Diviser pour régner

M. Tellene

Diviser pour régner

Prenons une situation basique : Alice veut ranger ses bandes dessinées par ordre alphabétique de titre

Où est le problème ?

Diviser pour régner

Prenons une situation basique : Alice veut ranger ses bandes dessinées par ordre alphabétique de titre

Où est le problème ?

Elle en possède plus de 300 !

Diviser pour régner

Prenons une situation basique : Alice veut ranger ses bandes dessinées par ordre alphabétique de titre

Où est le problème ?

Elle en possède plus de 300 !

Alice fait donc appelle à son frère Bob pour l'aider, mais comment peut-il l'aider de manière efficace ?

Diviser pour régner

Prenons une situation basique : Alice veut ranger ses bandes dessinées par ordre alphabétique de titre

Où est le problème ?

Elle en possède plus de 300 !

Alice fait donc appelle à son frère Bob pour l'aider, mais comment peut-il l'aider de manière efficace ?

Alice et Bob se partagent les bandes dessinées et chacun d'eux trie sa moitié, sous la forme d'une pile de bande dessinées. Ensuite les bandes dessinées sont sont rangées dans la bibliothèque en fusionnant les deux piles

Diviser pour régner

Dans un contexte informatique, cette façon de procéder suggère que plusieurs ordinateurs ou plusieurs programmes pourraient collaborer pour effectuer une tâche

Diviser pour régner

Dans un contexte informatique, cette façon de procéder suggère que plusieurs ordinateurs ou plusieurs programmes pourraient collaborer pour effectuer une tâche

Il s'avère que même un unique programme peut tirer avantages à **décomposer un problème en sous-problèmes plus petits ou plus simples** qu'il va résoudre successivement

Diviser pour régner

Si l'on transpose cette idée au problème d'Alice, cette dernière peut trier elle-même ces bandes dessinées

Elle peut partager les bandes dessinées en deux tas égaux, les trier puis les fusionner

Diviser pour régner

Si l'on transpose cette idée au problème d'Alice, cette dernière peut trier elle-même ces bandes dessinées

Elle peut partager les bandes dessinées en deux tas égaux, les trier puis les fusionner

Pour trier chacun des deux tas, elle peut recommencer ainsi avec la même idée, jusqu'à ce que chaque tas soit suffisamment petit pour pouvoir être trié sans effort

Diviser pour régner

Si l'on transpose cette idée au problème d'Alice, cette dernière peut trier elle-même ces bandes dessinées

Elle peut partager les bandes dessinées en deux tas égaux, les trier puis les fusionner

Pour trier chacun des deux tas, elle peut recommencer ainsi avec la même idée, jusqu'à ce que chaque tas soit suffisamment petit pour pouvoir être trié sans effort

Alice vient d'inventer le **tri fusion**, une méthode de tri extrêmement efficace

Diviser pour régner

Le tri fusion repose sur le principe ≪ **Diviser pour régner** ≫

Diviser pour régner

Le tri fusion repose sur le principe « **Diviser pour régner** »

Ce principe consiste à :

1. décomposer un problème à résoudre en sous-problèmes, plus petits, puis à les résoudre, éventuellement en appliquant le même principe autant de fois que nécessaire
2. combiner les résultats des sous-problèmes pour en déduire le résultat du problème initial

Diviser pour régner

Le tri fusion repose sur le principe « **Diviser pour régner** »

Ce principe consiste à :

1. décomposer un problème à résoudre en sous-problèmes, plus petits, puis à les résoudre, éventuellement en appliquant le même principe autant de fois que nécessaire
2. combiner les résultats des sous-problèmes pour en déduire le résultat du problème initial

Cette idée de se ramener à la résolution de sous-problèmes ne vous fait pas penser à quelques chose ?

Diviser pour régner

Le tri fusion repose sur le principe « **Diviser pour régner** »

Ce principe consiste à :

1. décomposer un problème à résoudre en sous-problèmes, plus petits, puis à les résoudre, éventuellement en appliquant le même principe autant de fois que nécessaire
2. combiner les résultats des sous-problèmes pour en déduire le résultat du problème initial

Cette idée de se ramener à la résolution de sous-problèmes ne vous fait pas penser à quelques chose ? La récursivité

Diviser pour régner - Recherche dichotomique

Retour sur un algorithme vu en première (normalement) : **la recherche dichotomique**

Diviser pour régner - Recherche dichotomique

Retour sur un algorithme vu en première (normalement) : **la recherche dichotomique**

Rappel du problème : déterminer si un élément `x` se trouve dans un tableau `t`

Diviser pour régner - Recherche dichotomique

Retour sur un algorithme vu en première (normalement) : **la recherche dichotomique**

Rappel du problème : déterminer si un élément `x` se trouve dans un tableau `t`

Première solution

Diviser pour régner - Recherche dichotomique

Retour sur un algorithme vu en première (normalement) : **la recherche dichotomique**

Rappel du problème : déterminer si un élément `x` se trouve dans un tableau `t`

Première solution : parcourir tout le tableau

Diviser pour régner - Recherche dichotomique

Retour sur un algorithme vu en première (normalement) : **la recherche dichotomique**

Rappel du problème : déterminer si un élément `x` se trouve dans un tableau `t`

Première solution : parcourir tout le tableau

```
def est_present(t, x):
    for i in range(len(t)):
        if t[i] == x:
            return True
    return False
```

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Dans le meilleur des cas :

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Dans le meilleur des cas : *O(1)*

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Dans le meilleur des cas : $O(1)$

Dans le pire des cas :

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Dans le meilleur des cas : *O(1)*

Dans le pire des cas : *O(n)*

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Dans le meilleur des cas : *O(1)*

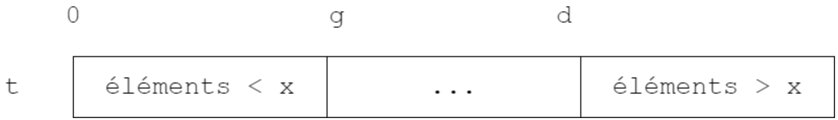
Dans le pire des cas : *O(n)*

La complexité reste bonne mais il est possible de faire mieux !

Diviser pour régner - Recherche dichotomique

La recherche dichotomique est plus efficace que le parcours d'un tableau pour déterminer la présence d'un élément

L'idée principale de cet algorithme consiste à délimiter une portion du tableau dans laquelle `x` peut encore se trouver ; avec deux indices `g` et `d`



Diviser pour régner - Recherche dichotomique

La recherche dichotomique est plus efficace que le parcours d'un tableau pour déterminer la présence d'un élément

L'idée principale de cet algorithme consiste à délimiter une portion du tableau dans laquelle `x` peut encore se trouver ; avec deux indices `g` et `d`

	0	g	d
t	éléments < x	...	éléments > x

Pour que la recherche dichotomique puisse être utilisée, il faut que le tableau **soit trié**

Diviser pour régner - Recherche dichotomique

On compare la valeur au centre de l'intervalle $[g\ ..\ d]$

Selon le résultat de cette comparaison :

- on signale que l'on a trouvé la valeur
- on se déplace vers la gauche ou vers la droite $\rightarrow$ on réduit l'intervalle de recherche

Pourquoi la recherche dichotomique suit le principe diviser pour régner ?

Diviser pour régner - Recherche dichotomique

On compare la valeur au centre de l'intervalle *[g .. d]*

Selon le résultat de cette comparaison :

- on signale que l'on a trouvé la valeur
- on se déplace vers la gauche ou vers la droite $\rightarrow$ on réduit l'intervalle de recherche

Pourquoi la recherche dichotomique suit le principe diviser pour régner ?

On réduit le problème de la recherche dans *[g .. d]* à celui de la recherche dans un intervalle plus petit

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Prenons le pire des cas :

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Prenons le pire des cas : `x` n'est pas dans le tableau, ce qui nous oblige à répéter la boucle jusqu'à ce que l'intervalle soit vide

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Prenons le pire des cas : `x` n'est pas dans le tableau, ce qui nous oblige à répéter la boucle jusqu'à ce que l'intervalle soit vide

Prenons l'exemple d'un tableau de taille 100

Première itération, on va se restreindre à un sous-ensemble contenant 49 ou 50 éléments (selon le côté choisi). Prenons le cas le moins favorable

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Prenons le pire des cas : `x` n'est pas dans le tableau, ce qui nous oblige à répéter la boucle jusqu'à ce que l'intervalle soit vide

Prenons l'exemple d'un tableau de taille 100

Première itération, on va se restreindre à un sous-ensemble contenant 49 ou 50 éléments (selon le côté choisi). Prenons le cas le moins favorable

A la deuxième itération, on va se retrouver avec au plus 25 éléments

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Prenons le pire des cas : `x` n'est pas dans le tableau, ce qui nous oblige à répéter la boucle jusqu'à ce que l'intervalle soit vide

Prenons l'exemple d'un tableau de taille 100

Première itération, on va se restreindre à un sous-ensemble contenant 49 ou 50 éléments (selon le côté choisi). Prenons le cas le moins favorable

A la deuxième itération, on va se retrouver avec au plus 25 éléments

Puis 12, puis 6, puis 3, puis 1, puis 0

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Ainsi, pour savoir si un élément est dans un tableau de 100 éléments, on fera 7 comparaisons (au lieu de 100)

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Ainsi, pour savoir si un élément est dans un tableau de 100 éléments, on fera 7 comparaisons (au lieu de 100)

De la même manière, on peut montrer que 20 itérations **sont toujours suffisantes**, dans le pire des cas, pour rechercher une valeur dans un tableau d'un million d'éléments (cf divisions successives d'un million par 2)

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

Ainsi, pour savoir si un élément est dans un tableau de 100 éléments, on fera 7 comparaisons (au lieu de 100)

De la même manière, on peut montrer que 20 itérations **sont toujours suffisantes**, dans le pire des cas, pour rechercher une valeur dans un tableau d'un million d'éléments (cf divisions successives d'un million par 2)

Inversement, on peut chercher la plus petite puissance de 2 qui dépasse la taille du tableau :

$$2^7 > 100\,;\, 2^{20} > 10^6$$

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

De manière générale, pour un tableau `t` de taille N, la complexité est dans le pire des cas égal à :

$$2^k > N > 2^{k-1}$$

Diviser pour régner - Recherche dichotomique

Analyse de cet algorithme : complexité

De manière générale, pour un tableau `t` de taille N, la complexité est dans le pire des cas égal à :

$$2^k > N > 2^{k-1}$$

Il existe une fonction mathématique, appelée **logarithme de base 2**, qui permet de déterminer le plus petit entier k

Cette fonction permet de dire que la complexité de l'algorithme de recherche dichotomique est de complexité logarithmique, en $O(log(n))$

Diviser pour régner - Tri fusion

Maintenant que les rappels ont été faits, passons à un autre algorithme, mentionné plus tôt, suivant le principe « Diviser pour régner » : **le tri fusion**

1. cf. la première NSI

Diviser pour régner - Tri fusion

Maintenant que les rappels ont été faits, passons à un autre algorithme, mentionné plus tôt, suivant le principe « Diviser pour régner » : **le tri fusion**

Le tri fusion est le tri implémenté par Python

Cette méthode de tri est **bien meilleur** comparé au tri insertion et sélection [1]

1. cf. la première NSI

Diviser pour régner - Tri fusion

Comment ça marche ?

Diviser pour régner - Tri fusion

Comment ça marche ?

1. Séparer le tableau à trier en deux tableaux de même taille (à un élément près)

Diviser pour régner - Tri fusion

Comment ça marche ?

1. Séparer le tableau à trier en deux tableaux de même taille (à un élément près)
2. Trier chacun des deux tableaux avec le tri fusion, récursivement

Diviser pour régner - Tri fusion

Comment ça marche ?

1. Séparer le tableau à trier en deux tableaux de même taille (à un élément près)
2. Trier chacun des deux tableaux avec le tri fusion, récursivement
3. Fusionner les deux tableaux

Diviser pour régner - Tri fusion

Comment ça marche ?

1. Séparer le tableau à trier en deux tableaux de même taille (à un élément près)
2. Trier chacun des deux tableaux avec le tri fusion, récursivement
3. Fusionner les deux tableaux

Pourquoi le tri fusion suit le principe « Diviser pour régner » ?

Diviser pour régner - Tri fusion

Comment ça marche ?

1. Séparer le tableau à trier en deux tableaux de même taille (à un élément près)
2. Trier chacun des deux tableaux avec le tri fusion, récursivement
3. Fusionner les deux tableaux

Pourquoi le tri fusion suit le principe « Diviser pour régner » ?

On ramène le problème du tri d'un tableau à un sous-problème du tri de deux tableaux plus petits jusqu'à parvenir à des tableaux d'un élément

Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`

Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`

6	2	1	4

Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`

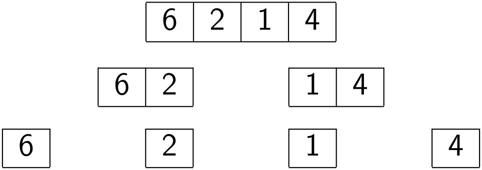
6	2	1	4

6	2

1	4

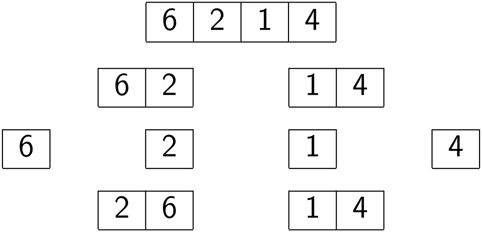
Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`



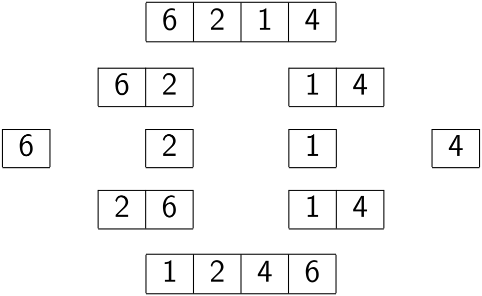
Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`



Diviser pour régner - Tri fusion

Un exemple : `[6,2,1,4]`



Diviser pour régner - Tri fusion

Pour que le tri fusion puisse fonctionner, il faut couper le tableau à trier en deux, mais comment faire ?

Diviser pour régner - Tri fusion

Pour que le tri fusion puisse fonctionner, il faut couper le tableau à trier en deux, mais comment faire ?

Faire une fonction `coupe(tab)`

Diviser pour régner - Tri fusion

Pour que le tri fusion puisse fonctionner, il faut couper le tableau à trier en deux, mais comment faire ?

Faire une fonction `coupe(tab)`

```
def coupe(tab):
    t1 = []
    t2 = []
    moitie = len(tab)//2
    for i in range(moitie):
        t1.append(tab[i])
    for i in range(moitie, len(tab)):
        t2.append(tab[i])
    return t1, t2
```

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Rappels :

	Meilleur des cas	**Pire des cas**
Tri insertion	$O(n)$	$O(n^2)$
Tri sélection	$O(n^2)$	$O(n^2)$

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Rappels :

	Meilleur des cas	**Pire des cas**
Tri insertion	$O(n)$	$O(n^2)$
Tri sélection	$O(n^2)$	$O(n^2)$

Le tri fusion, quant à lui, demande un temps proportionnel à $n\log_2(n)$ (dans tous les cas)

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Preuve de la complexité (attention les yeux) :

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Preuve de la complexité (attention les yeux) :

Master theorem : si $T(n) = aT(\frac{n}{b}) + \Theta(n^d)$
où $a > 0$, $b > 1$; $d \geq 0$

$$\Rightarrow T(n) = \begin{cases} \Theta(n^d) & si\ d > \log_b(a) \\ \Theta(n^d \log_2(n)) & si\ d = \log_b(a) \\ \Theta(n^{\log_b(a)}) & si\ d < \log_b(a) \end{cases}$$

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Preuve de la complexité (attention les yeux) :

Master theorem : si $T(n) = aT(\frac{n}{b}) + \Theta(n^d)$
où $a > 0$, $b > 1$; $d \geq 0$

$$\Rightarrow T(n) = \begin{cases} \Theta(n^d) & si\ d > \log_b(a) \\ \Theta(n^d \log_2(n)) & si\ d = \log_b(a) \\ \Theta(n^{\log_b(a)}) & si\ d < \log_b(a) \end{cases}$$

Tir fusion : $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Preuve de la complexité (attention les yeux) :

Master theorem : si $T(n) = aT(\frac{n}{b}) + \Theta(n^d)$
où $a > 0$, $b > 1$; $d \geq 0$

$$\Rightarrow T(n) = \begin{cases} \Theta(n^d) & si\ d > \log_b(a) \\ \Theta(n^d \log_2(n)) & si\ d = \log_b(a) \\ \Theta(n^{\log_b(a)}) & si\ d < \log_b(a) \end{cases}$$

Tir fusion : $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
$d = 1$ et $\log_b(a) = \log_2(2) = 1$

Diviser pour régner - Tri fusion

Analyse de cet algorithme : complexité

Preuve de la complexité (attention les yeux) :

Master theorem : si $T(n) = aT(\frac{n}{b}) + \Theta(n^d)$
où $a > 0$, $b > 1$; $d \geq 0$

$$\Rightarrow T(n) = \begin{cases} \Theta(n^d) & si\ d > \log_b(a) \\ \Theta(n^d \log_2(n)) & si\ d = \log_b(a) \\ \Theta(n^{\log_b(a)}) & si\ d < \log_b(a) \end{cases}$$

Tir fusion : $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
$d = 1$ et $\log_b(a) = \log_2(2) = 1$
$\Rightarrow T(n) = \Theta(n \log_2(n))$

Diviser pour régner

En résumé :

- ≪ Diviser pour régner ≫ est une méthode de programmation
- Cette méthode se divise en trois étapes :
  1. Diviser : découper un problème initial en sous-problèmes
  2. Régner : résoudre les sous-problèmes (récursivement ou directement s'ils sont assez petits)
  3. Combiner : calculer une solution au problème initial à partir des solutions des sous-problèmes
- Cette technique fournit des algorithmes **efficaces** pour de nombreux problèmes