un nombre grâce à la fonction ci-dessous.

```
def puissance(x : float or int, n : int) -> float or int :

    if n == 0 :
        return 1

    else :
        return x * puissance(x,n-1)
```

À chaque appel, la puissance est diminuée de 1. Toutefois, il est possible de faire mieux en la divisant par deux à chaque appel en considérant la disjonction de cas suivante.

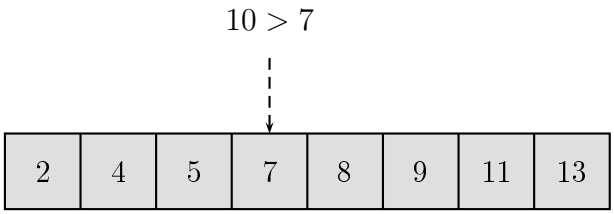
$$x^n = \begin{cases} 1 & \text{si } n = 0, \\ (x \times x)^{\frac{n}{2}} & \text{si } n \text{ est pair et non nul,} \\ x \times (x \times x)^{\frac{n-1}{2}} & \text{si } n \text{ est impair.} \end{cases}$$

Le cas de la méthode récursive simple donne une complexité temporelle en $O(n)$ puisqu'on calcule toutes les puissances entre 0 et n. La méthode récursive couplée à diviser pour régner donne elle un résultat en $O(\ln_2(n))$, ce qui est infiniment meilleur.

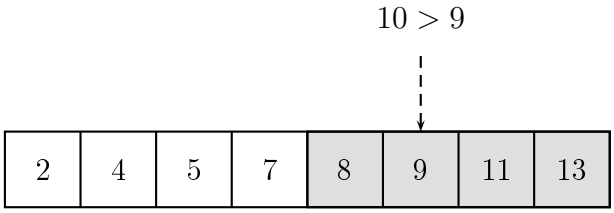
1.2.2 Recherche dichotomique

La recherche dichotomique est une méthode de recherche d'une valeur dans un tableau trié. Dans celle-ci, on compare la valeur recherchée à la valeur de l'élément situé au milieu du tableau. Si cette dernière est plus grande que la valeur recherchée, c'est qu'il faut chercher dans la partie gauche du tableau ; si elle est plus petite, c'est qu'il faut chercher dans la partie droite. Chacune des deux parties du tableau constitue un sous-tableau dans le lequel on réitère le procédé.

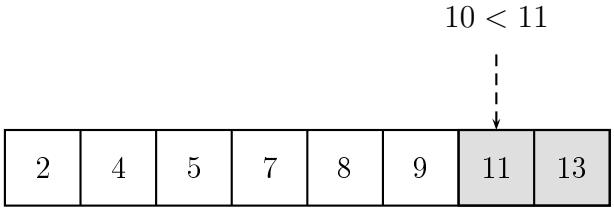
Exemple : On recherche 10 dans le tableau ci-dessous. On commence par comparer à l'élément à la positon $\left\lfloor \frac{i_D + i_P}{2} \right\rfloor$ partie entière de $\frac{i_D + i_P}{2}$ (égale au reste la division euclidienne par 2) où i_P et i_D sont respectivement les indices du premier et du dernier élément du sous-tableau considéré.



Étape 1 :
Position du pivot : $\lfloor (8+0)/2 \rfloor = 4$. 10 étant plus grand que la valeur pivot. On réitère le processus sur la partie droite du tableau. La partie blanche dans le schéma représente la partie du tableau qui n'est plus considérée.



Étape 2 :
Position du pivot : $\lfloor (8+5)/2 \rfloor = 6$. 10 étant à nouveau plus grand que la valeur pivot. On réitère à nouveau le processus sur la partie droite du tableau.



Étape 3 :
Position du pivot : $\lfloor (8+7)/2 \rfloor = 7$. Sur cette dernière étape, la partie restante du tableau est éliminée et il en ressort que 10 n'appartient pas au tableau.

La recherche dichotomique est donc un algorithme pouvant s'écrire de façon récursive sur des tableaux dont la taille est divisée par deux à chaque étape. Il s'agit de l'approche « diviser » de la méthode diviser pour régner : on divise par deux la difficulté du problème à chaque étape. On notera toutefois que les deux approches : « régner » et « combiner », ne sont pas présentes ici. En effet, elles ne sont pas nécessaires puisque à la fin de la division, on obtient un tableau contenant un unique élément qui est ou non la valeur recherchée, ce qui répond au problème.

La recherche dichotomique a une complexité temporelle en $O(\ln_2(n))$ où n représente la taille du tableau dans lequel est effectuée la recherche.

1.2.3 Tri rapide

Le tri rapide (*quicksort* en anglais) est une méthode de tri inventé par C.A.R. Hoare en 1961. Il consiste à choisir une position **pivot** puis d'effectuer des échanges entre les éléments restant de façon à avoir tous les éléments inférieurs à la valeur du pivot dans le sous-tableau à sa gauche et tous ceux qui sont supérieurs à sa valeur dans le sous-tableau à sa droite. La valeur du pivot est ainsi dans sa position finale. On réitère alors cette opération sur les sous-tableaux obtenus. Le pivot peut être choisi aléatoirement à chaque étape.

Il repose sur les deux parties « diviser » et « régner » de la méthode diviser pour régner. À chaque étape, on divise le tableau en cours en deux sous-tableaux sur lesquels on peut « régner » en les triant. Il n'y a pas besoin ici de recombiner car le tableau reste intact.

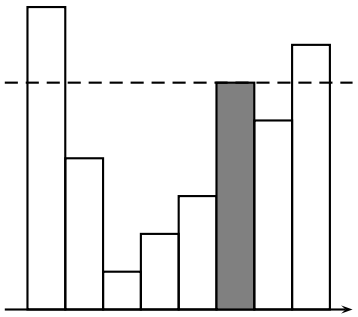
Le tri rapide possède une complexité moyenne en $O(n\ln(n))$, ce qui est optimal pour un algorithme de tri. Toutefois, il ne s'agit que d'une moyenne. En effet, dans le pire des cas, celle-ci est quadratique ; cependant ce cas est improbable et malgré ce désavantage théorique, cet algorithme reste très utilisé.

Un autre de ces désavantages est qu'il ne tire pas partie d'un tableau en entrée déjà partiellement trié, auquel cas le tri par insertion s'avère plus performant.

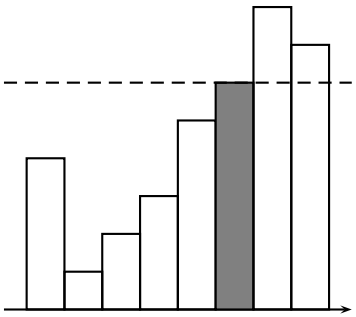
Algorithme 1 : Tri rapide

```
1  Fonction partitionner(T : tableau, premier : entier, dernier : entier, pivot : entier) :
2
3      échanger(T[pivot], T[dernier]) // Échange le pivot avec le dernier du tableau, le
         pivot devient le dernier du tableau
4
5      j ← premier
6
7      Pour i allant de premier à dernier − 1 :
8
9          // Comparaison du pivot (actuellement dernier) avec l'élément en cours
10         Si T[i] ≤ T[dernier] :
11
12             échanger(T[i], T[j])
13             j ← j + 1
14
15     échanger (T[dernier], T[j])
16
       Renvoyer : j // indice de la première valeur du tableau supérieure au pivot
17
18 Fonction tri_rapide(T : tableau, premier : entier, dernier : entier) :
19
20     Si premier < dernier :
21
22         pivot ← choix_pivot(T, premier, dernier)
23         pivot ← partitionner(T, premier, dernier, pivot)
24         tri_rapide(T, premier, pivot − 1) // appel sur le sous-tableau de gauche
25         tri_rapide(T, pivot + 1, dernier) // appel sur le sous-tableau de gauche
```

Illustration : on considère un tableau dont les valeurs pourraient être représentées graphiquement de la façon suivante. Le pivot est obtenu aléatoirement est égale à 5, c'est donc la sixième valeur qui sera fixée à la fin de l'opération de tri. À la fin de l'étape, toutes les valeurs contenues dans le sous-tableau à gauche du pivot sont inférieures à sa valeur tandis que toutes celles dans le sous-tableau à gauche du pivot lui sont supérieures.



Étape 1, pivot : 5



Étape 1, fin

Exemple : Appliquons l'algorithme du tri rapide au tableau $T[18, 14, 11, 12, 13, 16, 15, 17]$, lequel correspond aux graphes ci-dessus.

Étape 1 : Appel de la fonction ***tri_ rapide***$(T, 0, longueur(T) - 1)$.

- On a $premier = 0$ et $dernier = 7$, les indices des premier et dernier éléments du tableau, le test $premier < dernier$ est donc vérifié.
- On prend un pivot aléatoirement entre 0 et 7 : 5 (afin de retrouver l'exemple ci-dessus). La valeur numéro 5 du tableau est 16.
- Appel de la fonction ***partitionner***$(T, 0, 7, 5)$.
- On échange la valeur du pivot avec la dernière du tableau, ce dernier devient

$$T[18, 14, 11, 12, 13, 17, 15, 16].$$

- La valeur 0 est affectée à j.
- On effectue la boucle pour i allant de $premier = 0$ à $dernier - 1 = 7 - 1 = 6$. Dans le tableau ci-dessous, la deuxième colonne le résultat du test contenu dans la boucle et les troisième et quatrième l'évolution des valeurs de T et j.

i	$T[i] \leq 16$	T	j
0	$(18 \leq 16) =$ Faux	id.	0
1	$(14 \leq 16) =$ Vrai	[14, 18, 11, 12, 13, 17, 15, 16]	1
2	$(11 \leq 16) =$ Vrai	[14, 11, 18, 12, 13, 17, 15, 16]	2
3	$(12 \leq 16) =$ Vrai	[14, 11, 12, 18, 13, 17, 15, 16]	3
4	$(13 \leq 16) =$ Vrai	[14, 11, 12, 13, 18, 17, 15, 16]	4
5	$(17 \leq 16) =$ Faux	id.	4
6	$(15 \leq 16) =$ Vrai	[14, 11, 12, 13, 15, 17, 18, 16]	5

On constate que j est la position du premier élément du tableau supérieur à la valeur du pivot. En effectuant l'échange entre leurs deux valeurs, on obtient le tableau ci-dessous et on renvoie $j = 5$.

$$T[14,\ 11,\ 12,\ 13,\ 15,\ 16,\ 18,\ 17].$$

Étape 2.1 (sous-tableau de gauche) : La dernière valeur de j est affectée à *pivot* qui vaut maintenant 5. Appel de la fonction ***tri_ rapide***$(T,\ 0,\ pivot - 1)$.

- On a $premier = 0$ et $dernier = 4$, les indices des premier et dernier éléments du sous-tableau considéré, le test $premier < dernier$ est donc vérifié.
- On prend un pivot aléatoirement entre 0 et 4 : 2. La valeur numéro 2 du tableau est 12.
- Appel de la fonction ***partitionner***$(T,\ 0,\ 4,\ 2)$.
- On échange la valeur du pivot avec la dernière du sous-tableau $T[0:4]$, ce dernier devient

$$T[14,\ 11,\ 15,\ 13,\ 12,\ 16,\ 18,\ 17].$$

- La valeur 0 est affectée à j.
- On effectue la boucle pour i allant de $premier = 0$ à $dernier - 1 = 4 - 1 = 3$.

i	$T[i] \leq 12$	T	j
0	$(14 \leq 12) =$ Faux	id.	0
1	$(11 \leq 12) =$ Vrai	[11, 14, 15, 13, 12, 16, 18, 17]	1
2	$(15 \leq 12) =$ Faux	id.	1
3	$(12 \leq 12) =$ Faux	id.	1

En effectuant l'échange entre la valeur du pivot et celle de l'élément en position j, on obtient

$$T[11,\ 12,\ 15,\ 13,\ 14,\ 16,\ 18,\ 17].$$

Étape 2.1.1 (sous-tableau de gauche, partie gauche) : La dernière valeur de j est affectée à *pivot* qui vaut maintenant 1. Appel de la fonction ***tri_ rapide***$(T,\ 0,\ pivot - 1)$.

- On a $premier = 0$ et $dernier = 0$, les indices des premier et dernier éléments du sous-tableau considéré, le test $premier < dernier$ n'est donc pas vérifié et l'exécution de ***tri_ rapide***$(T,\ 0,\ 0)$ s'arrête là.

Étape 2.1.2 (sous-tableau de gauche, partie droite) : Appel de la fonction ***tri_ rapide***$(T,\ pivot + 1,$ dernier$)$.

- On a $premier = 2$ et $dernier = 4$, les indices des premier et dernier éléments du sous-tableau considéré, le test $premier < dernier$ est donc vérifié.
- On prend un pivot aléatoirement entre 2 et 4 : 3. La valeur numéro 3 du tableau est 13.
- Appel de la fonction ***partitionner***$(T,\ 2,\ 4,\ 3)$.
- On échange la valeur du pivot avec la dernière du sous-tableau $T[2:4]$, ce dernier devient

$$T[11,\ 12,\ 15,\ 14,\ 13,\ 16,\ 18,\ 17].$$

- La valeur 2 est affectée à j.
- On effectue la boucle pour i allant de $premier = 2$ à $dernier - 1 = 4 - 1 = 3$.

i	$T[i] \leq 13$	T	j
2	$(15 \leq 13) =$ Faux	id.	2
3	$(14 \leq 13) =$ Faux	id.	2

En effectuant l'échange entre la valeur du pivot et celle de l'élément en position j, on obtient

$$T[11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 18,\ 17].$$

Étape 2.1.2 (sous-tableau de gauche, partie droite, sous-partie gauche) : La dernière valeur de j est affectée à *pivot* qui vaut maintenant 2. Appel de la fonction ***tri_ rapide***(T, $pivot + 1$, dernier).

- On a $premier = 3$ et $dernier = 4$, les indices des premier et dernier éléments du sous-tableau considéré, le test $premier < dernier$ est donc vérifié.
- On prend un pivot aléatoirement entre 3 et 4 : 4. La valeur numéro 3 du tableau est 14.
- Appel de la fonction ***partitionner***(T, 3, 4, 4).
- L'échange de la valeur du pivot avec la dernière du sous-tableau $T[3 : 4]$ ne change pas le tableau.
- La valeur 3 est affectée à j.
- On effectue la boucle pour i allant de $premier = 3$ à $dernier - 1 = 4 - 1 = 3$.

i	$T[i] \leq 14$	T	j
3	$(13 \leq 14) =$ Vrai	id.	4

L'échange entre la valeur du pivot et celle de l'élément en position j ne change pas le tableau. Cette étape était d'évidence inutile mais elle permet de visualiser que le tri rapide laisse intact un tableau trié.

Étape 2.2 (sous-tableau de droite) : Les appels récursifs sur la partie gauche sont terminés, on passe à la partie droite. La reprend la valeur de j de la fin de l'étape 1, laquelle est affectée à *pivot* qui vaut maintenant 5. Appel de la fonction ***tri_ rapide***(T, $pivot+1$, dernier).

- On a $premier = 6$ et $dernier = 7$, les indices des premier et dernier éléments du sous-tableau considéré, le test $premier < dernier$ est donc vérifié.
- On prend un pivot aléatoirement entre 6 et 7 : 7. La valeur numéro 7 du tableau est 17.
- Appel de la fonction ***partitionner***(T, 6, 7, 7).
- L'échange de la valeur du pivot avec la dernière du sous-tableau $T[6 : 7]$ ne change pas le tableau.
- La valeur 6 est affectée à j.
- On effectue la boucle pour i allant de $premier = 6$ à $dernier - 1 = 7 - 1 = 6$.

i	$T[i] \leq 17$	T	j
6	$(18 \leq 17) =$ Faux	id.	6

- En effectuant l'échange entre la valeur du pivot et celle de l'élément en position j, on obtient

$T[11, 12, 13, 14, 15, 16, 17, 18]$.

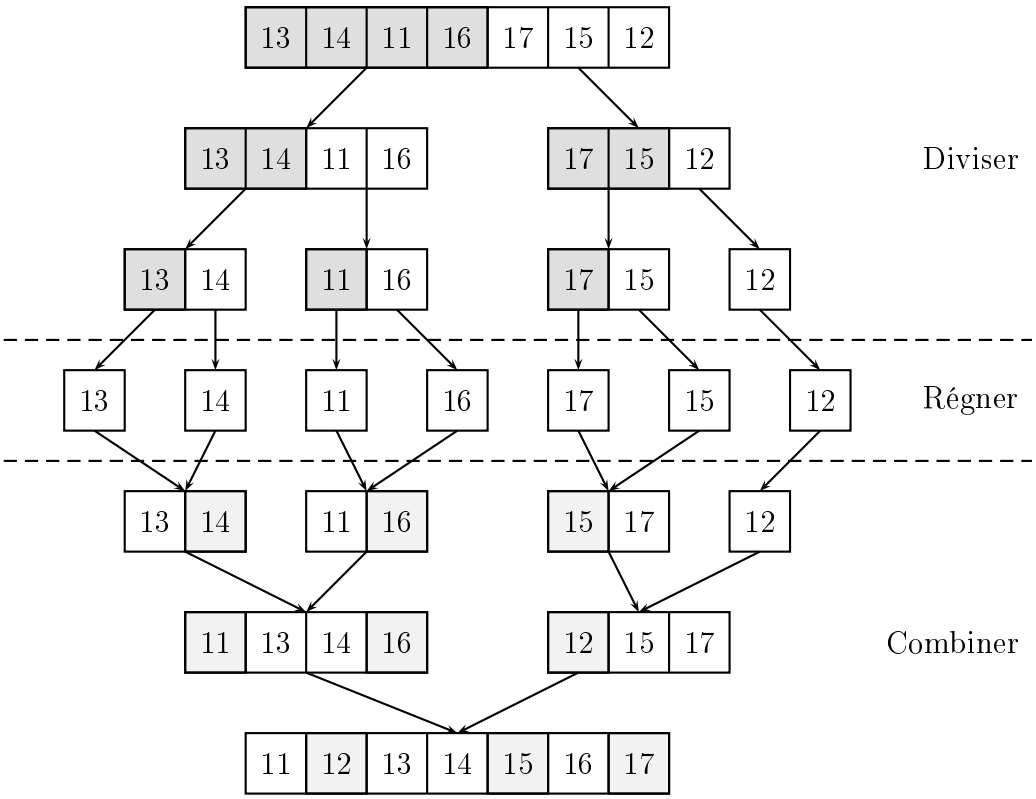
Étapes 2.2.1 et 2.2.2 (sous-tableau de droite) : ces deux appels n'aboutissent à rien ils se font sur des tableaux de tailles 1. La récursion prend fin.

1.2.4 Tri fusion

Le tri fusion est un algorithme de tri inventé en 1945 par John von Neumann (encore lui). Il repose sur les trois axes de la méthode diviser pour régner et consiste à diviser chaque tableau sur lequel il est appelé en deux sous-tableau jusqu'à arriver à des tableaux unitaires (nécessairement triés) puis à les recombiner de façon triés. Il a donc une écriture naturellement récursive.

Comme le tri rapide, sa complexité temporelle est optimale pour un algorithme de tri : en $O(n\ln(n))$ avec n la taille du tableau à trier. Et cela est aussi valable dans le pire des cas contrairement au tri rapide. La complexité en espace (mémoire) est dans le pire des cas en $O(n)$ (taille du tableau) et dans le meilleur en $O(1)$ (tri sur place).

Illustration : de l'algorithme de tri fusion sur le tableau [13, 14, 11, 16, 17, 15, 12]. Dans la partie « diviser », les zones grises représentent les sous-tableaux gauches sur lesquels sont appelés la fonction de tri à l'étape suivante. Dans la partie « combiner », les zones grises représentent les valeurs issues du sous-tableau droit à l'étape précédente.



Algorithme 2 : Tri fusion

```
1  Fonction fusionner(A : tableau, B : tableau) :
2  |
3  |  Si A est vide :
   |  |  Renvoyer : B
4  |
5  |  Si B est vide :
   |  |  Renvoyer : A
6  |
7  |  Si A[0] ≤ B[0] :
   |  |  Renvoyer : A[0] + fusionner(A[1 : n_A], B)
8  |  Sinon
   |  |  Renvoyer : B[0] + fusionner(A, B[1 : n_B])
9  |
10
11 Fonction tri_fusion(T : tableau) :
12 |
13 |  n ← longueur(T)
14 |
15 |  Si n ≤ 1 :
   |  |  Renvoyer : T
16 |  Sinon
   |  |  Renvoyer : fusionner(tri_fusion(T[0 : n/2]), tri_fusion(T[n/2 + 1 : n]))
```

Exemple : concentrons nous sur la partie la plus difficile, la fusion. Fusionnons A[2, 3] et B[1, 4, 5].

fusionner(A, B) :
A[0]≥B[0] donc renvoie : B[0] + ***fusionner***(A, B[1 : 2])

Appel ***fusionner***(A, B[1 : 2]) :
A[0]≤B[1] donc renvoi : A[0] + ***fusionner***(A[1 : 1], B[1 : 2])

Appel ***fusionner***(A[1], B[1 : 2]) :
A[1]≤B[1] donc renvoi : A[1] + ***fusionner***(A[], B[1 : 2])

Appel ***fusionner***(A[], B[1 : 2]) :
A[] vide donc renvoi : B[1 : 2]

Renvoi : A[1] + B[1 : 2] = [3, 4, 5]

Renvoi : A[0] + [3, 4, 5] = [2, 3, 4, 5]

Renvoi : B[0] + [2, 3, 4, 5] = [1, 2, 3, 4, 5]

Remarque : la fusion s'effectue toujours sur des tableaux triés.

1.3 Ressources supplémentaires

— Vidéo sur le tri fusion.

1.4 Exercices

1.4.1 Démarrage

Exercice 1.1. [Puissance rapide] Programmer une fonction de calcul de puissance d'un nombre basée sur la méthode diviser pour régner.

Exercice 1.2. [Recherche dichotomique] Programmer une fonction de recherche dichotomique basée sur la méthode diviser pour régner.

Exercice 1.3. Exécuter l'algorithme du tri rapide sur le tableau [15, 16, 12, 11, 13, 14]. On choisira le pivot aléatoirement.

Exercice 1.4. Exécuter l'algorithme du tri fusion sur les deux listes de l'exercice précédent.

1.4.2 Approfondissement

Exercice 1.5. [Tri rapide] Programmer une fonction de tri rapide.

Exercice 1.6. [Tri fusion] Programmer une fonction de tri fusion.

1.4.3 Entraînement

Exercice 1.7. Exécuter l'algorithme du tri rapide sur le tableau [14, 15, 11, 12, 13, 16]. On choisira le pivot aléatoirement.

Exercice 1.8. Exécuter l'algorithme du tri fusion sur le tableau de l'exercice précédent.