EPFL

Information, Calcul et Communication

CS-119(g) ICC – Théorie
Semaine 14

Rafael Pires
rafael.pires@epfl.ch

EPFL

Lausanne, 12.12.2025

ICC-T 9

- Les ingrédients de base des algorithmes
  - Données (variables)
  - Instructions (affectations, structures de contrôle)
- Problèmes :
  - Calcul du modulo 3 d'un grand nombre
  - Recherche du minimum dans une liste
  - Problème du voyageur du commerce
  - Comparaison tous contre tous
  - L'algorithme d'Euclide (pgdc)

EPFL

ICC-T 9 : Comparaisons tous contre tous

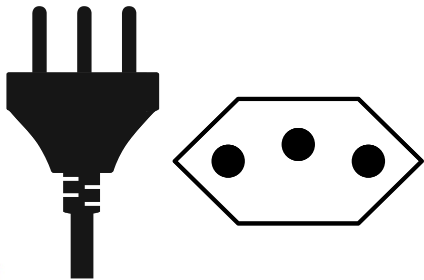
i \ k	1	2	3	4	5
1	1,1	1,2	1,3	1,4	1,5
2	2,1	2,2	2,3	2,4	2,5
3	3,1	3,2	3,3	3,4	3,5
4	4,1	4,2	4,3	4,4	4,5
5	5,1	5,2	5,3	5,4	5,5

Tous différents

entrée : liste ***L*** de ***n*** objets
sortie : valeur binaire oui/non

s ← ***oui***
Pour ***i*** *allant de 1 à* ***n****-1 :*
    Pour ***k*** *allant de* ***i****+1 à* ***n*** **:**
        Si ***L***(***i***) = ***L***(***k***), alors ***s*** ← ***non***
Sortir : ***s***

ICC-T 10



- Les sous-algorithmes
  - Problème : Tri par insertion
- La complexité temporelle
  - nombre d'opérations élémentaires dans le pire des cas
- La notation Grand Theta Θ
  - Problème : Deux font la paire

ICC-T 10 : Question :

Analysons trois algorithmes différents :

Algorithme A

$s \leftarrow 0$
Pour i allant de 1 à n :
  Pour j allant de 1 à n :
    $s \leftarrow s + 1$
Sortir : s

Algorithme B

$s \leftarrow 0$
Pour i allant de 1 à n :
  Pour j allant de i à n :
    $s \leftarrow s + 1$
Sortir : s

Algorithme C

$s \leftarrow 0$
$i \leftarrow 1$
Tant que $i \leq n$:
    $s \leftarrow s + 1$
    $i \leftarrow i \times 2$
Sortir : s

Quelle est la complexité temporelle Θ de chaque algorithme ?

Si n = 1000, combien d'opérations (environ) effectue chaque algorithme ?

EPFL

ICC-T 10 : Tri par insertion (algorithme entier)

Tri par insertion

entrée : liste *L* de taille *n*
sortie : liste *L* triée dans l'ordre croissant

```
Pour i allant de 2 à n :
        Si L(i) < L(i-1), alors
                L ← insérer(L, i)
Sortir : L
```

insérer

entrée : liste *L*, indice *i*
sortie : liste *L* avec l'élement L(i) bien placé

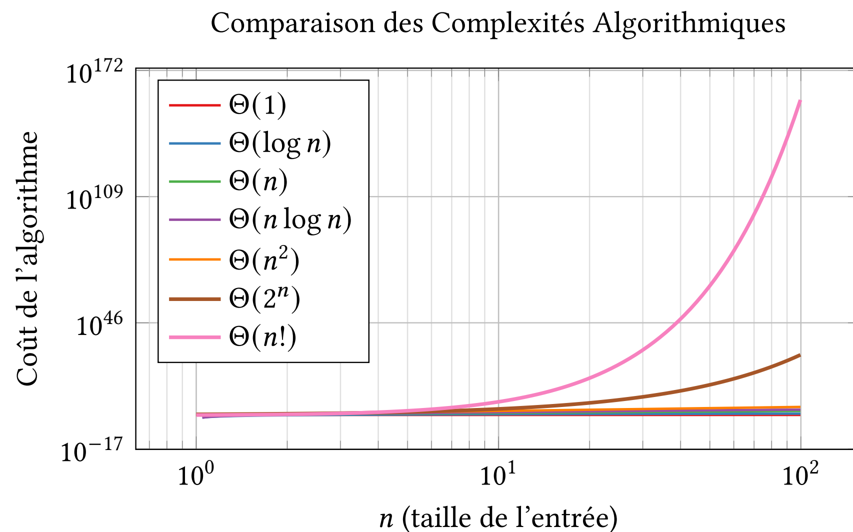
```
Tant que i > 1 et L(i) < L(i-1) :
        L ← permuter(L, i, i-1)
        i ← i-1
Sortir : L
```

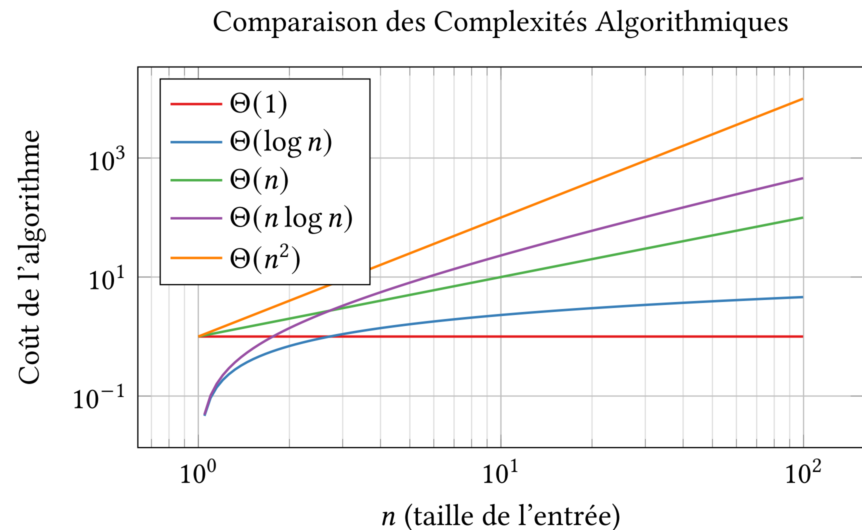
permuter

entrée : liste *L*, indices *j* et *k*
sortie : liste *L* avec les éléments L(j) et L(k) permutés

```
temp ← L(j)
L(j) ← L(k)
L(k) ← temp
Sortir : L
```

ICC-T 10 : Notation $\Theta(\cdot)$: Ordres de grandeur





Impraticables : $\Theta(2^n)$, $\Theta(n!)$

Plus lents, mais souvent acceptés : $\Theta(n^2)$ … $\Theta(n^k)$, $\Theta(n.\log(n))$

Rapides: $\Theta(1)$, $\Theta(\log n)$, $\Theta(n)$

EPFL

ICC-T 10 : Question :

Quelle est la complexité temporelle Θ de chaque algorithme ?

Si n = 10 000, combien de fois (environ) la comparaison de somme est-elle effectuée dans le pire des cas ?

Somme cible – Version 1

entrée : liste ***L*** de ***n*** nombres entiers, cible ***k***
sortie : oui si deux éléments somment à ***k***

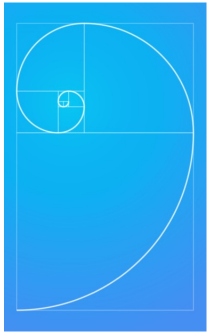
```
Pour i allant de 1 à n-1 :
    Pour j allant de i+1 à n :
        Si L(i) + L(j) = k, alors :
            Sortir oui
Sortir : non
```

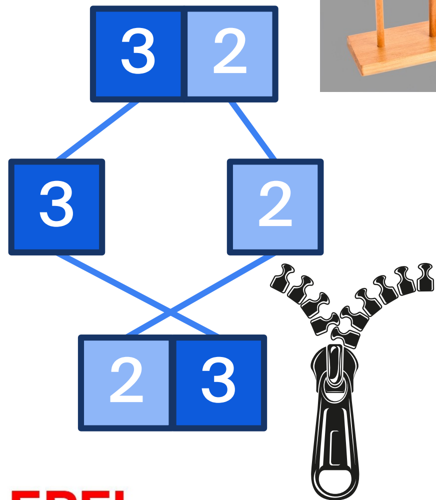
Somme cible – Version 2

entrée : liste ordonnée ***L*** de ***n*** nombres entiers, cible ***k***
sortie : oui si deux éléments somment à ***k***

$i \leftarrow 1$
$j \leftarrow n$
Tant que $i < j$:
  Si $L(i) + L(j) = k$**, alors** : **Sortir** : *oui*
  Si $L(i) + L(j) < k$**, alors** : $i \leftarrow i + 1$
  Si $L(i) + L(j) > k$**, alors** : $j \leftarrow j - 1$
Sortir : *non*

ICC-T 11





- La récursivité
  - Factorielle - 1 appel récursif
  - Suite de Fibonacci - 2 appels récursifs
  - Tours de Hanoï - 3 appels récursifs
- Le logarithme dans la complexité
  - Recherche dichotomique - $\Theta(\log_2 n)$
  - Tri par fusion - $\Theta(n \log_2 n)$

ICC-T 11 : Problème : Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, …

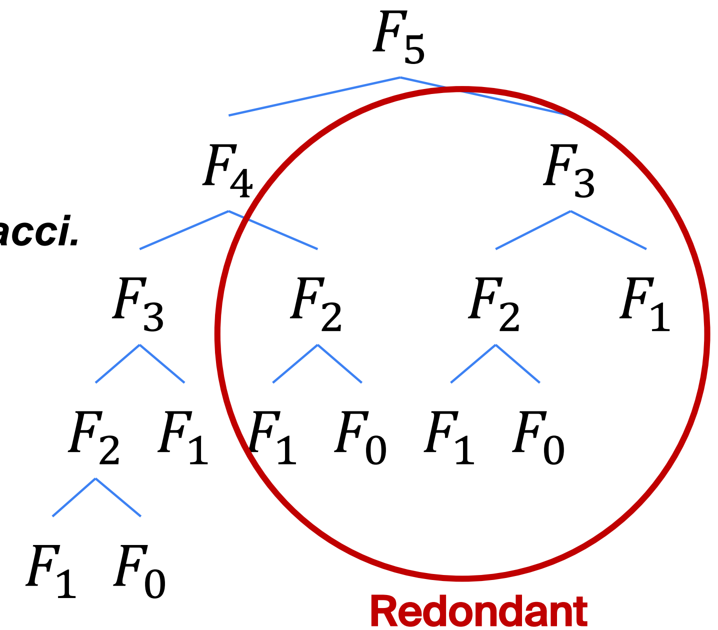
Le nième terme de la suite de Fibonacci.

$$F(n) = F(n-1) + F(n-2)$$

$$F(0) = F(1) = 1$$

Fibonacci
entrée : entier naturel **n** **sortie** : nième nombre dans la suite de Fibonacci

Si n <= 1
  Sortir : 1
Sinon
  Sortir : Fibonacci(n - 1) + Fibonacci(n - 2)



Complexité :

$$\Theta(\phi^n)$$

ICC-T 11 : Recherche dichotomique

- **Problème**
  - Identifier si un élément fait partie d'une liste ordonnée.

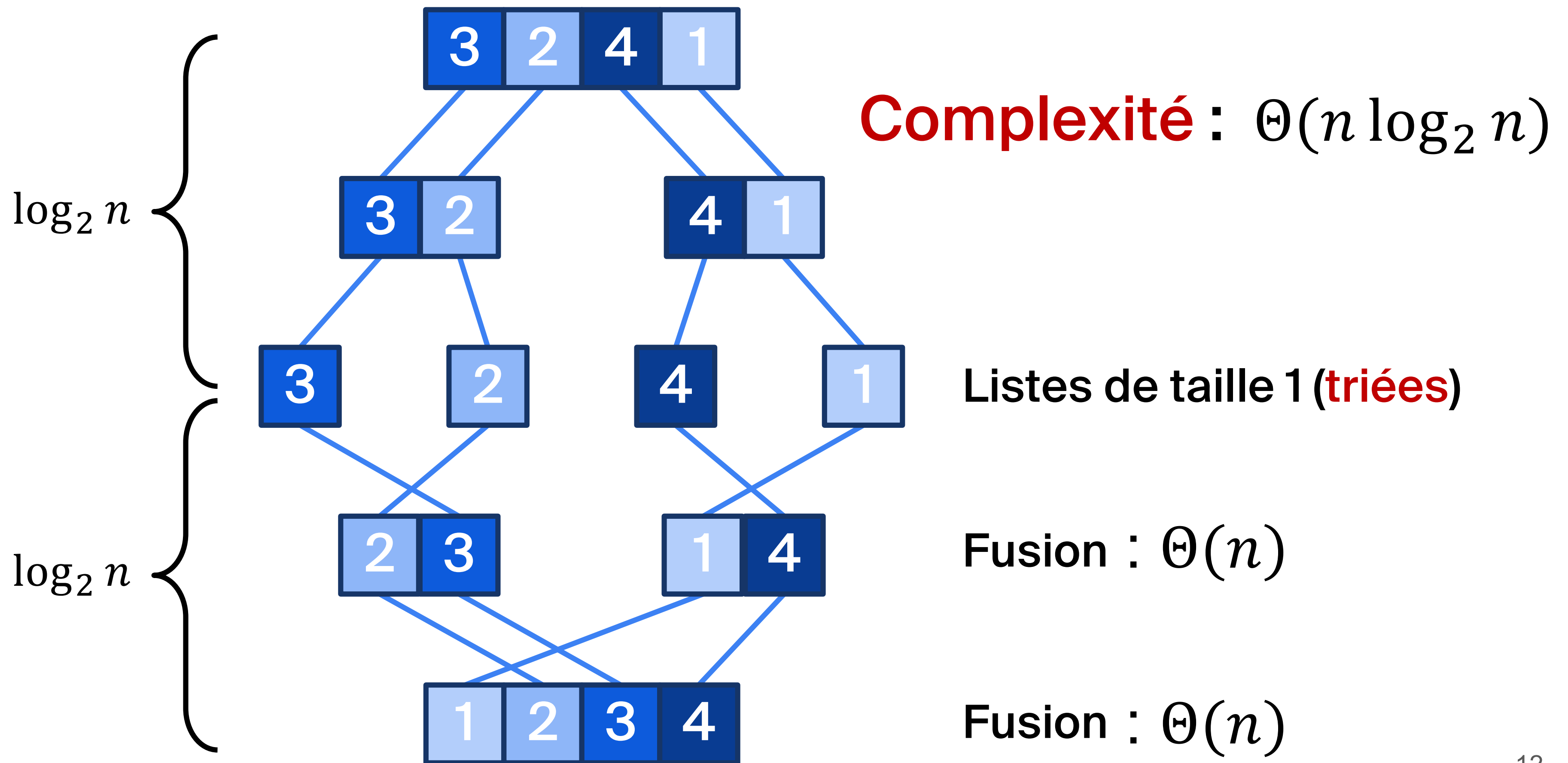
Dichotomie

entrée : Liste ordonée L de nombres entiers de taille n
objet x qu'on veut chercher
sortie : oui si x est dans L, **non** sinon

Si n = 1
  Sortir : x = L(1)

milieu ← $\left\lfloor \frac{n}{2} \right\rfloor$
Si x ≤ L(milieu)
  Sortir : Dichotomie(L(1 : milieu), milieu, x)
Sinon
  Sortir : Dichotomie(L(1+milieu : n), n - milieu, x)

ICC-T 11 : Complexité temporelle : Tri par fusion



Algorithme entier : Tri par fusion

Tri par fusion

entrée : Liste L non triée de nombres entiers, de taille n
sortie : Liste L' triée

```
Si n = 1
    Sortir : L

milieu ← ⌊n/2⌋
L₁ ← Tri par fusion(L(1 : milieu), milieu)
L₂ ← Tri par fusion(L(1+milieu : n), n - milieu)
L' ← fusion(L₁,L₂)
Sortir : L'
```

fusion

entrée : Listes ordonnées L_1, L_2 de taille m_1 et m_2 resp.
sortie : Liste L de taille $m_1 + m_2$ également ordonnée

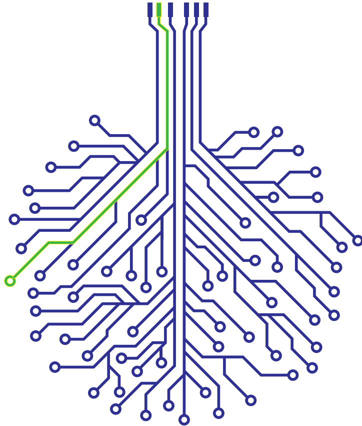
```
j₁ ← 1
j₂ ← 1
j ← 1

Tant que j₁ ≤ m₁ et j₂ ≤ m₂ :
    Si L₁(j₁) ≤ L₂(j₂) :
        L(j) ← L₁(j₁)
        j₁ ← j₁ + 1
    Sinon :
        L(j) ← L₂(j₂)
        j₂ ← j₂ + 1
    j ← j + 1

Si j₁ = m₁ + 1 :
    Tant que j₂ ≤ m₂ :
        L(j) ← L₂(j₂)
        j₂ ← j₂ + 1
        j ← j + 1
Sinon :
    Tant que j₁ ≤ m₁ :
        L(j) ← L₁(j₁)
        j₁ ← j₁ + 1
        j ← j + 1

Sortir : L
```

ICC-T 12



- Algorithmes gloutons
- Programmation dynamique
  - Rendu de pièces de monnaie
- Memoïsation
  - Suite de Fibonacci
- Théorie de la calculabilité : décidabilité
  - Le problème de l’arrêt

ICC-T 12 : Algorithmes gloutons

Pas toujours la solution optimale du problème

Rendu glouton

entrée : z = 0.60
p = { 0.10, 0.30, 0.40 }
sortie : ?

Si $z = 0$
  Sortir : $\emptyset$
Si $p = \emptyset$ **ou** $z < p_1$
  Sortir : « Rendu exact impossible »
Si $z < p_n$
  Sortir : Rendu glouton(z, $p \setminus \{ p_n \}$)
Sortir : $\{ p_n \} \cup$ Rendu glouton($z - p_n$, p)

Considérons maintenant z = 0.60, avec l'ensemble des pièces:

$$P = \{ 0.10, 0.30, 0.40 \}$$

Résultat :

$$\{ 0.40, 0.10, 0.10 \}$$

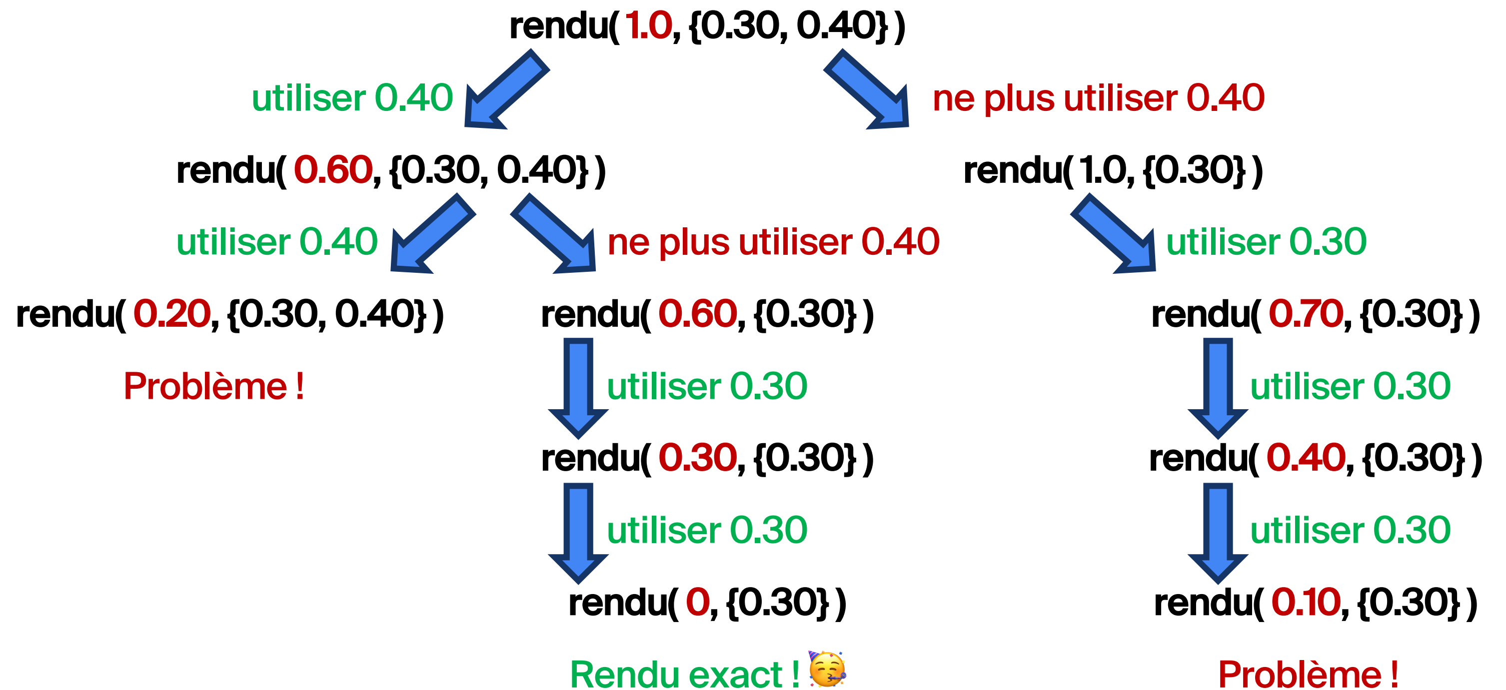
Ce qui est certes un rendu exact, mais ne minimise pas le nombre de pièces rendues, car il aurait mieux valu rendre :

$$\{ 0.30, 0.30 \}$$

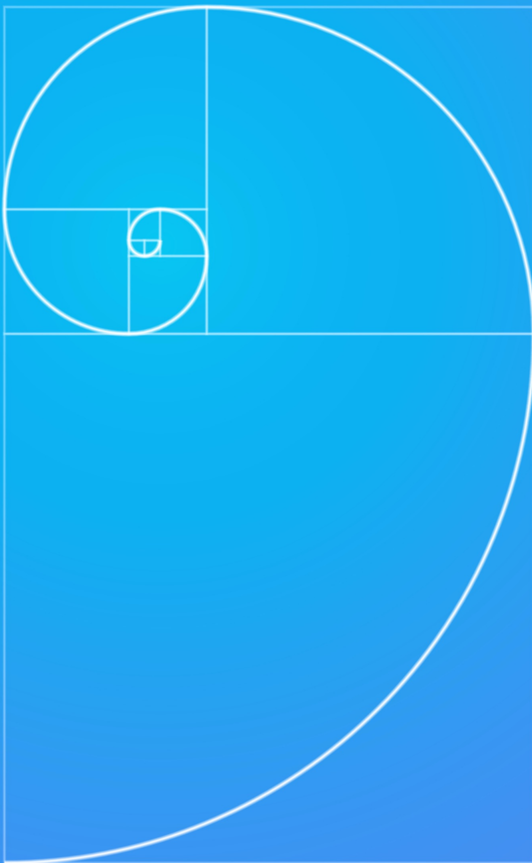
ICC-T 12 : Programmation dynamique : Rendu de pièces de monnaie

- Exemple :

P = { 0.30, 0.40 }, z = 1.0



ICC-T 12 : Fibonacci : memoïsation



$\Theta(\phi^n)$

Fibonacci naïve

entrée : entier naturel **n**
sortie : nième nombre dans la suite de Fibonacci

Si n ≤ 1
    Sortir : 1
Sinon
    Sortir : Fibonacci naïve(n - 1) + Fibonacci naïve(n - 2)

$\Theta(n)$

Fibonacci

entrée : entier naturel **n**, liste des résultats **memo**
sortie : nième nombre dans la suite de Fibonacci

Si n ≤ 1
    Sortir : 1
Si n est dans memo
    Sortir : memo[n]
Sinon
    memo[n] ← **Fibonacci(n – 1, memo) + Fibonacci(n – 2, memo)**
    Sortir : memo[n]

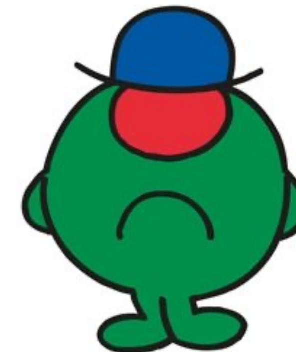
ICC-T 12 : Le problème de l'arrêt

- Supposons, par hypothèse, qu'un tel algorithme A existe, c'est-à-dire :
  - A(P, X) sort oui si P(X) s'arrête
  - A(P, X) sort non si P(X) continue indéfiniment

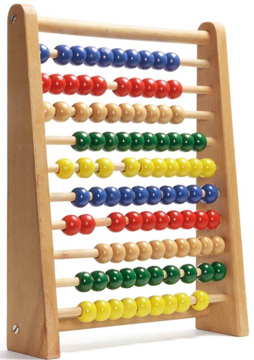
- A partir de cet algorithme A, on construit un autre algorithme qu'on appelle « M. Non » :

M. NON

M. Non
entrée : algorithme P **sortie** : aucune
Si A(P, P) = oui, alors : **Effectue une boucle infinie** **Sinon** **S'arrêter**



ICC-T 13

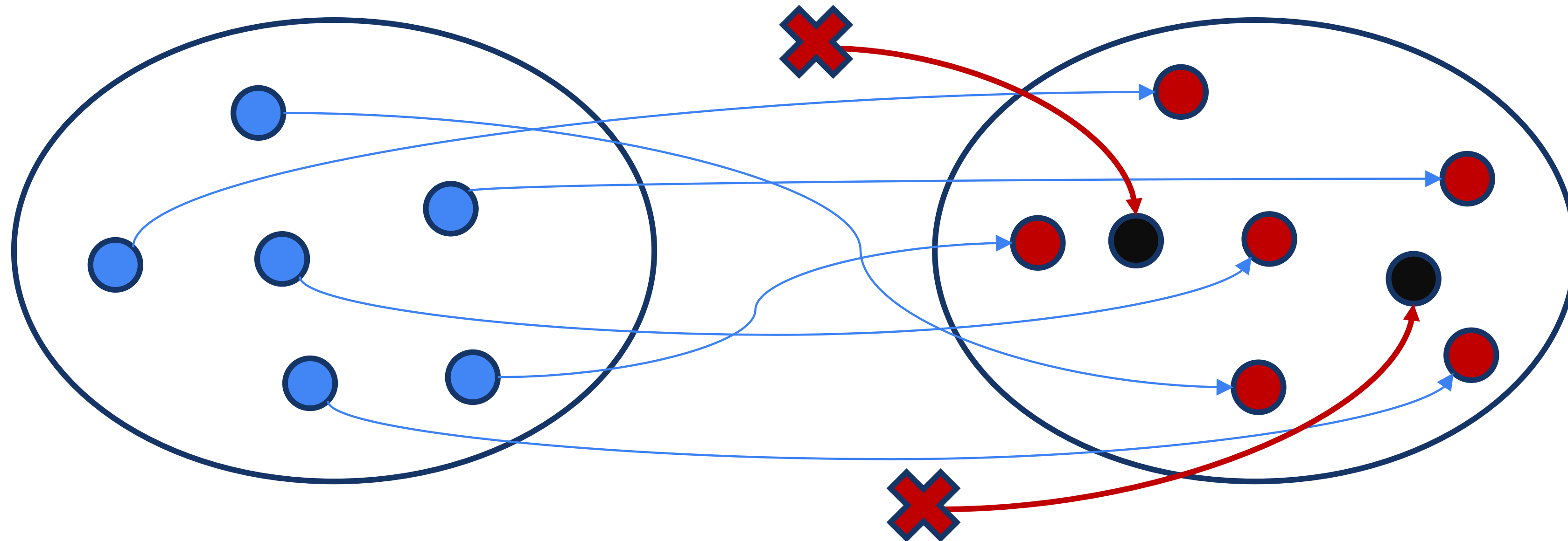


- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d'optimisation discrète

EPFL

ICC-T 13 : Limites des algorithmes

- A = Ensemble de tous les algorithmes définissables.
- B = Ensemble de toutes les sorties possibles.



- Existe-t-il une fonction définissable (via un algorithme) pour calculer chaque élément de B ?

Non

EPFL

ICC-T 13 : Classe P : Résoudre en temps polynomial

- La **classe P** est l'ensemble des problèmes qui peuvent être résolus en temps polynomial
  - il existe un algorithme de résolution dont la complexité temporelle est

$$O(n^p)$$ pour des données d'entrée de taille n (avec $p \geq 1$ un nombre fixé)

ICC-T 13 : Classe NP : Vérifier en temps polynomial

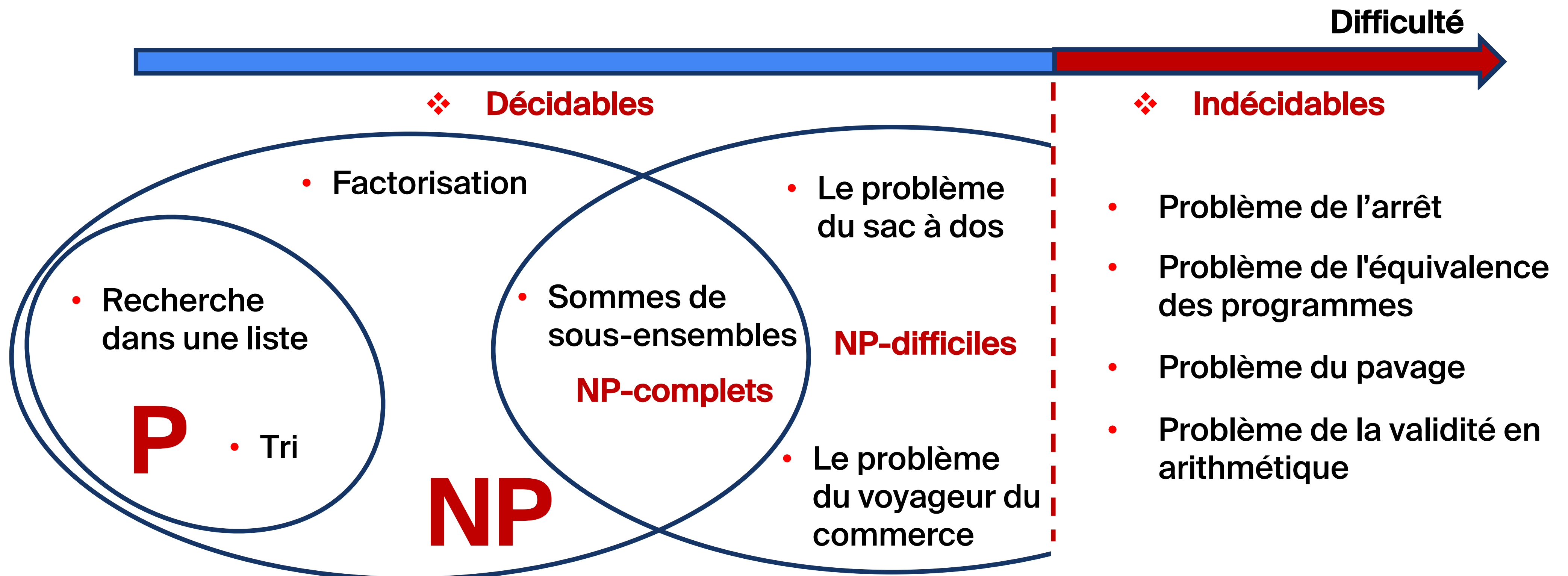
NP

- La **classe NP** est l'ensemble des problèmes pour lesquels, si on nous propose une solution du problème, il est alors possible de
  vérifier en temps polynomial si celle-ci en est une ou pas.

- **Remarques** :
  - NP ne veut pas dire « non-polynomial »
  - P ⊂ NP : Il est plus facile de vérifier une solution que d'en proposer une !

  - Tous les problèmes appartenant à la classe P appartiennent également à la classe NP !

ICC-T 13 : Classification des problèmes



rafael.pires@epfl.ch

EPFL

Merci