Algorithmique: théorie, tris, complexité, en 2 heures.

Emily Clement

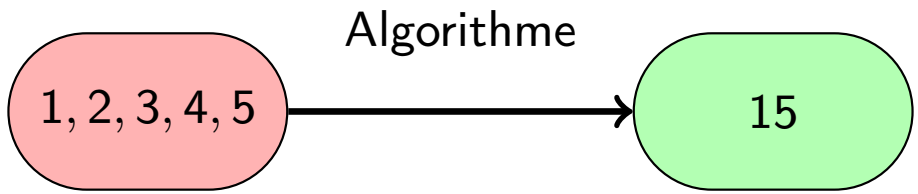
OOP - M1 Bio-Informatique

14 Novembre 2019

• Sommaire du cours

- **C'est quoi un algorithme ?**

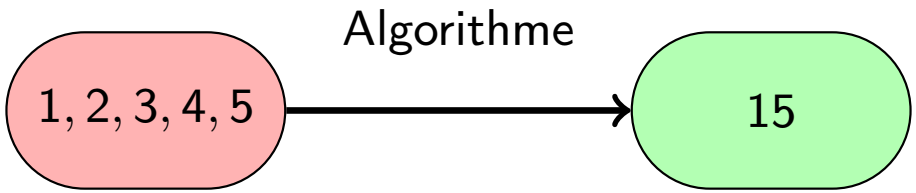
Ex : Algorithme de somme



Retourner $1 + 2 + 3 + 4 + 5$

• C'est quoi un algorithme ?

Ex : Algorithme de somme



Retourner $1 + 2 + 3 + 4 + 5$

Def : Algorithme [Source : Wikipédia]

suite d'opération **non ambiguë** pour **résoudre** une **classe de problème**

• Exemple de problèmes

Sur des nombres

Sachant une suite de nombres $n_1, , \cdots , n_N$, **calculer...**

- la somme $\sum_{i=1}^{N} n_i$

• Exemple de problèmes

Sur des nombres

Sachant une suite de nombres $n_1, , \cdots , n_N$, **calculer...**

- la somme $\sum_{i=1}^{N} n_i$
- le produit $\prod_{i=1}^{N} n_i$

• Exemple de problèmes

Sur des nombres

Sachant une suite de nombres $n_1, , \cdots , n_N$, **calculer...**

- la somme $\sum_{i=1}^{N} n_i$
- le produit $\prod_{i=1}^{N} n_i$
- une liste **triée** de $n_1, \cdots , n_N$

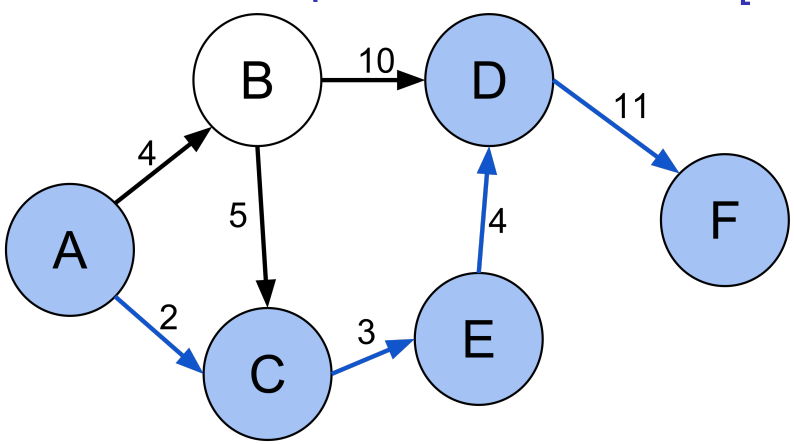
• Exemple de problèmes

Sur des nombres

Sachant une suite de nombres $n_1, , \cdots , n_N$, **calculer...**

- la somme $\sum_{i=1}^{N} n_i$
- le produit $\prod_{i=1}^{N} n_i$
- une liste **triée** de $n_1, \cdots , n_N$

Problème du plus court chemin [Source : Wikipedia]



Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée :
- Sortie :

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie :

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie :

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Exemples : Chez le docteur

- Entrée : Un patient.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Exemples : Chez le docteur

- Entrée : Un patient.
- Sortie : Retourner **Vrai** si il est malade, **Faux** sinon.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Exemples : Chez le docteur

- Entrée : Un patient.
- Sortie : Retourner **Vrai** si il est malade, **Faux** sinon.

Le problème de la plus courte sur-séquence commune (SSC)

- Entrée : une collection F de mots sur l'alphabet $\{A, C, G, T\}$.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Exemples : Chez le docteur

- Entrée : Un patient.
- Sortie : Retourner **Vrai** si il est malade, **Faux** sinon.

Le problème de la plus courte sur-séquence commune (SSC)

- Entrée : une collection F de mots sur l'alphabet $\{A, C, G, T\}$.
- Sortie : Un mot S de longueur minimale, tel que tout mot f de F soit un **facteur** de S. *i.e* la plus courte **sur-séquence** commune de F.

Quizz : exemple tirés des TPs

Feuille 1 : The next second

- Entrée : L'heure (h, m, s).
- Sortie : L'heure à la seconde suivante.

Feuille 1 : Division euclidienne

- Entrée : Deux nombres a, b entiers.
- Sortie : Le reste r et le quotient q de la division euclidienne de a par b.

Exemples : Chez le docteur

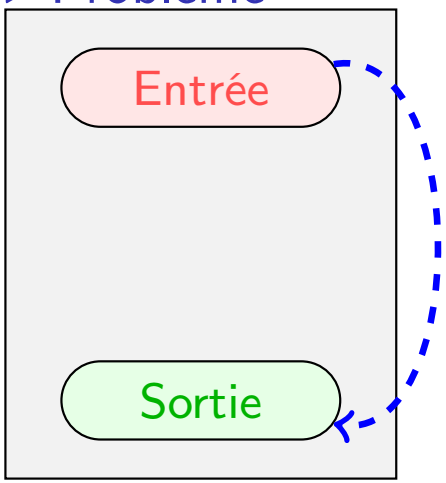
- Entrée : Un patient.
- Sortie : Retourner **Vrai** si il est malade, **Faux** sinon.

Le problème de la plus courte sur-séquence commune (SSC)

- Entrée : une collection F de mots sur l'alphabet $\{A, C, G, T\}$.
- Sortie : Un mot S de longueur minimale, tel que tout mot f de F soit un **facteur** de S. *i.e* la plus courte **sur-séquence** commune de F.
- Exemple : $F = \{CTA, ACT, AGT\}$, alors .

• Quelques définitions formelles

▷ Problème



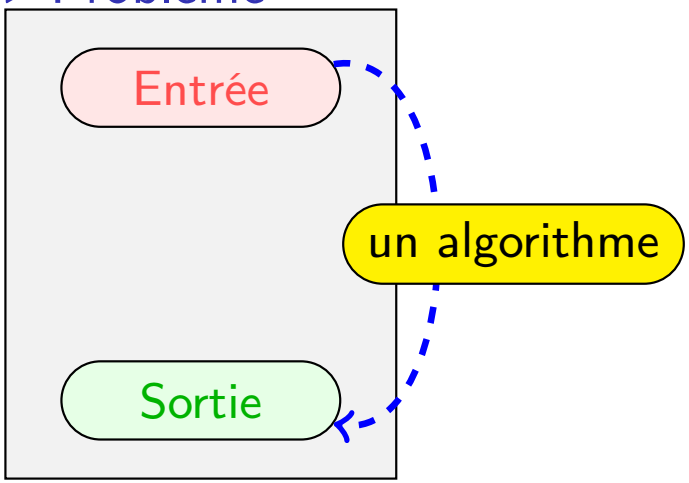
Remarque : Il peut ne pas exister d'algorithmes...

▷ L'entrée d'un problème

- Une instance d'un problème est une entrée **satisfaisant les contraintes du problème**.
- *Ex* : Contraintes de type.

• Quelques définitions formelles

▷ Problème



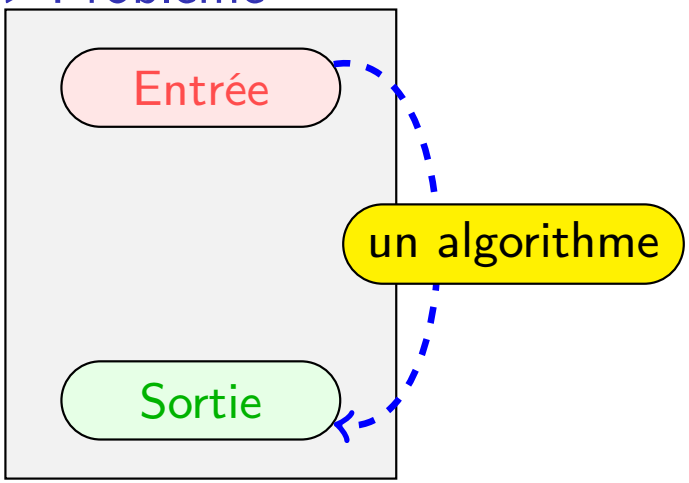
Remarque : Il peut ne pas exister d'algorithmes...

▷ L'entrée d'un problème

- Une instance d'un problème est une entrée **satisfaisant les contraintes du problème**.
- *Ex* : Contraintes de type.

• Quelques définitions formelles

▷ Problème



Remarque : Il peut ne pas exister d'algorithmes...

▷ Types de problème

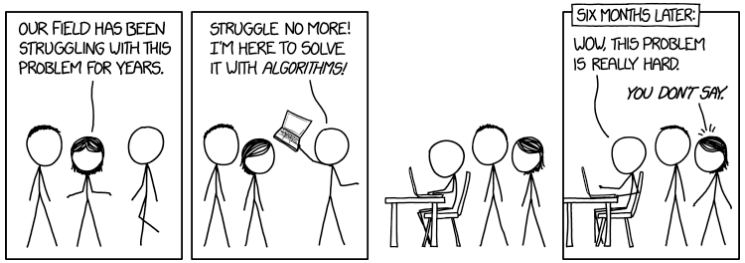
- Sortie = Oui/Non. C'est un problème de décision.
- Sinon, c'est un problème d'évaluation.
- *Ex* : Malade/Pas Malade ≠ "Vous avez la *nom de maladie*".

▷ L'entrée d'un problème

- Une instance d'un problème est une entrée **satisfaisant les contraintes du problème**.
- *Ex* : Contraintes de type.

- C'est quoi un problème difficile ?

  C'est parfois plus compliqué qu'on ne le pense...

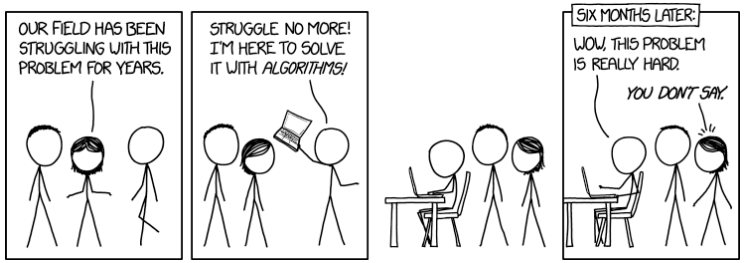


Source : xkcd about algorithms

1. Alan turing, 1936

- C'est quoi un problème difficile ?

C'est parfois plus compliqué qu'on ne le pense...



Source : xkcd about algorithms

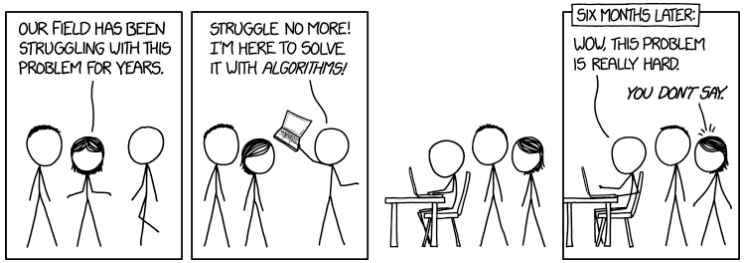
Décidabilité algorithmique

- Un problème de décision est dit **décidable** s'il existe un algorithme qui le résoud.
- S'il n'en existe pas, le problème est dit **indécidable.**

1. Alan turing, 1936

- C'est quoi un problème difficile ?

C'est parfois plus compliqué qu'on ne le pense...



Source : xkcd about algorithms

Décidabilité algorithmique

- Un problème de décision est dit **décidable** s'il existe un algorithme qui le résoud.
- S'il n'en existe pas, le problème est dit **indécidable.**

Ex : Problème de l'arrêt

- Entrée : Un programme informatique P, une entrée I de P.
- Sortie : **Vrai** si le programme, avec l'entrée I, termine **Faux** sinon.

Le problème de l'arrêt est indécidable [1].

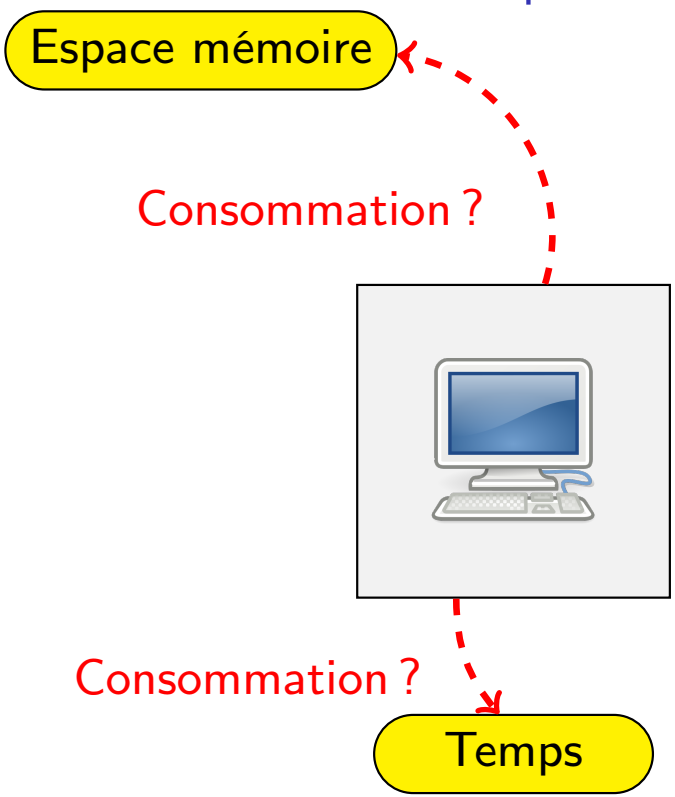
1. Alan turing, 1936

Pause

Pause !

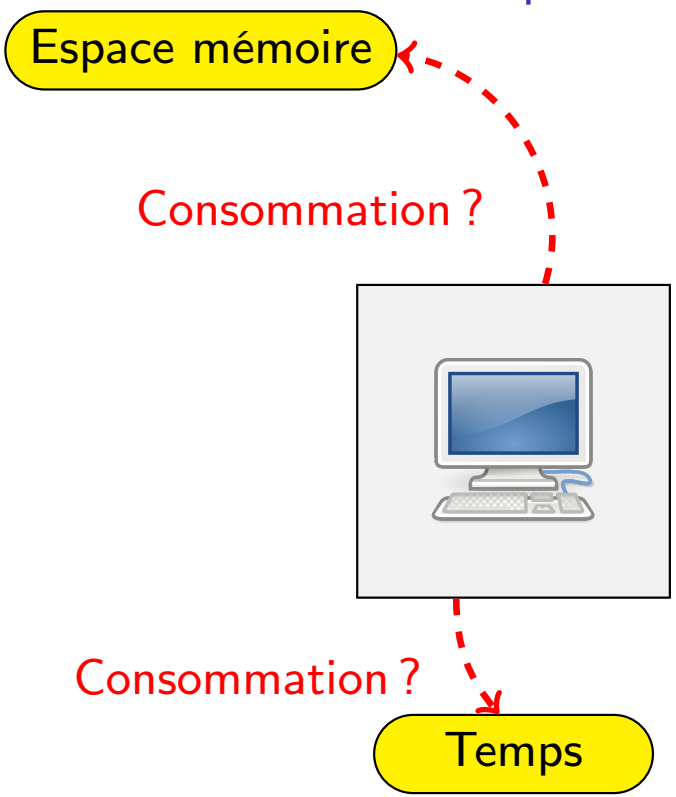
• Complexité ? C'est quoi ?

▷ La théorie de la complexité



• Complexité ? C'est quoi ?

▷ La théorie de la complexité

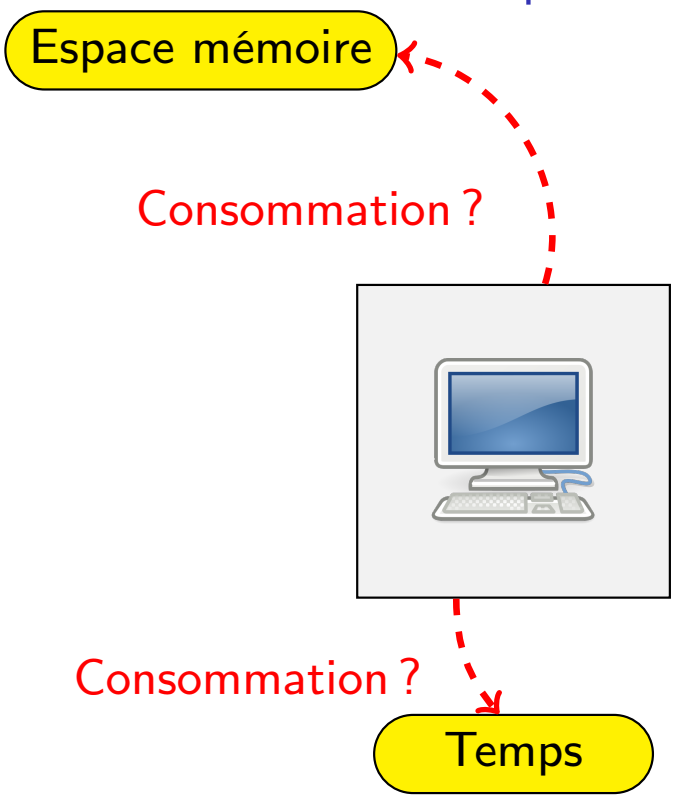


▷ Complexité (en temps) d'un problème/algorithme

- Quelle efficacité (en temps) a mon algorithme ?
- On "compte" les opérations effectuées.
- Mon problème peut-il être résolu en au plus/environ/au moins X opérations ?

• Complexité ? C'est quoi ?

▷ La théorie de la complexité



▷ Complexité (en temps) d'un problème/algorithme

- Quelle efficacité (en temps) a mon algorithme ?
- On "compte" les opérations effectuées.
- Mon problème peut-il être résolu en au plus/environ/au moins X opérations ?

▷ Mots-clés

Pire cas, Moyen cas, Nombre d'opérations élémentaires

• Comment on calcule la complexité d'un algorithme ?

▷ Analyser un algorithme

- n : taille des données
- Unité : nombres d'opérations élémentaires. On notera le coût total $T(n)$.
- On va considérer le pire ou le moyen cas.

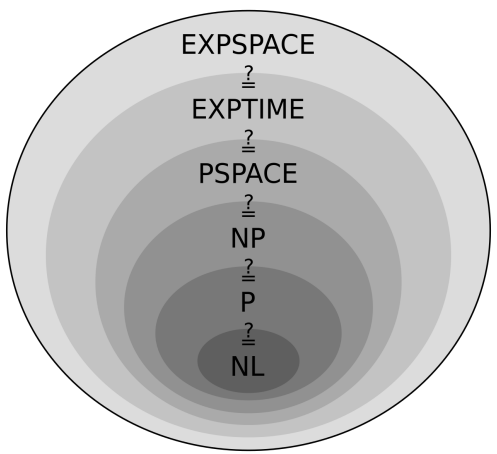
▷ Pire cas/Moyen cas.

- Complexité dans le pire cas : temps d'exécution le plus **long** possible de l'algorithme.
- Complexité dans le moyen cas : Moyenne des complexités. (Raffinements possibles, cf Wikipédia)

▷ Comparer : les ordres de grandeur

- $T(n) = \mathcal{O}(f(n))$: Majoré asymptotiquement à une cste près.

▷ Classe de complexité [Source : Wikipedia]



Opération pour calculer la complexité

▷ Séquences d'opérations : Somme

$$\left.\begin{array}{ll} \text{Op 1} & \text{Coût } T_1(n) \\ \text{Op 2} & \text{Coût } T_2(n) \end{array}\right\} \Rightarrow T(n) = T_1(n) + T_2(n)$$

Opération pour calculer la complexité

▷ Séquences d'opérations : Somme

$$\left.\begin{array}{ll} \text{Op 1} & \text{Coût } T_1(n) \\ \text{Op 2} & \text{Coût } T_2(n) \end{array}\right\} \Rightarrow T(n) = T_1(n) + T_2(n)$$

▷ Embranchement (If) : Maximum

$$\left.\begin{array}{ll} \text{Si (Condition) alors :} & \text{Coût } T_{cd}(n) \\ \text{Op 1} & \text{Coût } T_1(n) \\ \text{Sinon :} & \\ \text{Op 2} & \text{Coût } T_2(n) \end{array}\right\} \Rightarrow T(n) = T_{cd}(n) + \max\left[T_1(n), T_2(n)\right]$$

Opération pour calculer la complexité

▷ Séquences d'opérations : Somme

$$\left.\begin{array}{ll} \text{Op 1} & \text{Coût } T_1(n) \\ \text{Op 2} & \text{Coût } T_2(n) \end{array}\right\} \Rightarrow T(n) = T_1(n) + T_2(n)$$

▷ Embranchement (If) : Maximum

$$\left.\begin{array}{ll} \text{Si (Condition) alors :} & \text{Coût } T_{cd}(n) \\ \text{Op 1} & \text{Coût } T_1(n) \\ \text{Sinon :} & \\ \text{Op 2} & \text{Coût } T_2(n) \end{array}\right\} \Rightarrow T(n) = T_{cd}(n) + \max\left[T_1(n), T_2(n)\right]$$

▷ Boucle While : Somme

$$\left.\begin{array}{ll} \text{Tant que (Cond) faire :} & T_{cd}(n) \\ \text{Operation} & T_i(n) \end{array}\right\} \Rightarrow T(n) = T_{cd}(n) + T_1(n) + \cdots + T_K(n)$$

- $T_i(n)$: cours de la $i-$ème itération de la boucle.
- On doit déterminer K, le nombre d'opérations faite par la boucle while.
- Même raisonnement pour les boucles "**For** i in $\{0 \cdots K-1\}$", sauf qu'on connaît K !

Opération pour calculer la complexité (suite)

▷ Exemple de fonction récursive $F(N)$

Si $(N > 1)$ faire :	Coût $\mathcal{O}(1)$
$F(N/2)$	Coût $T(n/2)$
Operation	Coût $C(n)$
$F(N/2)$	Coût $T(n/2)$

$$\Rightarrow T(n) = \mathcal{O}(1) + 2 \cdot T(n/2) + C(n)$$

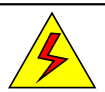
Opération pour calculer la complexité (suite)

▷ Exemple de fonction récursive $F(N)$

Si $(N > 1)$ faire :	Coût $\mathcal{O}(1)$
$F(N/2)$	Coût $T(n/2)$
Operation	Coût $C(n)$
$F(N/2)$	Coût $T(n/2)$

$$\Rightarrow T(n) = \mathcal{O}(1) + 2 \cdot T(n/2) + C(n)$$

▷ Classes de complexité

$\mathcal{O}(1)$	Temps constant
$\mathcal{O}(\log n)$	Logarithmique
$\mathcal{O}(n)$	Temps linéaire
$\mathcal{O}(n \log n)$	Temps quasi-linéaire
$\mathcal{O}(n^2)$	Temps quadratique
$\mathcal{O}(P), P \in \mathbb{R}[X]$	Temps polynomial
$\mathcal{O}(2^n)$	Exponentiel 

Exemple : Conversion d'un entier en chaine de caractère

▷ TP - Jeu

Conversion d'un entier en chaine de caractère

```
1: procedure Conversion(n)
2:     int ← n                          ▷ Coût O(1)
3:     Str ← String.valueOf(n)          ▷ Coût O(log(n))
4:     return Str                       ▷ Coût O(1)
5: end procedure
```

Exemple : Conversion d'un entier en chaine de caractère

▷ TP - Jeu

Conversion d'un entier en chaine de caractère

```
1: procedure Conversion(n)
2:     int ← n                          ▷ Coût O(1)
3:     Str ← String.valueOf(n)          ▷ Coût O(log(n))
4:     return Str                       ▷ Coût O(1)
5: end procedure
```

▷ Coût total

$T(n) = ?$

Exemple : Conversion d'un entier en chaine de caractère

▷ TP - Jeu

Conversion d'un entier en chaine de caractère

```
1: procedure Conversion(n)
2:     int ← n                          ▷ Coût O(1)
3:     Str ← String.valueOf(n)          ▷ Coût O(log(n))
4:     return Str                       ▷ Coût O(1)
5: end procedure
```

▷ Coût total

$$T(n) = \mathcal{O}(1) + \mathcal{O}(\log(n)) + \mathcal{O}(1) = \mathcal{O}(\log(n))$$

Exemple : Recherche dans une liste quelconque

Sachant un élément elt, et une liste L, on veut savoir si elt est un élément de la liste L.

▷ Recherche d'un élément dans une liste

Recherche d'un élément elt dans une liste L de taille n

```
1: procedure Recherche(elt, L, n)
2:     i ← 0                                          ▷ Coût O(1)
3:     while i < n − 1 AND L[i] ≠ elt do              ▷ Coût O(1), X fois
4:         i ← i + 1                                  ▷ Coût O(1), X fois
5:     end while
6:     return L[i] == elt                             ▷ Coût O(1)
7: end procedure
```

Exemple : Recherche dans une liste quelconque

Sachant un élément elt, et une liste L, on veut savoir si elt est un élément de la liste L.

▷ Recherche d'un élément dans une liste

Recherche d'un élément elt dans une liste L de taille n

```
1: procedure Recherche(elt, L, n)
2:     i ← 0                                          ▷ Coût O(1)
3:     while i < n − 1 AND L[i] ≠ elt do              ▷ Coût O(1), X fois
4:         i ← i + 1                                  ▷ Coût O(1), X fois
5:     end while
6:     return L[i] == elt                             ▷ Coût O(1)
7: end procedure
```

▷ Coût total dans le pire cas

$T(n) = ?$

Exemple : Recherche dans une liste quelconque

Sachant un élément elt, et une liste L, on veut savoir si elt est un élément de la liste L.

▷ Recherche d'un élément dans une liste

Recherche d'un élément elt dans une liste L de taille n

```
1: procedure Recherche(elt, L, n)
2:     i ← 0                                          ▷ Coût O(1)
3:     while i < n − 1 AND L[i] ≠ elt do              ▷ Coût O(1), X fois
4:         i ← i + 1                                  ▷ Coût O(1), X fois
5:     end while
6:     return L[i] == elt                             ▷ Coût O(1)
7: end procedure
```

▷ Coût total dans le pire cas

$$T(n) = \quad \mathcal{O}(1) + 2n \cdot \mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(n)$$

Exemple : Recherche dichotomique

▷ Recherche dichotomique dans un tableau trié $L\langle L_0, \cdots, L_n\rangle$

Recherche d'un élément elt dans un tableau trié L

```
1:  procedure Recherche(elt, L = ⟨L_0, ··· , L_n⟩)
2:      i ← 0
3:      result ← n + 1
4:      while i < result − 1 do
5:          if elt > L_{(i+result)/2} then
6:              i ← (i + result) /2
7:          else
8:              result = (i + result) /2
9:          end if
10:     end while
11:     return result
12: end procedure
```

Exemple : Recherche dichotomique

▷ Recherche dichotomique dans un tableau trié $L\langle L_0, \cdots, L_n\rangle$

Recherche d'un élément elt dans un tableau trié L

```
1:  procedure Recherche(elt, L = ⟨L_0, ··· , L_n⟩)
2:      i ← 0
3:      result ← n + 1
4:      while i < result − 1 do
5:          if elt > L_{(i+result)/2} then
6:              i ← (i + result) /2
7:          else
8:              result = (i + result) /2
9:          end if
10:     end while
11:     return result
12: end procedure
```

▷ Coût total en fonction de $T(n/2)$

$T(n) = ?$

Exemple : Recherche dichotomique

▷ Recherche dichotomique dans un tableau trié $L\langle L_0, \cdots, L_n\rangle$

Recherche d'un élément elt dans un tableau trié L

```
1:  procedure Recherche(elt, L = ⟨L_0, ··· , L_n⟩)
2:      i ← 0
3:      result ← n + 1
4:      while i < result − 1 do
5:          if elt > L_{(i+result)/2} then
6:              i ← (i + result) /2
7:          else
8:              result = (i + result) /2
9:          end if
10:     end while
11:     return result
12: end procedure
```

▷ Coût total en fonction de $T(n/2)$

$T(n) = \quad 1 + T(n/2)$

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$

On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$
On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

▷ Substitution

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$
On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

▷ Substitution

- Hyp : $T(n) = a \cdot \log_2 n + c$

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$
On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

▷ Substitution

- Hyp : $T(n) = a \cdot \log_2 n + c$
- Remettre ça dans $T(n/2) = a \cdot \log_2 n - a + c$

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$
On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

▷ Substitution

- Hyp : $T(n) = a \cdot \log_2 n + c$
- Remettre ça dans $T(n/2) = a \cdot \log_2 n - a + c$
- Résoudre : $a \log_2 n + c - 1 = a \log_2 n - a + c$ et conclure, $a = 1$ et $c = 1$.

Exemple : Recherche dichotomique (suite)

▷ Comment trouver la complexité ?

$T(n) = 1 + T(n/2)$ et $T(1) = 1$
On voudrait dire que $T(n) = \mathcal{O}(\log_2 n)$, comment faire ?

▷ Substitution

- Hyp : $T(n) = a \cdot \log_2 n + c$
- Remettre ça dans $T(n/2) = a \cdot \log_2 n - a + c$
- Résoudre : $a \log_2 n + c - 1 = a \log_2 n - a + c$ et conclure, $a = 1$ et $c = 1$.

Méthodes générales

- D'autres méthodes existent

Pause

Pause !

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	-3	101	0

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	−3	101	0

- Sortie :

−3	0	1	5	10	101

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	−3	101	0

- Sortie :

−3	0	1	5	10	101

▷ Algorithmes de tri

- Beaucoup d'algorithmes de tri différents existent

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	−3	101	0

- Sortie :

−3	0	1	5	10	101

▷ Algorithmes de tri

- Beaucoup d'algorithmes de tri différents existent
- On verra le tri par **insertion** et le tri **fusion**.

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	−3	101	0

- Sortie :

−3	0	1	5	10	101

▷ Algorithmes de tri

- Beaucoup d'algorithmes de tri différents existent
- On verra le tri par **insertion** et le tri **fusion**.
- Dans le cas moyen, on ne peut pas trier avec des comparaisons plus rapidement qu'en $\mathcal{O}(n \log n)$

• Problème du tri

▷ Problème du tri d'un tableau de nombres

- Entrée Une suite de n nombres $\langle a_0, \cdots, a_{n-1} \rangle$, qu'on peut noter $\boldsymbol{a}$
- Sortie Une permutation (réorganisation) $\langle a'_0, \cdots, a'_{n-1} \rangle$ de la suite $\boldsymbol{a}$ de façon que $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

▷ Exemple

- Entrée :

1	5	10	−3	101	0

- Sortie :

−3	0	1	5	10	101

▷ Algorithmes de tri

- Beaucoup d'algorithmes de tri différents existent
- On verra le tri par **insertion** et le tri **fusion**.
- Dans le cas moyen, on ne peut pas trier avec des comparaisons plus rapidement qu'en $\mathcal{O}(n \log n)$
- Certains tris sont mieux que d'autres selon la situation !

- Tri par insertion : Intuition
  - ▷ Le tri par insertion
    - Intuition : Le tri de carte. Voici une animation !
    - Complexité : pas optimal...

- Tri par insertion : Intuition
  - ▷ Le tri par insertion
    - Intuition : Le tri de carte. Voici une animation !
    - Complexité : pas optimal...
  - ▷ Présentation du tri par insertion

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do
3:          clé ← L[j]
4:          i ← j − 1
5:          while i ≥ 0 AND L[i] > clé do    ▷ Insère L[j] dans la séquence triée L[0, · · · , j − 1]
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do
3:          clé ← L[j]
4:          i ← j − 1
5:          while i ≥ 0 AND L[i] > clé do
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]
4:          i ← j − 1
5:          while i ≥ 0 AND L[i] > clé do
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1
5:          while i ≥ 0 AND L[i] > clé do
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1                                     ▷ O(1)
5:          while i ≥ 0 AND L[i] > clé do
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1                                     ▷ O(1)
5:          while i ≥ 0 AND L[i] > clé do                 ▷ Pire cas : j − 1 fois
6:              L[i + 1] ← L[i]
7:              i ← i − 1
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1                                     ▷ O(1)
5:          while i ≥ 0 AND L[i] > clé do                 ▷ Pire cas : j − 1 fois
6:              L[i + 1] ← L[i]                           ▷ O(1)
7:              i ← i − 1                                 ▷ O(1)
8:          end while
9:          L[i + 1] ← clé
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1                                     ▷ O(1)
5:          while i ≥ 0 AND L[i] > clé do                 ▷ Pire cas : j − 1 fois
6:              L[i + 1] ← L[i]                           ▷ O(1)
7:              i ← i − 1                                 ▷ O(1)
8:          end while
9:          L[i + 1] ← clé                                ▷ O(1)
10:     end for
11:     return L
12: end procedure
```

- Tri par insertion : Complexité
  - ▷ Complexité de l'algorithme dans le pire cas

Tri par insertion d'un tableau L de longueur n

```
1:  procedure triInsertion(L, n)
2:      for j from 1 to n − 1 do                          ▷ n − 1 fois
3:          clé ← L[j]                                    ▷ O(1)
4:          i ← j − 1                                     ▷ O(1)
5:          while i ≥ 0 AND L[i] > clé do                 ▷ Pire cas : j − 1 fois
6:              L[i + 1] ← L[i]                           ▷ O(1)
7:              i ← i − 1                                 ▷ O(1)
8:          end while
9:          L[i + 1] ← clé                                ▷ O(1)
10:     end for
11:     return L
12: end procedure
```

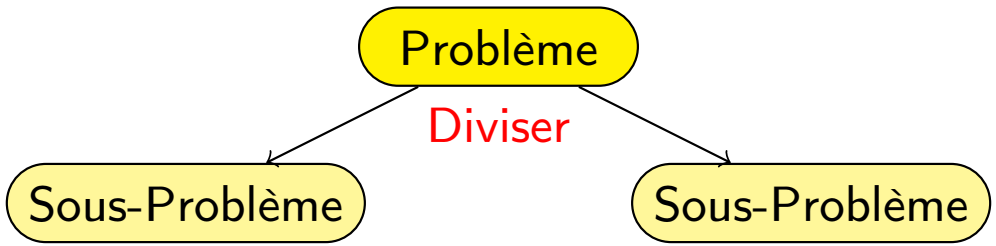
  - ▷ Coût final (pire cas)

$$T(n) = \sum_{j=1}^{n-1} \left[\mathcal{O}(1) + \sum_{i=1}^{j-1} \mathcal{O}(1) \right] = n + n \cdot \mathcal{O}(n) = \mathcal{O}\left(n^2\right)$$

- Diviser pour régner
  - ▷ Concept général

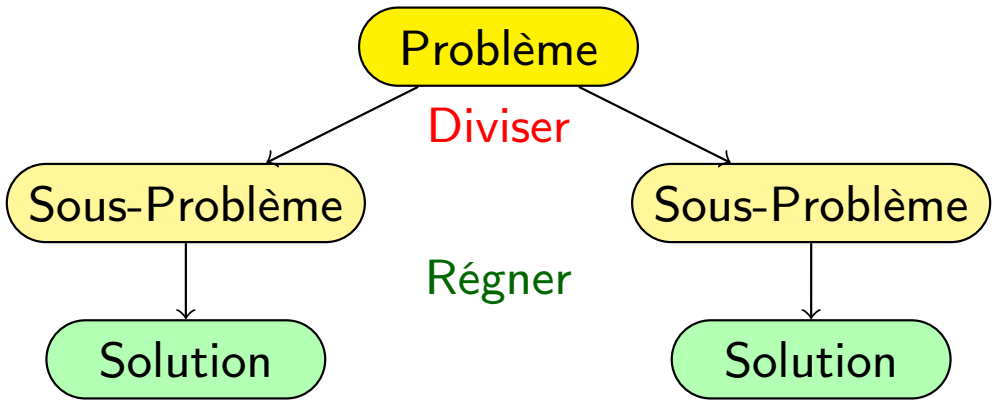
- Diviser pour régner

  ▷ Concept général



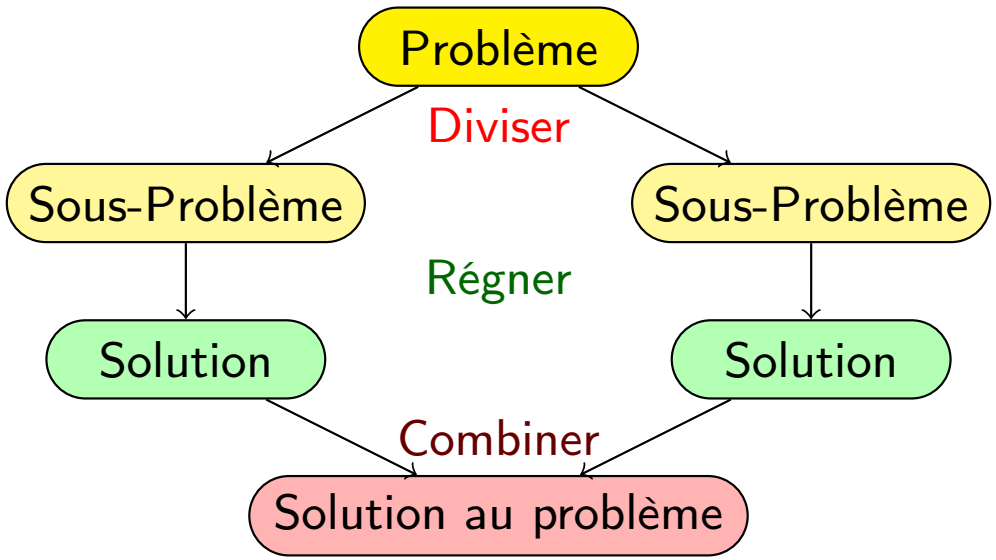
- Diviser pour régner

  ▷ Concept général



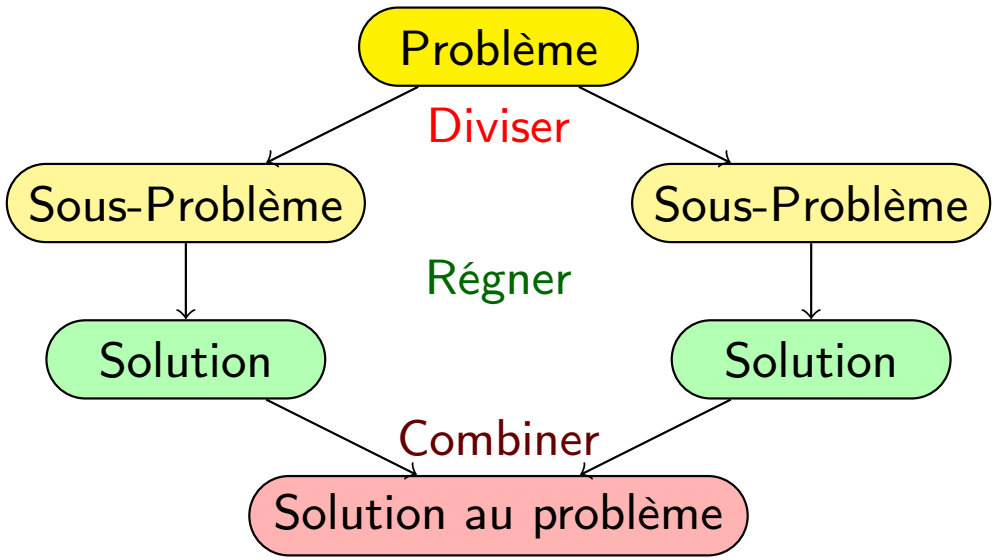
- Diviser pour régner

  ▷ Concept général



- Diviser pour régner

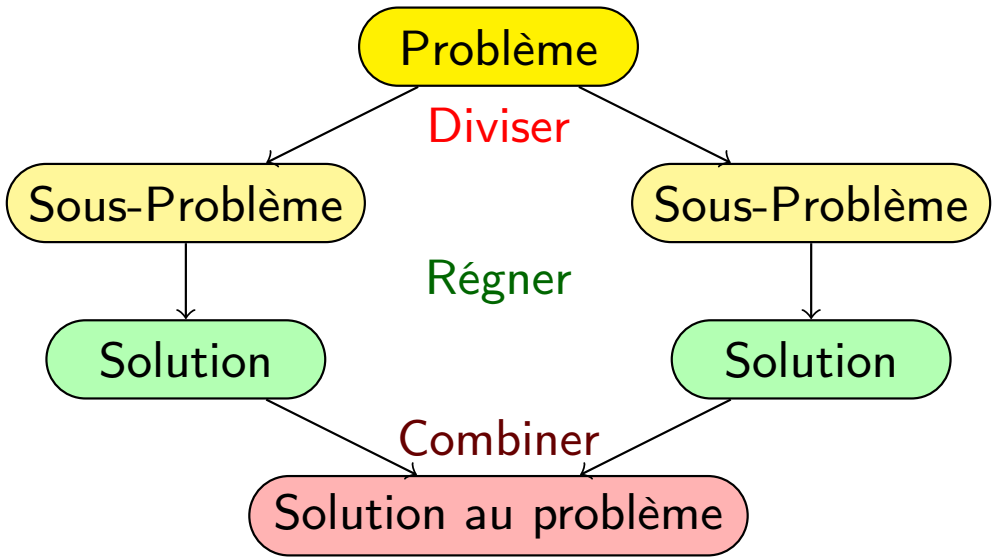
  ▷ Concept général



  - DIVISER : Les sous-problèmes sont des plus petites instances du même problème général

- Diviser pour régner

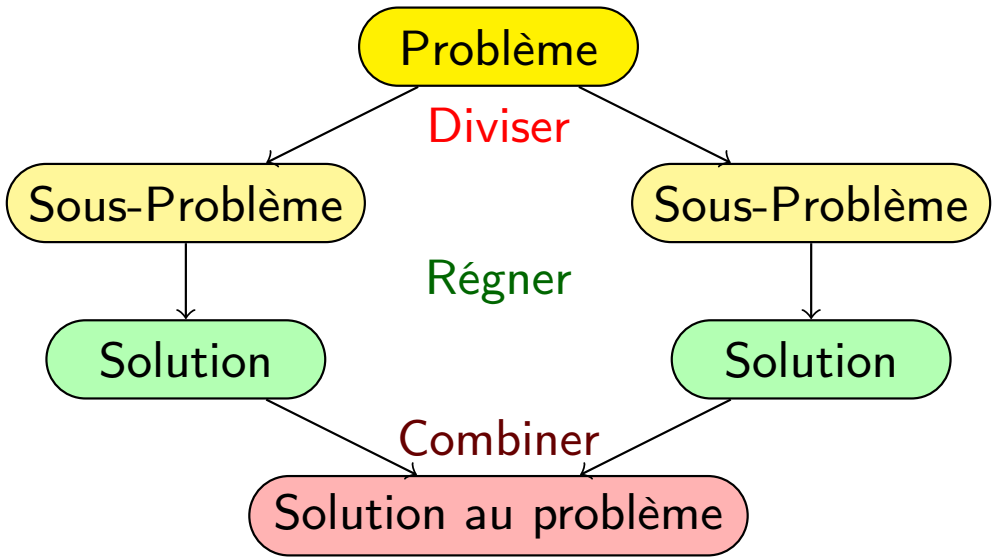
  ▷ Concept général



- DIVISER : Les sous-problèmes sont des plus petites instances du même problème général
- REGNER : Si on réduit suffisamment les sous-problèmes, on arrive à les résoudre directement

- Diviser pour régner

  ▷ Concept général



- DIVISER : Les sous-problèmes sont des plus petites instances du même problème général
- REGNER : Si on réduit suffisamment les sous-problèmes, on arrive à les résoudre directement
- COMBINER : On combine ensuite les solutions des sous-problèmes pour résoudre le problème général

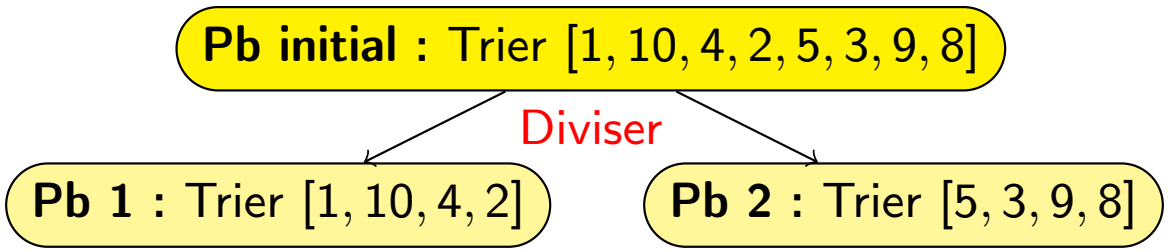
- Diviser pour régner : Exemple

  - ▷ Trier une liste

Pb initial : Trier $[1, 10, 4, 2, 5, 3, 9, 8]$

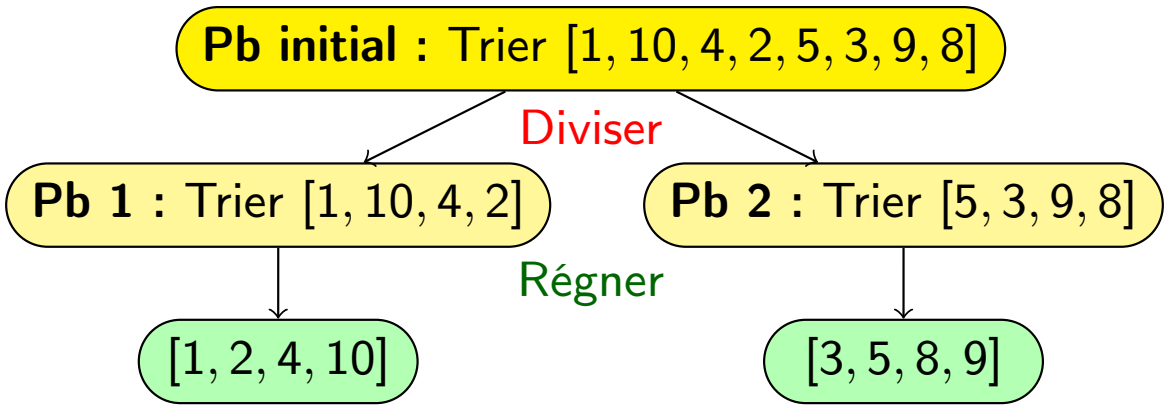
- Diviser pour régner : Exemple

  ▷ Trier une liste



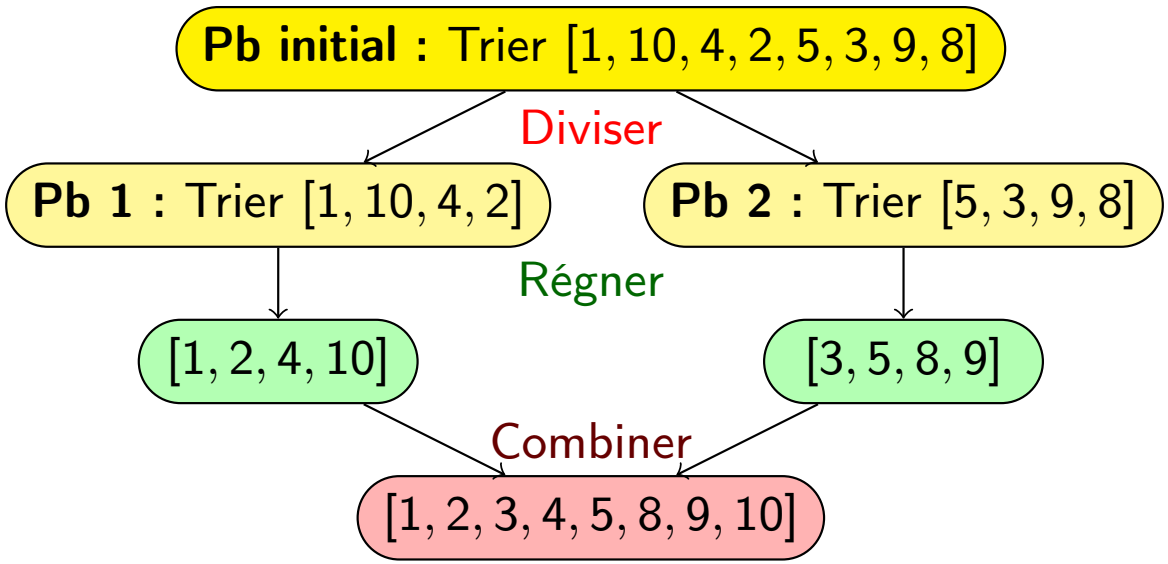
- Diviser pour régner : Exemple

  ▷ Trier une liste



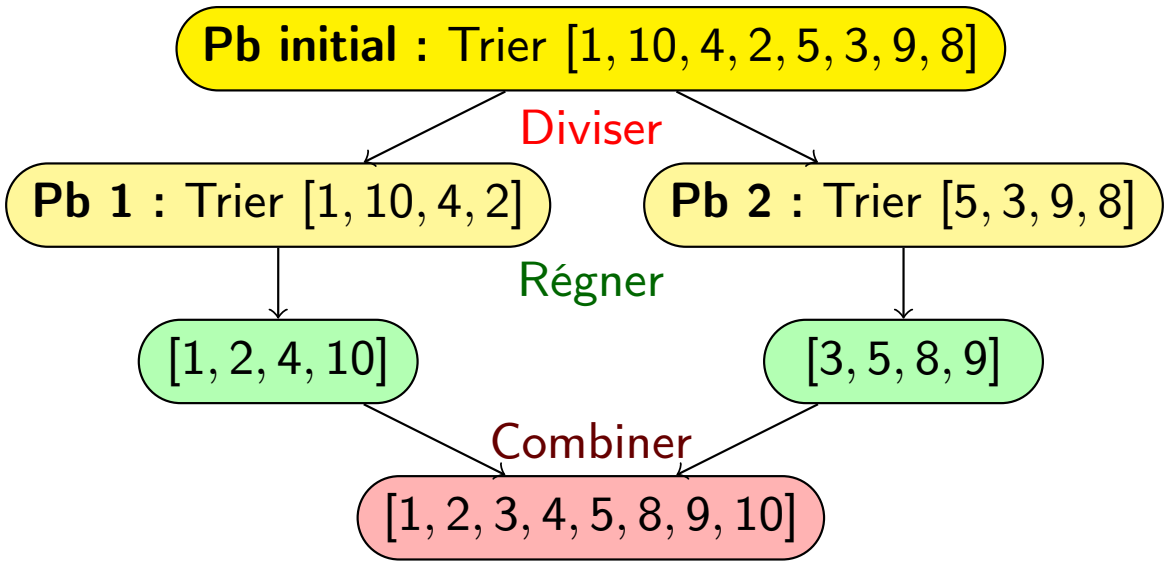
- Diviser pour régner : Exemple

  ▷ Trier une liste



• Diviser pour régner : Exemple

▷ Trier une liste

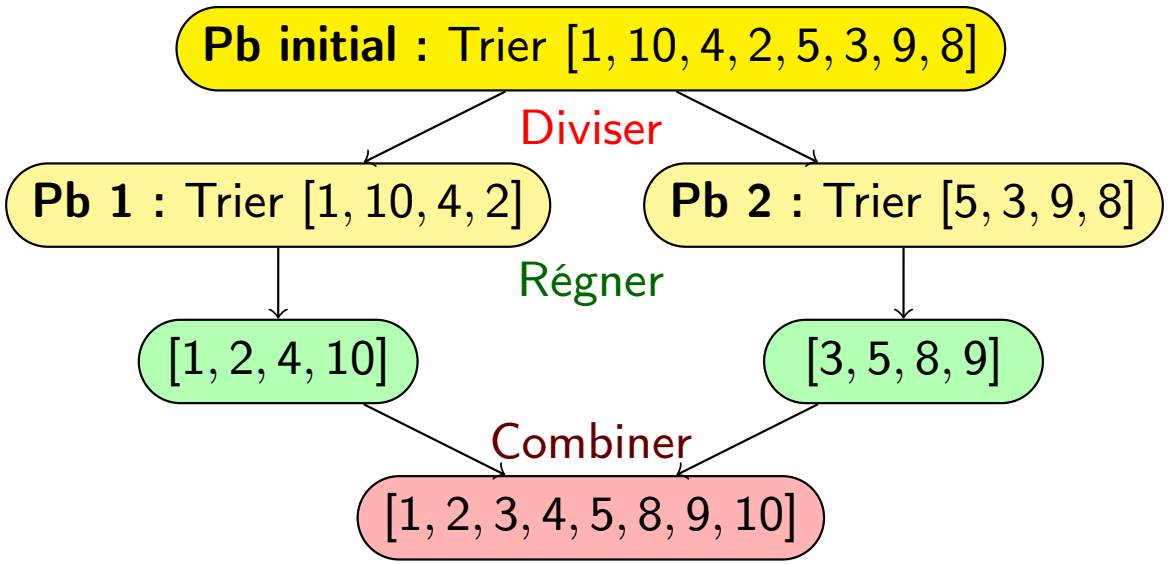


▷ Application : le tri fusion

- Un algorithme très efficace (optimal dans le cas moyen)

• Diviser pour régner : Exemple

▷ Trier une liste

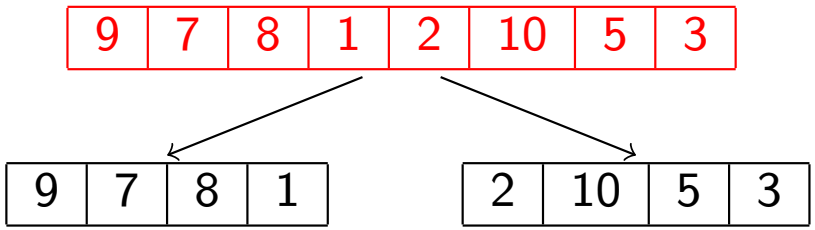


▷ Application : le tri fusion

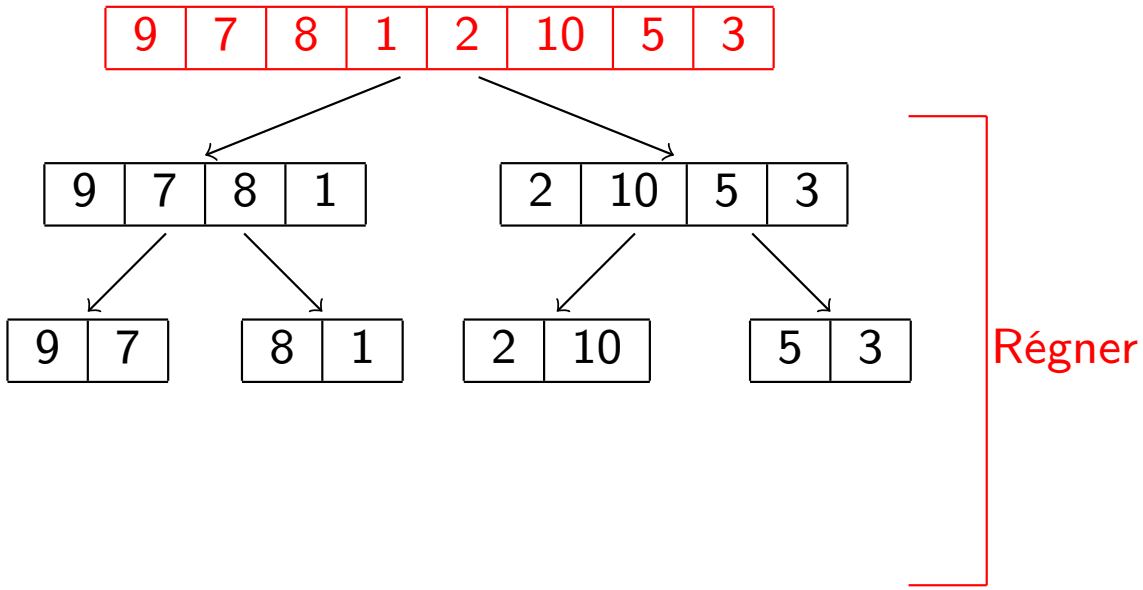
- Un algorithme très efficace (optimal dans le cas moyen)
- Des appels récursifs

• Tri fusion : Exemple

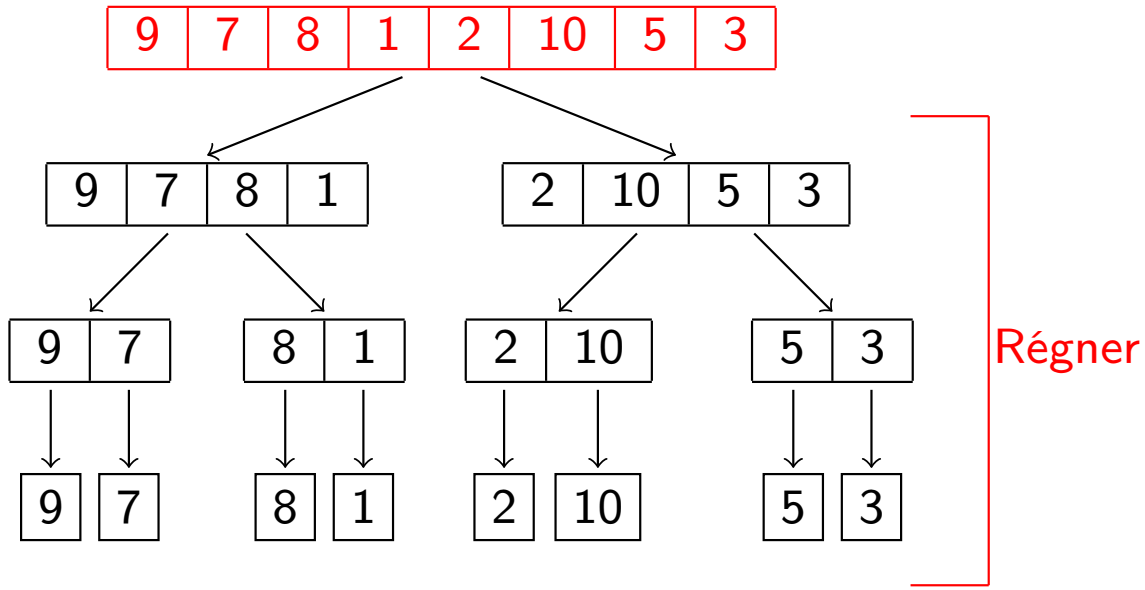
• Tri fusion : Exemple



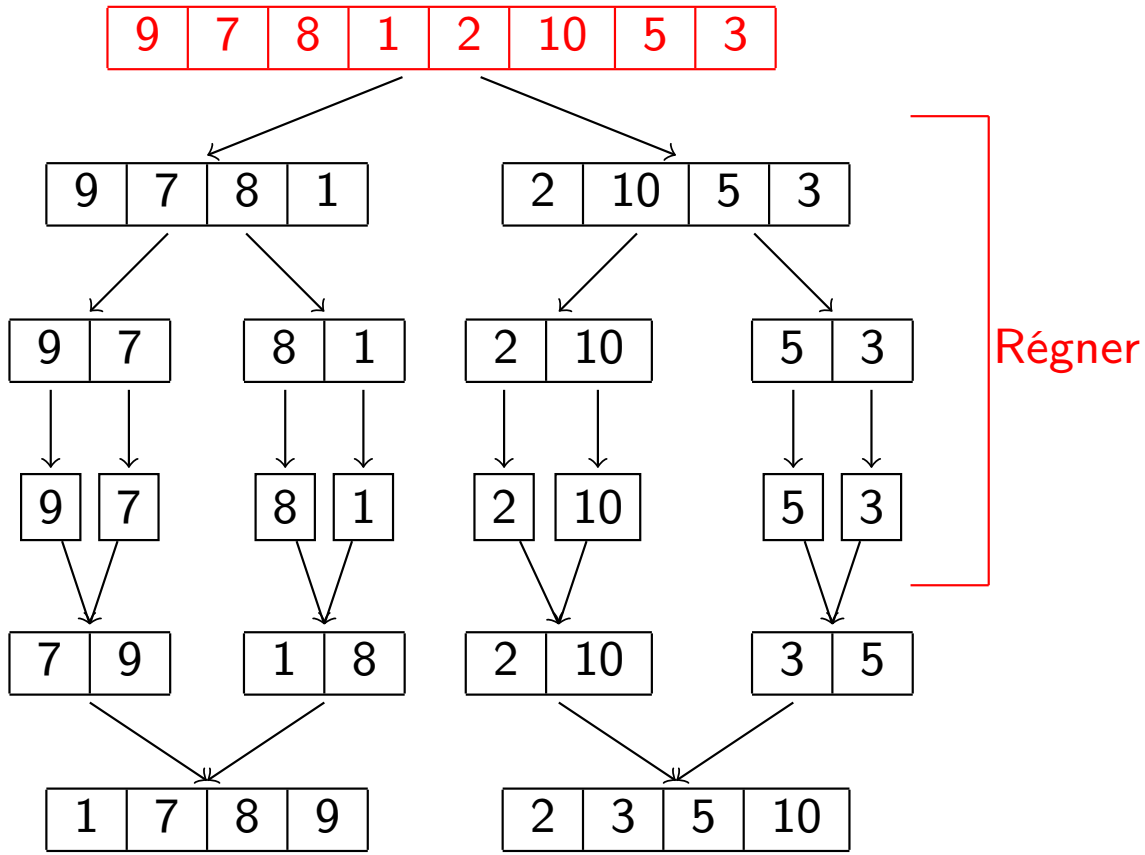
• Tri fusion : Exemple



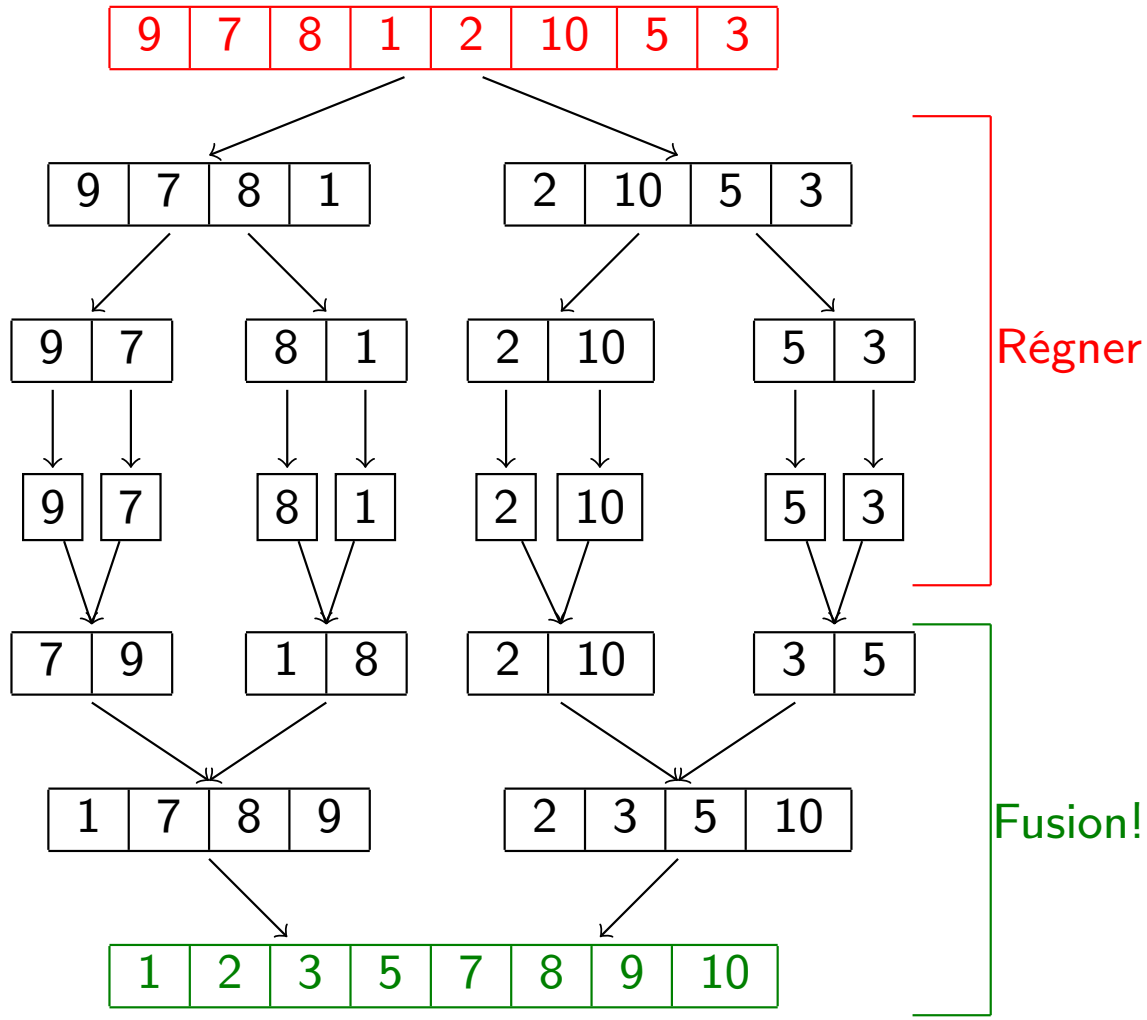
• Tri fusion : Exemple



• Tri fusion : Exemple



• Tri fusion : Exemple



Pause

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

```
1: procedure triFusion(L[0, ··· , n − 1], l, u)
2:     if u − l ≤ 1 then
3:         return L[l, ··· , u]
4:     else
5:         return fusion(triFusion(L, l, (u−l−1)/2), triFusion(L, (u−l−1)/2 + 1, u ))
6:     end if
7: end procedure
```

▷ Explications

.

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

1: **procedure** triFusion($L[0, \cdots, n-1], l, u$)
2: **if** $u - l \leq 1$ **then**
3: **return** $L[l, \cdots, u]$
4: **else**
5: **return fusion**(**triFusion**($L, l, \frac{u-l-1}{2}$), **triFusion**($L, \frac{u-l-1}{2} + 1, u$))
6: **end if**
7: **end procedure**

▷ Explications

- Si le tableau n'a qu'un élément : déjà trié.

.

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

```
1: procedure triFusion(L[0, ··· , n − 1], l, u)
2:     if u − l ≤ 1 then
3:         return L[l, ··· , u]
4:     else
5:         return fusion(triFusion(L, l, (u−l−1)/2), triFusion( L, (u−l−1)/2 + 1, u))
6:     end if
7: end procedure
```

▷ Explications

- Si le tableau n'a qu'un élément : déjà trié.
- Sinon, séparer le tableau en 2 parties, à peu près égales.

.

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

```
1: procedure triFusion(L[0, ··· , n − 1], l, u)
2:     if u − l ≤ 1 then
3:         return L[l, ··· , u]
4:     else
5:         return fusion(triFusion(L, l, (u−l−1)/2), triFusion( L, (u−l−1)/2 + 1, u))
6:     end if
7: end procedure
```

▷ Explications

- Si le tableau n'a qu'un élément : déjà trié.
- Sinon, séparer le tableau en 2 parties, à peu près égales.
- Trier récursivement les 2 parties avec l'algorithme du tri fusion.

.

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

```
1: procedure triFusion(L[0, ··· , n − 1], l, u)
2:     if u − l ≤ 1 then
3:         return L[l, ··· , u]
4:     else
5:         return fusion(triFusion(L, l, (u−l−1)/2), triFusion( L, (u−l−1)/2 + 1, u))
6:     end if
7: end procedure
```

▷ Explications

- Si le tableau n'a qu'un élément : déjà trié.
- Sinon, séparer le tableau en 2 parties, à peu près égales.
- Trier récursivement les 2 parties avec l'algorithme du tri fusion.
- Puis, fusionner les deux tableaux triés en un seul tableau trié (slide suivante).

• Tri fusion : Formalisation

▷ Tri fusion

Tri fusion sur un tableau L de taille n

```
1: procedure triFusion(L[0, ··· , n − 1], l, u)
2:     if u − l ≤ 1 then
3:         return L[l, ··· , u]
4:     else
5:         return fusion(triFusion(L, l, (u−l−1)/2), triFusion( L, (u−l−1)/2 + 1, u))
6:     end if
7: end procedure
```

▷ Explications

- Si le tableau n'a qu'un élément : déjà trié.
- Sinon, séparer le tableau en 2 parties, à peu près égales.
- Trier récursivement les 2 parties avec l'algorithme du tri fusion.
- Puis, fusionner les deux tableaux triés en un seul tableau trié (slide suivante). Coût total : $\mathcal{O}(n \log n)$

- Tri fusion : la fusion

  ▷ Fusion de deux tableau triés

Fusion de deux tableau triés A et B

```
1:  procedure fusion(A[0, ··· , a − 1], B[0, ··· , b − 1])
2:      if A est le tableau vide then
3:          return B
4:      else if B est le tableau vide then
5:          return A
6:      else if A[0] ≤ B[0] then
7:          return A[0] : : fusion(A[1, ··· , a − 1], B)
8:      else
9:          return B[1] : : fusion(A, B[1, ··· , b − 1])
10:     end if
11: end procedure
```

• Tri fusion : la fusion

▷ Fusion de deux tableau triés

Fusion de deux tableau triés A et B

```
1:  procedure fusion(A[0, ··· , a − 1], B[0, ··· , b − 1])
2:      if A est le tableau vide then
3:          return B
4:      else if B est le tableau vide then
5:          return A
6:      else if A[0] ≤ B[0] then
7:          return A[0] : : fusion(A[1, ··· , a − 1], B)
8:      else
9:          return B[1] : : fusion(A, B[1, ··· , b − 1])
10:     end if
11: end procedure
```

Coût de la fusion

- Taille de l'entrée :
- Coût total :

• Tri fusion : la fusion

▷ Fusion de deux tableau triés

Fusion de deux tableau triés A et B

```
1:  procedure fusion(A[0, ··· , a − 1], B[0, ··· , b − 1])
2:      if A est le tableau vide then
3:          return B
4:      else if B est le tableau vide then
5:          return A
6:      else if A[0] ≤ B[0] then
7:          return A[0] : : fusion(A[1, ··· , a − 1], B)
8:      else
9:          return B[1] : : fusion(A, B[1, ··· , b − 1])
10:     end if
11: end procedure
```

Coût de la fusion

- Taille de l'entrée : taille du tableau A, a et du tableau B, b.
- Coût total :

• Tri fusion : la fusion

▷ Fusion de deux tableau triés

Fusion de deux tableau triés A et B

```
1:  procedure fusion(A[0, ··· , a − 1], B[0, ··· , b − 1])
2:      if A est le tableau vide then
3:          return B
4:      else if B est le tableau vide then
5:          return A
6:      else if A[0] ≤ B[0] then
7:          return A[0] : : fusion(A[1, ··· , a − 1], B)
8:      else
9:          return B[1] : : fusion(A, B[1, ··· , b − 1])
10:     end if
11: end procedure
```

Coût de la fusion

- Taille de l'entrée : taille du tableau A, a et du tableau B, b.
- Coût total : $\mathcal{O}(a + b)$

- **Théorème sur la complexité des algorithmes de tri dans le pire cas**
  - ▷ Tris par comparaison
    - On assimile la complexité dans le temps au nombre de comparaisons nécessaires.
    - On ne peut faire mieux (moyen cas) que $\mathcal{O}(n \log n)$ où n est la taille du tableau.
    - Le tri fusion est un tri asymptotiquement optimal.

- **Théorème sur la complexité des algorithmes de tri dans le pire cas**

  ▷ Tris par comparaison

  - On assimile la complexité dans le temps au nombre de comparaisons nécessaires.
  - On ne peut faire mieux (moyen cas) que $\mathcal{O}(n \log n)$ où n est la taille du tableau.
  - Le tri fusion est un tri asymptotiquement optimal.

  ▷ Tableau de comparaison (temps asymptotique )

nom	cas optimal	cas moyen	pire cas	complexité spatiale
tri rapide	$n \log n$	$n \log n$	n^2	$\log n$ (moyen cas) n (pire cas)
tri fusion	$n \log n$	$n \log n$	$n \log n$	n
tri par tas	$n \log n$	$n \log n$	$n \log n$	1
tri par insertion	n	n^2	n^2	1
tri à bulles	n	n^2	n^2	1
tri stupide	n	$n!$	$n!$	1

- **Quels algorithme choisir ?**

  ▷ Tris lents

  - Tri par insertion : Le plus rapide pour les entrées de petites tailles, ou déjà presque triées car très simple.
  - Tri à bulles : Peu efficace. Intérêt pédagogique.
  - Tri par tas : lent en comparaison des autres algorithmes car la constante devant $n \log n$ est très grande, mais efficace asymptotiquement... Il est utilisé quand **le temps** prime moins que **l'espace**. (tri "en place").
  - Tri stupide : Tester des permutations ... à ne pas faire.

  ▷ Tris rapides

  - Tri fusion : Rapide, demande de la place en espace mémoire.
  - Tri rapide : Complexité quasi-indépendante des données avec quelques optimisations ...

  ▷ Tri fusion vs tri insertion

  On va voir la différence entre tri fusion et tri insertion pour une distribution aléatoire de nombres.

tri fusion vs tri insertion

Pour finir : Une référence

▷ **Algorithmique** de Cormen, Leiserson, Rivest, Stein.

- Algorithmique générale
- Complexité, algorithmiques probabilistes
- Tris et rangs
- Structures de données
- Algorithmique avancée et analyse
- Algorithmique des graphes
- etc...

▷ **Un site** dédié aux tris

Adresse : http ://lwh.free.fr/pages/algo/tri/tri_fusion.html

- Des exemples
- Les pseudo-codes et codes dans divers langages (C, Python, etc)
- Des animations