Algorithmique Avancée – Alternants 2A
Diviser pour Régner

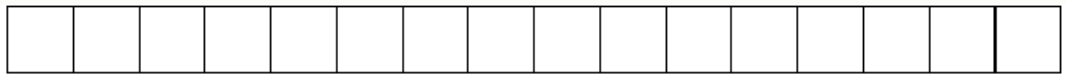
Denis TRYSTRAM

Notes de cours

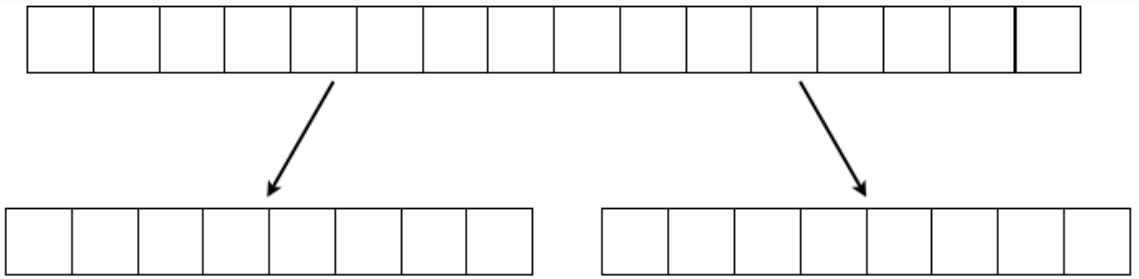
sept. 2023

Exemple préliminaire

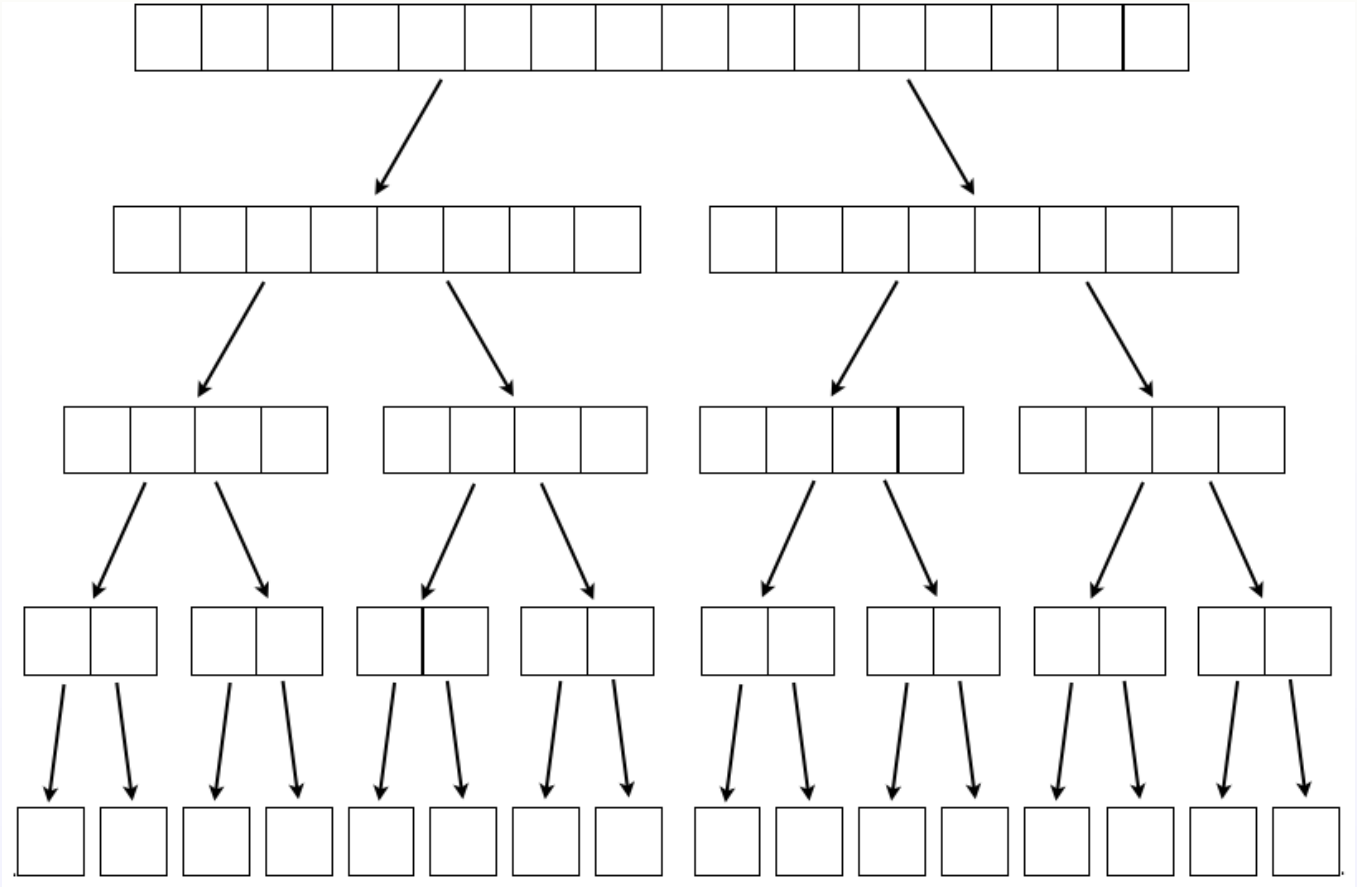
Le tri fusion (merge sort)



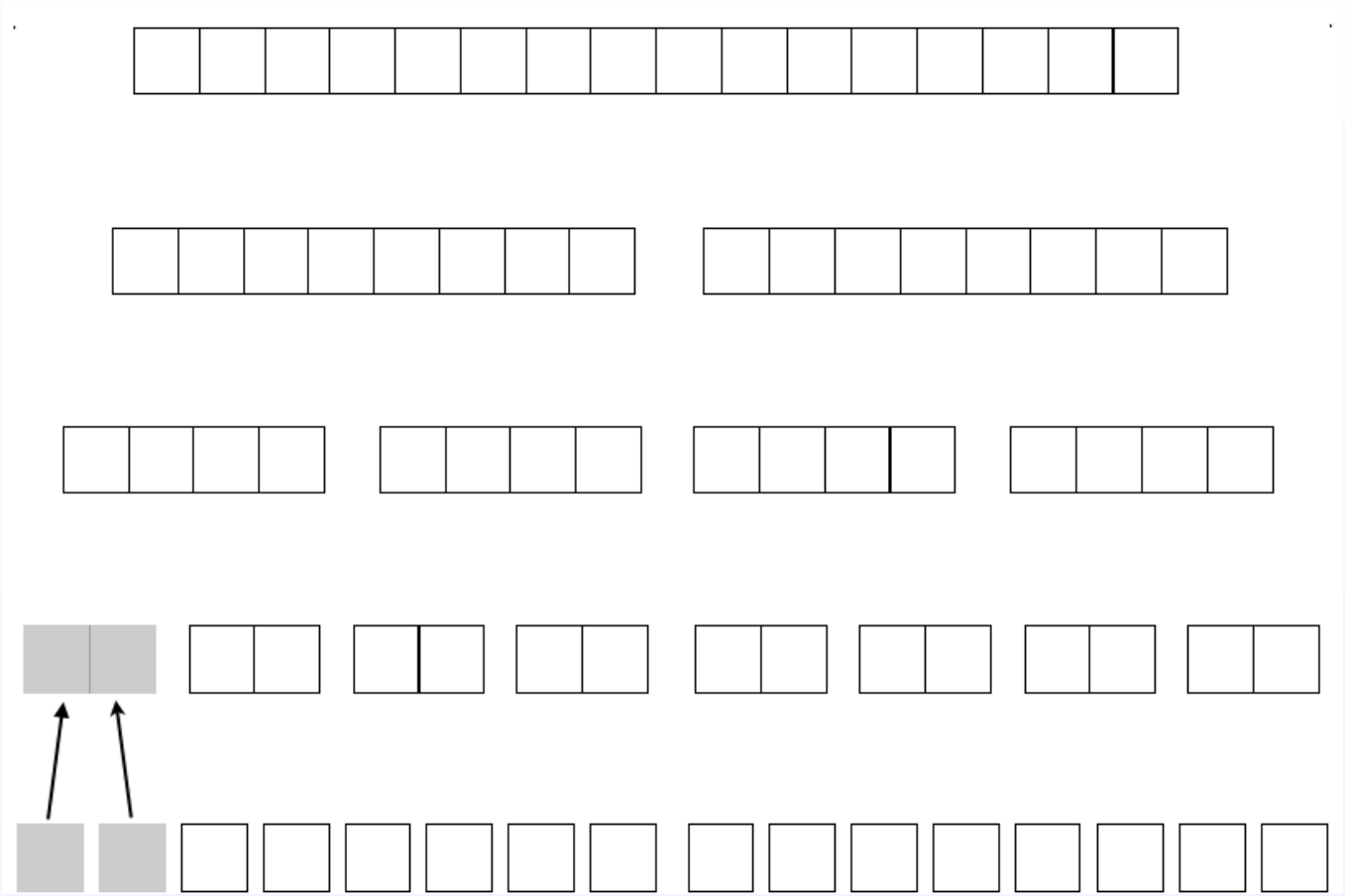
principe

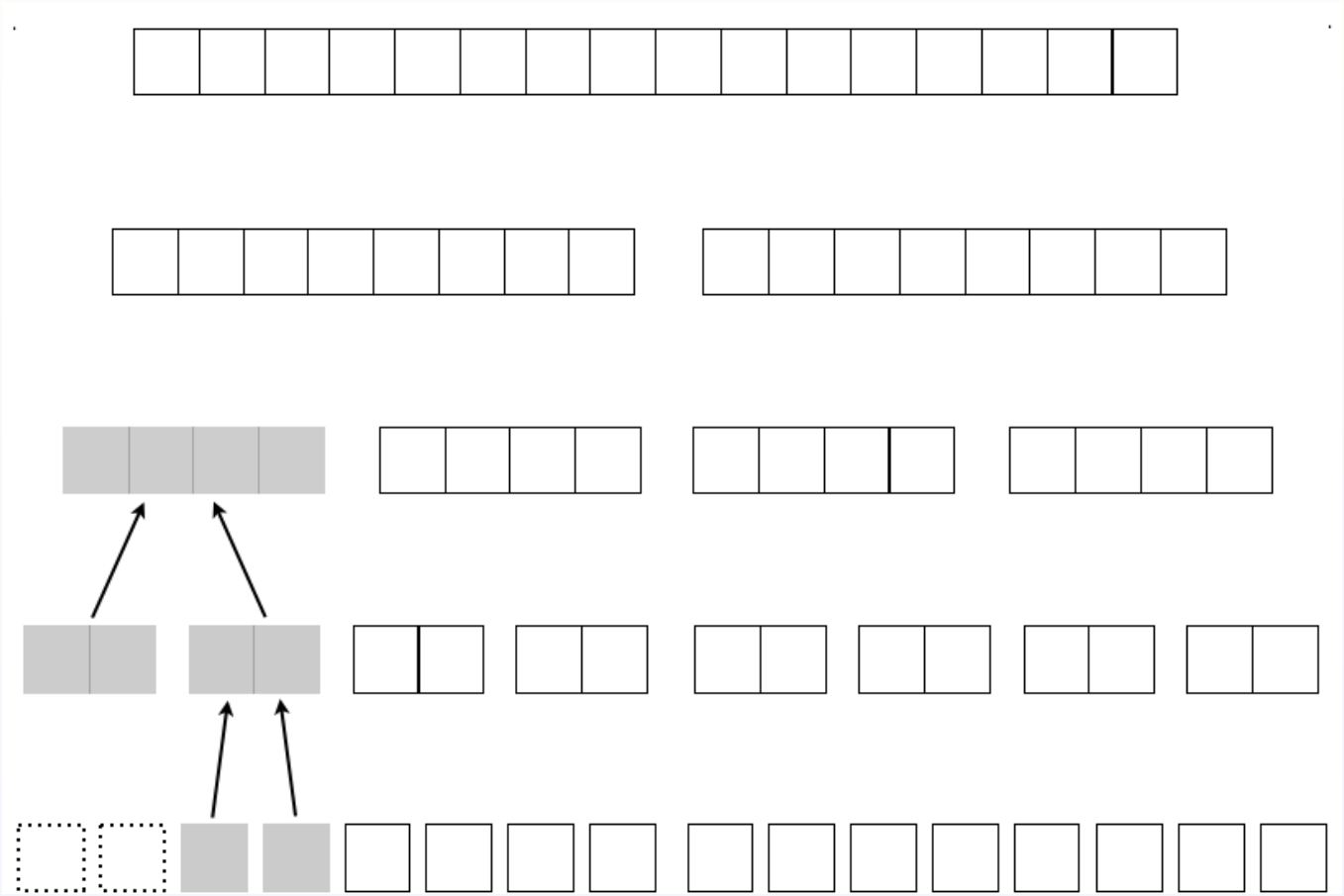


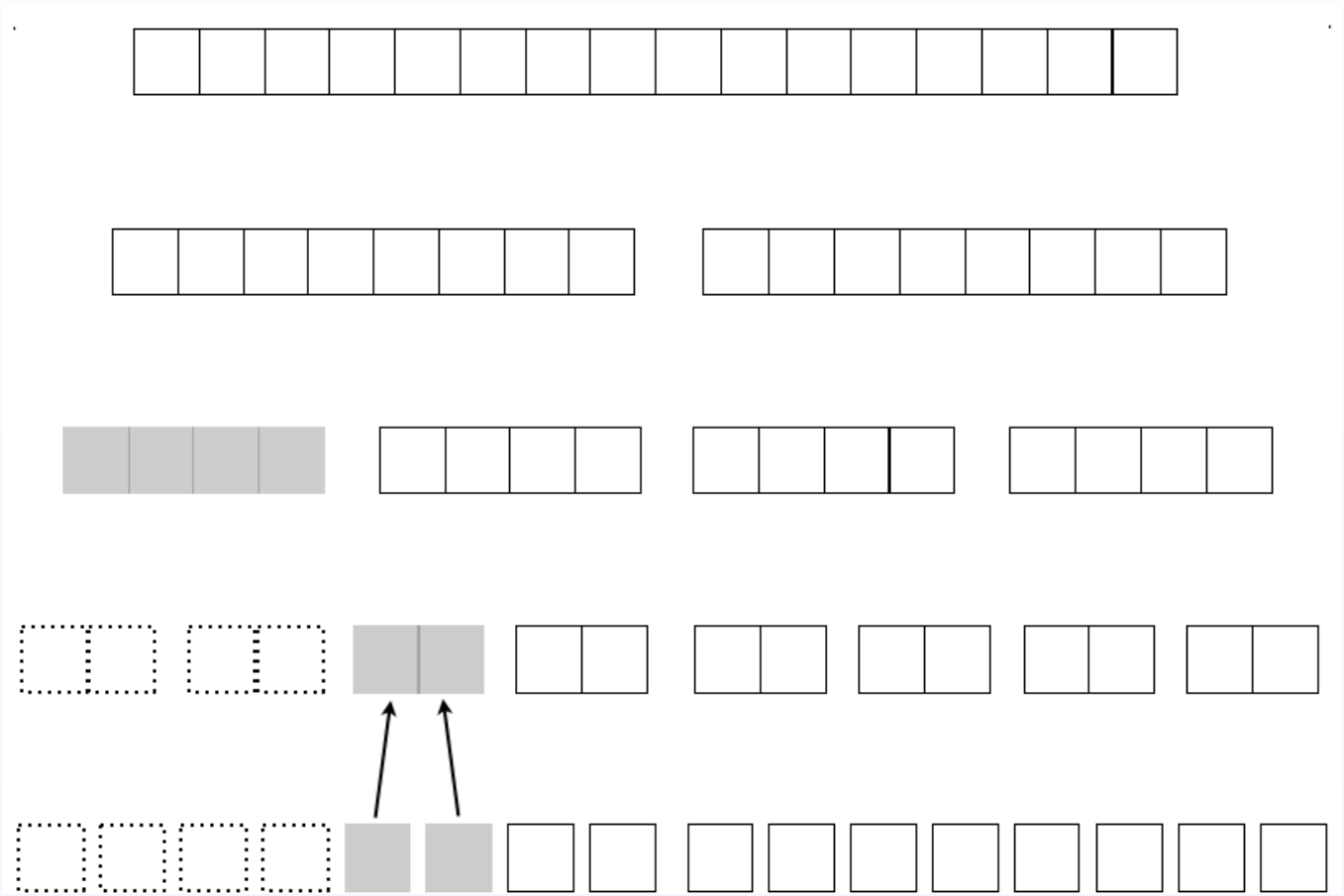
principe

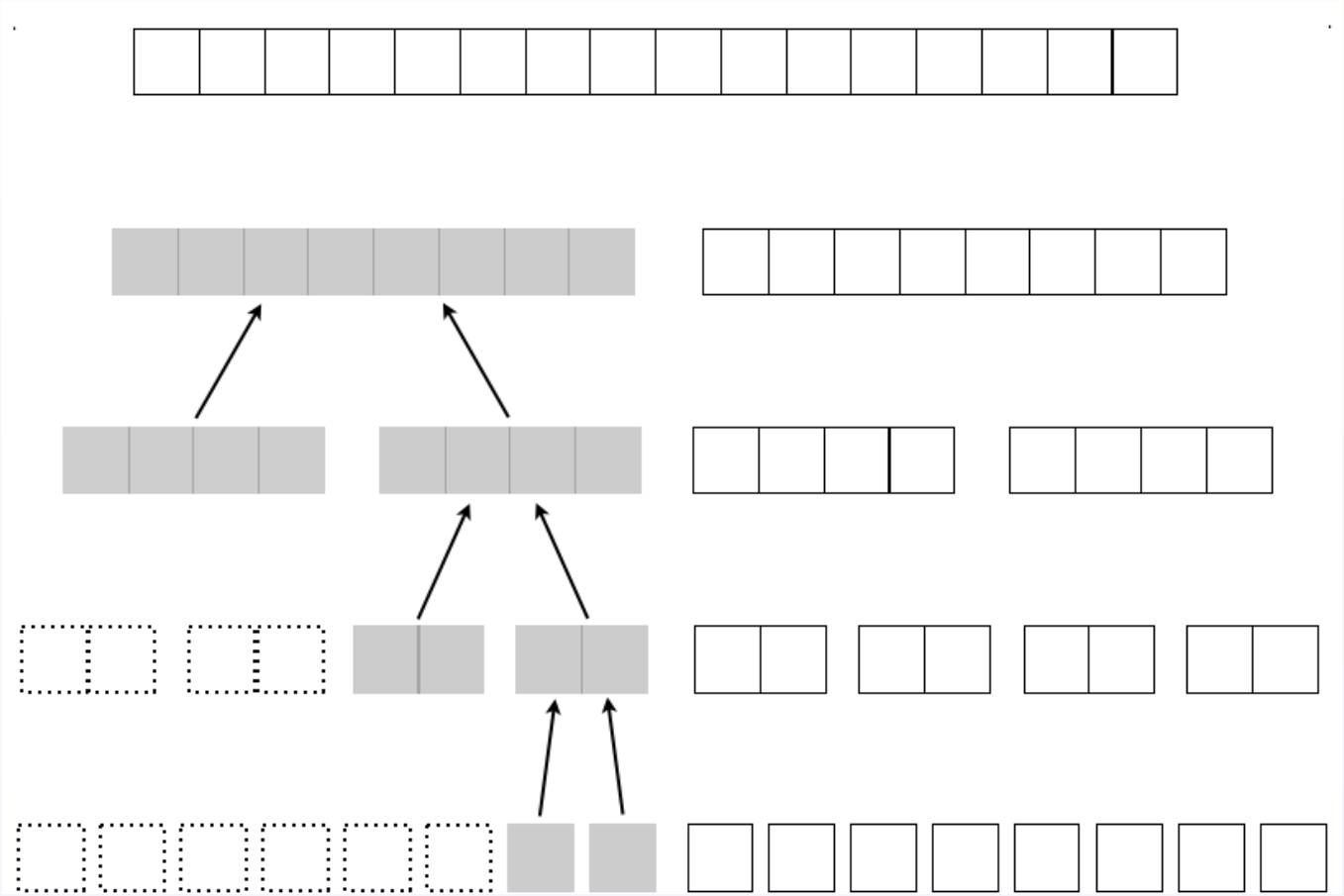


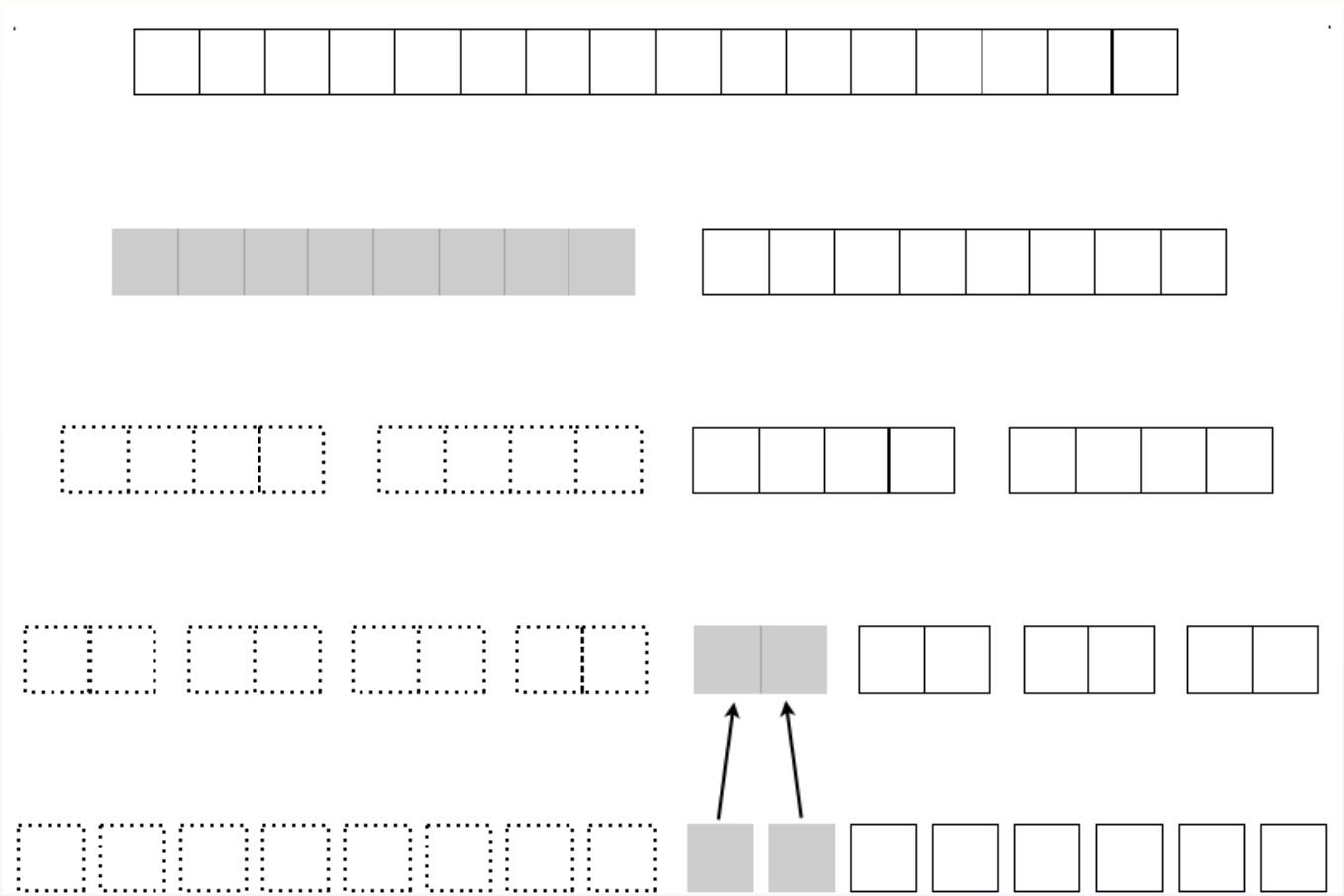
Calculs proprement dits

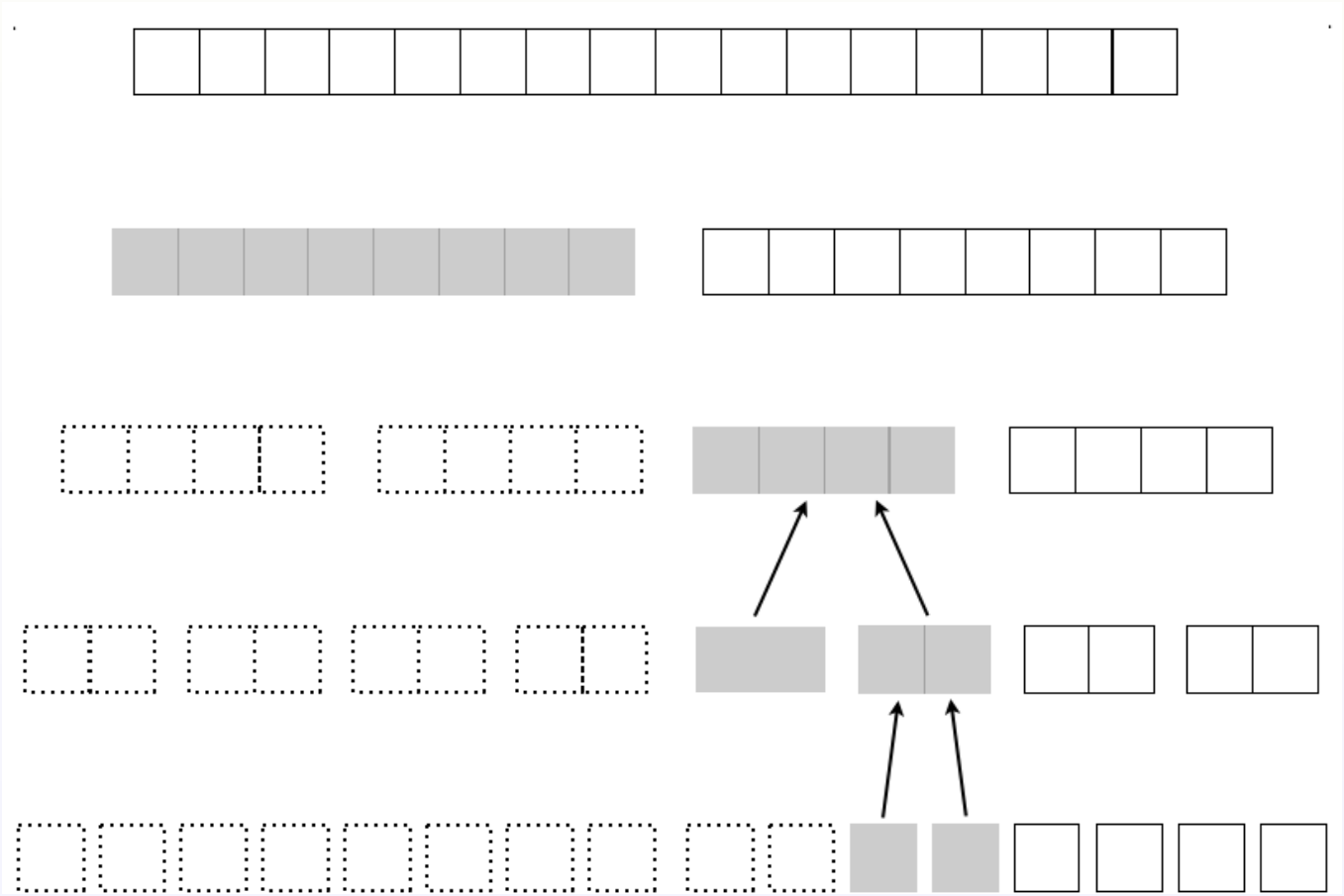












Calcul de coût

- constant par niveau en $\mathcal{O}(n)$ sur $log_2(n)$ niveaux

Principe du diviser pour régner

Motivation

Divide and conquer est un paradigme pour concevoir certains algorithmes pour des problèmes dont on peut décomposer les entrées.

Principe du diviser pour régner

Motivation

Divide and conquer est un paradigme pour concevoir certains algorithmes pour des problèmes dont on peut décomposer les entrées.

Principe pour un problème de taille n:

Si n est "petit" on calcule la solution par n'importe quelle méthode.

Sinon

1. Décomposer le problème en k sous-problèmes de taille n_i.
2. Résoudre les k sous-problèmes de tailles n_i $(1 \leq i \leq k)$.
3. Reconstruire la solution originale à partir des k solutions partielles

Principe du diviser pour régner

Motivation

Divide and conquer est un paradigme pour concevoir certains algorithmes pour des problèmes dont on peut décomposer les entrées.

Principe pour un problème de taille n:

Si n est "petit" on calcule la solution par n'importe quelle méthode.

Sinon

1. Décomposer le problème en k sous-problèmes de taille n_i.
2. Résoudre les k sous-problèmes de tailles n_i $(1 \leq i \leq k)$.
3. Reconstruire la solution originale à partir des k solutions partielles

Généralement, les k sous-problèmes sont résolus par la même mthode (récursivement).

Recursive tree

La méthode détaillée pour l'analyse de coût du tri fusion est généralisable :

Principe :

On construit l'arbre dont les **sommets** sont étiquetés par le coûts des sous-opérations et les **arêtes** correspondent au découpage en sous-problèmes.

Analyse de coût (générale)

n est la taille du problème.
Le coût correspondant est obtenu par la résolution de l'équation de récurrence suivante :

$T(1) \leq$ Cste.
$T(n) = \sum_{1 \leq i \leq k} T(n_i) + c_1(n) + c_3(n)$
où c_1 et c_3 sont les coûts respectifs des phases de décomposition/reconstruction (1) et (3) précédentes.

Analyse de coût simplifiée

Pratiquement, la plupart des méthodes sont des partitionnements en a sous-problèmes identiques ou de mêmes tailles $n_i = \frac{n}{b}$.
L'équation du coût se simplifie alors en :

$T(1) = 1$
$T(n) = a.T(\frac{n}{b}) + c(n)$

Analyse de coût simplifiée

Pratiquement, la plupart des méthodes sont des partitionnements en a sous-problèmes identiques ou de mêmes tailles $n_i = \frac{n}{b}$.
L'équation du coût se simplifie alors en :

$T(1) = 1$
$T(n) = a.T(\frac{n}{b}) + c(n)$

Pour le tri fusion, $a = b = 2$ et $c(n) = n$

Hypothèses pour le calcul

Considérations techniques pour simplifier les calculs :

- En pratique, la taille de l'instance n ne se divise pas exactement en parties égales.
- La recomposition est simplifiée (en ordre de grandeur).
- On simplifie souvent les conditions aux frontières $\mathcal{O}(1)$.

Master Theorem

L'analyse de coût tient dans l'équation de récurrence suivante : $a \geq 1$ et $b > 1$.

- $f(n) = \Theta(1)$ if $n \leq n_0$
- $f(n) = a.f(\frac{n}{b}) + c(n)$ if $n > n_0$

On donne ci-dessous la formulation générale pour résoudre cette équation:

1. if $c(n) \in \mathcal{O}(n^{log_b a - \epsilon})$ then $f(n) \in \Theta(n^{log_b a})$
2. if $c(n) \in \Theta(n^{log_b a})$ then $f(n) \in \Theta(n^{log_b a} log(n))$
3. if $c(n) \in \Omega(n^{log_b a + \epsilon})$ and if $a.c(n/b) \leq kf(n)$ for some constant $k < 1$ then, $f(n) \in \Theta(c(n))$

où ϵ est un nombre réel positif.

Analyse de deux cas simplifiés

On suppose que n est une puissance de b (en d'autres termes, il est parfaitement divisible par b jusqu'à atteindre 1).

- $f(1) = 1$
- $f(n) = a.f(\frac{n}{b}) + c$ (c est une constante positive)

- $f(1) = 1$
- $f(n) = a.f(\frac{n}{b}) + n$

Cas le plus simple

La fonction f est une simple récurrence linéaire. Dans ce cas, la solution de f en fonction de n est régi par l'équation simplifiée

$$\begin{aligned} f(n) &= (1+\log_b n)\cdot c && \text{if } a=1 \\ &= \frac{1-a^{\log_b n}}{1-a}\cdot c \approx \frac{c}{1-a} && \text{if } a<1 \\ &= \frac{a^{\log_b n}-1}{a-1}\cdot c && \text{if } a>1 \end{aligned} \tag{1}$$

preuve

Ecrivons l'expansion des calculs en remplaçant les occurences de $f(\cdot)$.
Dès que l'on a identifié un motif régulier, on peut déduire la forme générale :

$$\begin{aligned} f(n) &= af(n/b) + c \\ &= a\left(af(n/b^2) + c\right) + c \end{aligned}$$

preuve

Ecrivons l'expansion des calculs en remplaçant les occurences de $f(\cdot)$.
Dès que l'on a identifié un motif régulier, on peut déduire la forme générale :

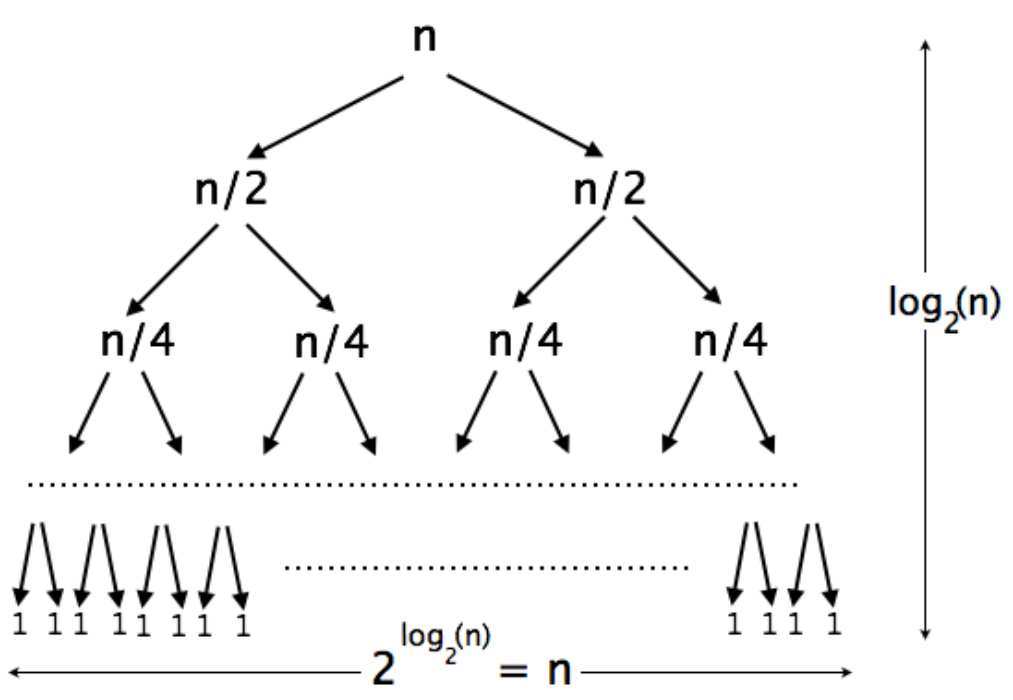
$$\begin{aligned} f(n) &= af(n/b) + c \\ &= a\left(af(n/b^2) + c\right) + c \\ &= a^2 f(n/b^2) + (a+1)c \\ &= a^2\left(af(n/b^3) + c\right) + (a+1)c \end{aligned}$$

preuve

Ecrivons l'expansion des calculs en remplaçant les occurences de $f(\cdot)$.
Dès que l'on a identifié un motif régulier, on peut déduire la forme générale :

$$\begin{aligned} f(n) &= af(n/b) + c \\ &= a\left(af(n/b^2) + c\right) + c \\ &= a^2 f(n/b^2) + (a+1)c \\ &= a^2\left(af(n/b^3) + c\right) + (a+1)c \\ &= a^3 f(n/b^3) + (a^2 + a + 1)c \\ &\vdots \quad \vdots \\ &= \left(a^{\log_b n} + \cdots + a^2 + a + 1\right) c \end{aligned} \tag{2}$$

Une vue graphique pour $a = b = 2$



Extension: coût linéaire

On emploie une fonction de reconstruction $c(n) = n$.

Equation

$$f(n) = a^{\log_b n} f(1) + \left(\sum_{i=0}^{\log_b(n)-1} (a/b)^i \right) n$$

Extension: coût linéaire

On emploie une fonction de reconstruction $c(n) = n$.

Equation

$$f(n) \;=\; a^{\log_b n} f(1) \;+\; \left(\sum_{i=0}^{\log_b(n)-1} (a/b)^i \right) n$$

Si $a > b$, le comportement de $f(n)$ est dominé par le premier terme.

$$a^{\log_b n} \cdot f(1) \;=\; n^{\log_b a}$$

Si $a < b$, $f(n)$ est dominé par le second.

$$n \cdot \sum_{i=0}^{\log_b(n)-1} (a/b)^i \;=\; \frac{\left(1 \;-\; (a/b)^{\log_b(n)}\right)}{1-(a/b)} \cdot n \;\approx\; \frac{b}{b-a} \cdot n$$

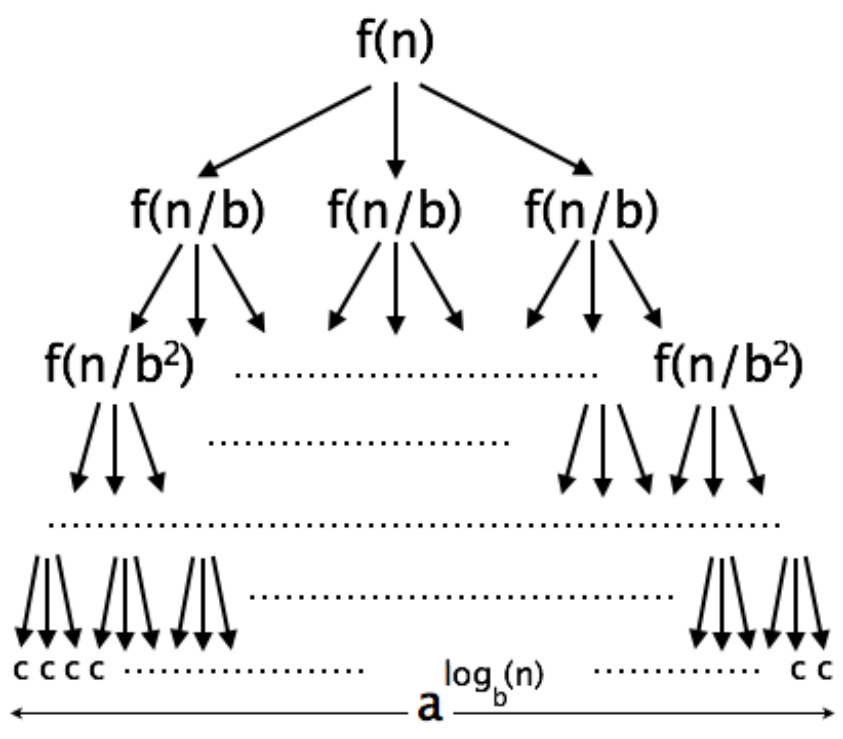
preuve

De même que précédemment, on fait une extension des calculs et on en déduit le résultat que l'on vérifie par récurrence.

$$
\begin{aligned}
f(n) &= af(n/b) + n \\
&= a\left(af(n/b^2) + n/b\right) + n \\
&= a^2 f(n/b^2) + (an/b + n) \\
&= a^2\left(af(n/b^3) + n/b^2\right) + (a/b + 1)n \\
&= a^3 f(n/b^3) + (a^2/b^2 + a/b + 1)n \\
&\vdots \quad \vdots \\
&= a^{\log_b n} f(1) + \left(\sum_{i=0}^{\log_b(n)-1} (a/b)^i\right) n
\end{aligned}
$$

Recursive tree pour a et b quelconques

$$f(n) = a.f(\frac{n}{b}) + c(n)$$



Interprétation du master Theorem

Intuitivement, le théorème nous dit que $f(n)$ est déterminé d'une des fonctions selon la granularité de la décomposition et le processus de recomposition.

- if $n^{log_b a}$ dominates, then, the solution is in $\Theta(n^{log_b a})$.
- if this is the contrary, the solution is in $\Theta(f(n))$.
- if it is well-balanced, the solution is in $\Theta(f(n).log(n))$

Applications

- Merge sort (Tri Fusion)
- Enveloppe Convexe
- Karatsuba