Master Informatique - M1 - UE Complexité
Chapitre 3 : Problèmes et Complexité des problèmes

Philippe Jégou

Laboratoire d'Informatique et Systèmes - LIS - UMR CNRS 7020

Équipe COALA - COntraintes, ALgorithmes et Applications

(Algorithmique et Complexité de l'Intelligence Artificielle)

Campus de Saint-Jérôme

Département Informatique et Interactions

Faculté des Sciences

Université d'Aix-Marseille

philippe.jegou@univ-amu.fr

21 septembre 2022

Plan

Plan

Problèmes de décision

Le problème CLIQUE

CLIQUE

Donnée (ou Instance) : Un graphe non-orienté $G = (S, A)$ et un entier k

Question : G possède-t-il une clique de taille k ou plus ?

Une clique de taille k dans un graphe non-orienté est un ensemble de k sommets tous mutuellement voisins (un sous-graphe complet)

Problèmes de décision

Le problème CLIQUE

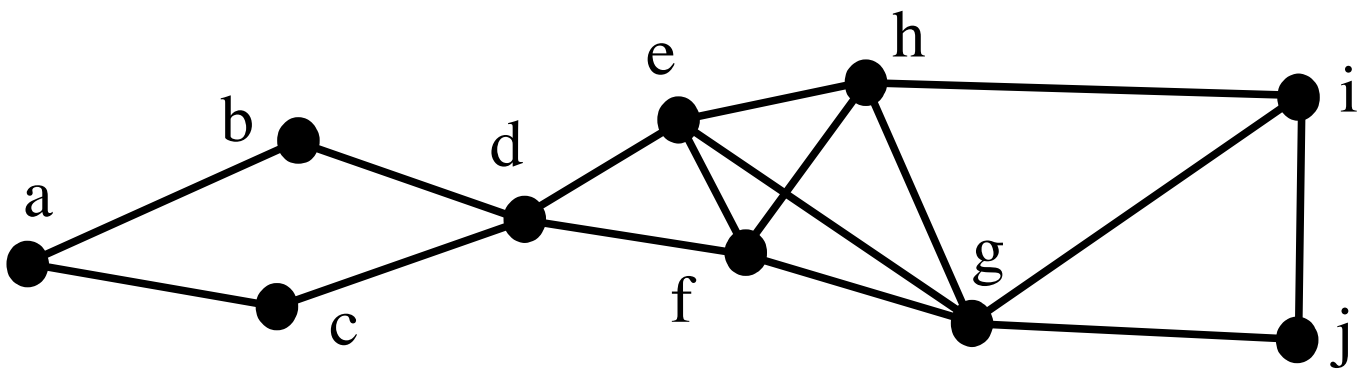
CLIQUE

Donnée (ou Instance) : Un graphe non-orienté $G = (S, A)$ et un entier k

Question : G possède-t-il une clique de taille k ou plus ?

Une clique de taille k dans un graphe non-orienté est un ensemble de k sommets tous mutuellement voisins (un sous-graphe complet)

Exemple d'instance du problème CLIQUE : un couple (G, k) soit un couple (graphe, entier) comme le graphe G ci-dessous et l'entier $k = 4$



La réponse à la question du problèmes de décision ici est...

.

Problèmes de décision

Le problème CLIQUE

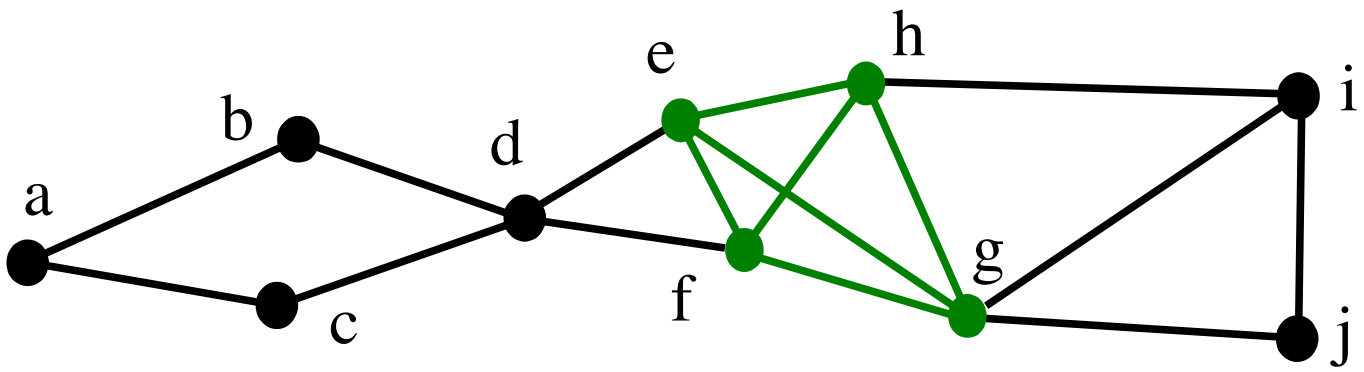
CLIQUE

Donnée (ou instance) : Un graphe non-orienté $G = (S, A)$ et un entier k

Question : G possède-t-il une clique de taille k ou plus ?

Une clique de taille k dans un graphe non-orienté est un ensemble de k sommets tous mutuellement voisins (un sous-graphe complet)

Exemple d'instance du problème CLIQUE : un couple (G, k) soit un couple (graphe, entier) comme le graphe G ci-dessous et l'entier $k = 4$



La réponse à la question du problèmes de décision ici est **OUI**

Le couple $(G, 4)$ est appelé **instance positive** du problème CLIQUE

Problèmes de décision

Le problème CLIQUE

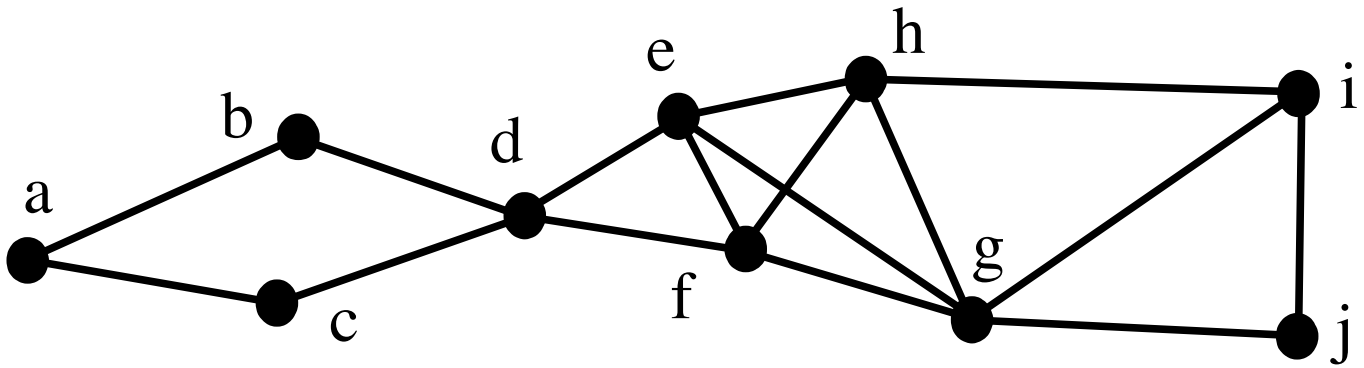
CLIQUE

Donnée (ou instance) : Un graphe non-orienté $G = (S, A)$ et un entier k

Question : G possède-t-il une clique de taille k ou plus ?

Une clique de taille k dans un graphe non-orienté est un ensemble de k sommets tous mutuellement voisins (un sous-graphe complet)

Autre exemple d'instance du problème CLIQUE : le même graphe G et l'entier $k = 5$



La réponse à la question du problèmes de décision ici est...

.

Problèmes de décision

Le problème CLIQUE

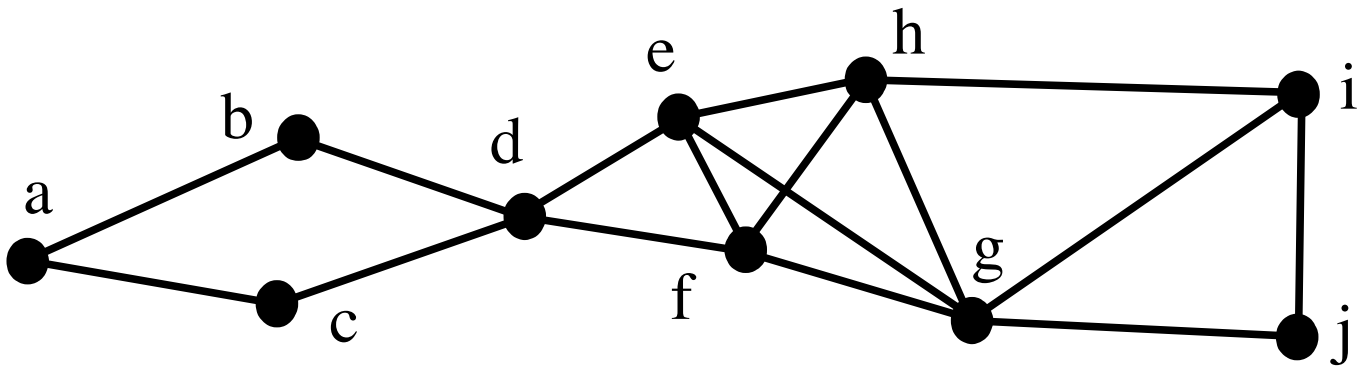
CLIQUE

Donnée (ou instance) : Un graphe non-orienté $G = (S, A)$ et un entier k

Question : G possède-t-il une clique de taille k ou plus ?

Une clique de taille k dans un graphe non-orienté est un ensemble de k sommets tous mutuellement voisins (un sous-graphe complet)

Autre exemple d'instance du problème CLIQUE : le même graphe G et l'entier $k = 5$



La réponse à la question du problèmes de décision ici est **NON**

Le couple $(G, 5)$ est appelé **instance négative** du problème CLIQUE

Problèmes de décision

Le problème CLIQUE

CLIQUE
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : G possède-t-il une clique de taille k ou plus ?

Définition

Un problème de décision Π est défini par un couple (*Donnée*, *Question*) :

- **Donnée** également appelée **instance**, voire **entrée**, qui est une définition générique des jeux de données possibles
- **Question** qui est une question portant sur l'instance et dont la réponse est soit **oui** soit **non**

Problème CLIQUE : un problème de décision puisque la question posée est à réponse oui/non.

Différents types de problèmes

Il existe différents types de problèmes :

Définition

À partir d'un problème de décision, on peut définir :

- **problème de recherche** : trouver une solution s'il en existe une
- **problème d'optimisation** : trouver la meilleure solution s'il en existe
- **problème de dénombrement** : trouver le nombre de solutions
- **problème d'énumération** : trouver (lister) toutes les solutions

Différents types de problèmes

Il existe différents types de problèmes :

Définition

À partir d'un problème de décision, on peut définir :

- **problème de recherche** : trouver une solution s'il en existe une
- **problème d'optimisation** : trouver la meilleure solution s'il en existe
- **problème de dénombrement** : trouver le nombre de solutions
- **problème d'énumération** : trouver (lister) toutes les solutions

2 remarques importantes :

- Terminologie : Garey et Jonhson (en 1979) appellent "énumération" ce que nous appelons "dénombrement" (aussi appelé *comptage*) ; notre terminologie apparaît aujourd'hui comme plus pertinente.

Différents types de problèmes

Il existe différents types de problèmes :

Définition

À partir d'un problème de décision, on peut définir :

- **problème de recherche** : trouver une solution s'il en existe une
- **problème d'optimisation** : trouver la meilleure solution s'il en existe
- **problème de dénombrement** : trouver le nombre de solutions
- **problème d'énumération** : trouver (lister) toutes les solutions

2 remarques importantes :

- Terminologie : Garey et Jonhson (en 1979) appellent "énumération" ce que nous appelons "dénombrement" (aussi appelé *comptage*) ; notre terminologie apparaît aujourd'hui comme plus pertinente.
- Rigueur mathématique : nous ne définirons pas ces notions de façon formelle dans ce cours (perte de temps inutile en M1) ; certaines définitions seront donc délibérément intuitives plutôt que rigoureuses mathématiquement.

Différents types de problèmes : Recherche

Variations à partir du problème CLIQUE

CLIQUE
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : G possède-t-il une clique de taille k ou plus ?

Solution ? Une clique de taille k :

Différents types de problèmes : Recherche

Variations à partir du problème CLIQUE

CLIQUE
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : G possède-t-il une clique de taille k ou plus ?

Solution ? Une clique de taille k :

Problème de recherche :

CLIQUE-Recherche
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : Si G possède une clique de taille k, la trouver

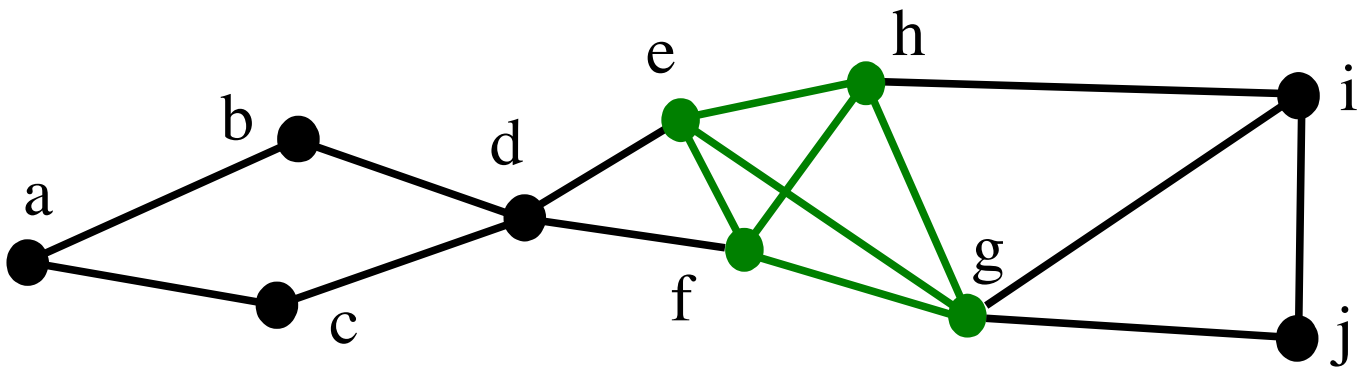
Ce n'est pas un problème de décision car la réponse n'est pas OUI ou NON, mais une solution (s'il en existe une).

Différents types de problèmes : Recherche

Variations à partir du problème CLIQUE

CLIQUE-Recherche
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : Si G possède une clique de taille k, la trouver.

Exemple d'une instance du problème CLIQUE-Recherche : le graphe G ci-dessous et l'entier $k = 4$



La réponse à la question du problème est la clique $\{e, f, g, h\}$

Différents types de problèmes : Optimisation

Variations à partir du problème CLIQUE

CLIQUE
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k
Question : G possède-t-il une clique de taille k ou plus ?

Problème d'optimisation :

CLIQUE-MAX
Donnée : Un graphe non-orienté $G = (S, A)$
Question : Trouver une clique de G de taille maximum.

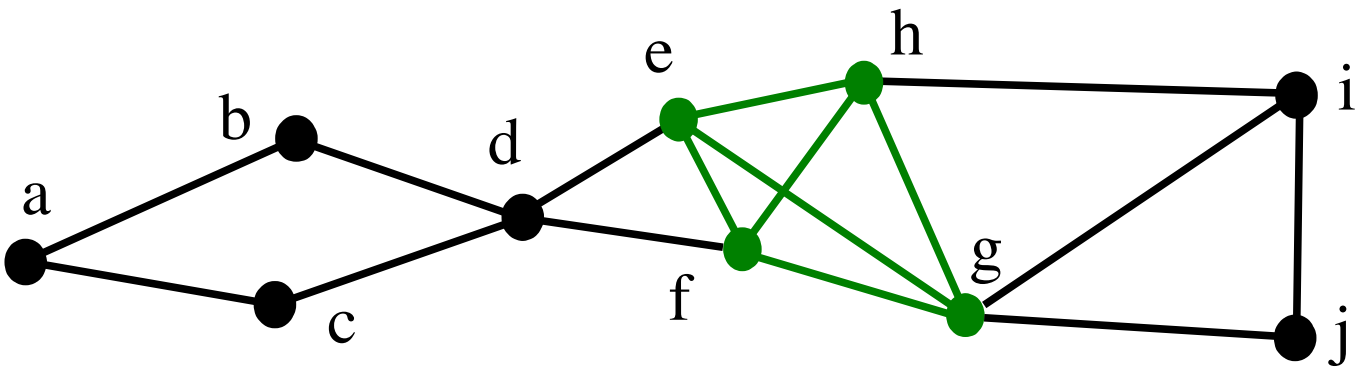
Ce n'est pas un problème de décision car la réponse n'est pas OUI ou NON, et on recherche la meilleure solution (selon un certain critère : la plus grande clique ici).

Différents types de problèmes : Optimisation

Problème d'optimisation

CLIQUE-MAX
Donnée : Un graphe non-orienté $G = (S, A)$.
Question : Trouver une clique de G de taille maximum.

Exemple d'une instance du problème CLIQUE-MAX : le graphe G ci-dessous



La réponse à la question du problème est la clique $\{e, f, g, h\}$

Différents types de problèmes : Énumération

Variations à partir du problème CLIQUE

CLIQUE
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : G possède-t-il une clique de taille k ou plus ?

Problème d'énumération :

CLIQUE-ENUM
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : Trouver toutes les cliques de G de taille k.

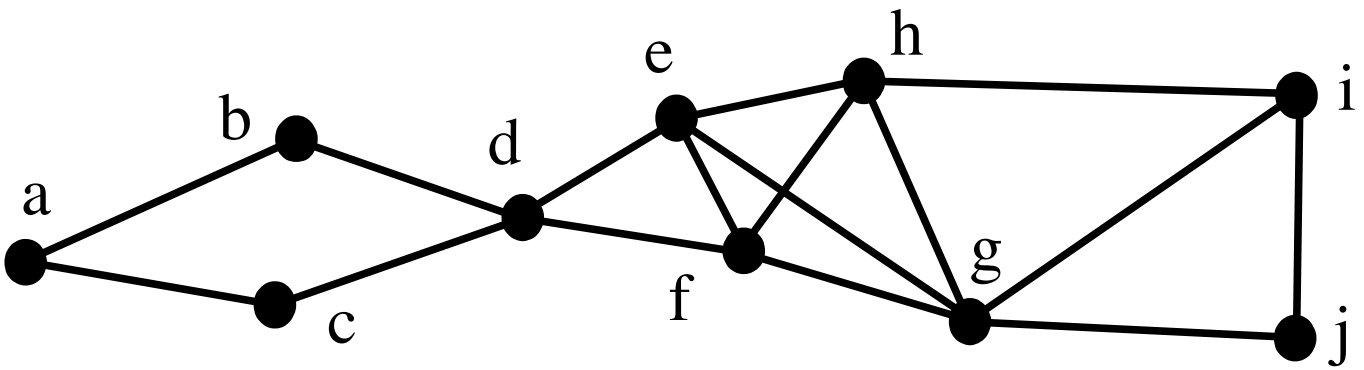
Ce n'est pas un problème de décision, ni de recherche, ni d'optimisation.

Différents types de problèmes : Énumération

Problème d'énumération

CLIQUE-ENUM
Donnée : Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : Trouver toutes les cliques de G de taille k.

Exemple d'une instance du problème CLIQUE-ENUM : le graphe G ci-dessous et l'entier $k = 3$



La réponse à la question du problème est l'ensemble :
$\{ \{d, e, f\}, \{e, f, g\}, \{e, f, h\}, \{e, g, h\}, \{f, g, h\}, \{g, h, i\}, \{g, i, j\} \}$

Différents types de problèmes : Dénombrement

Variations à partir du problème CLIQUE

CLIQUE
Donnée: Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : G possède-t-il une clique de taille k ou plus ?

Problème de dénombrement (ou comptage) :

#CLIQUE
Donnée: Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : Trouver le nombre de cliques de G de taille k.

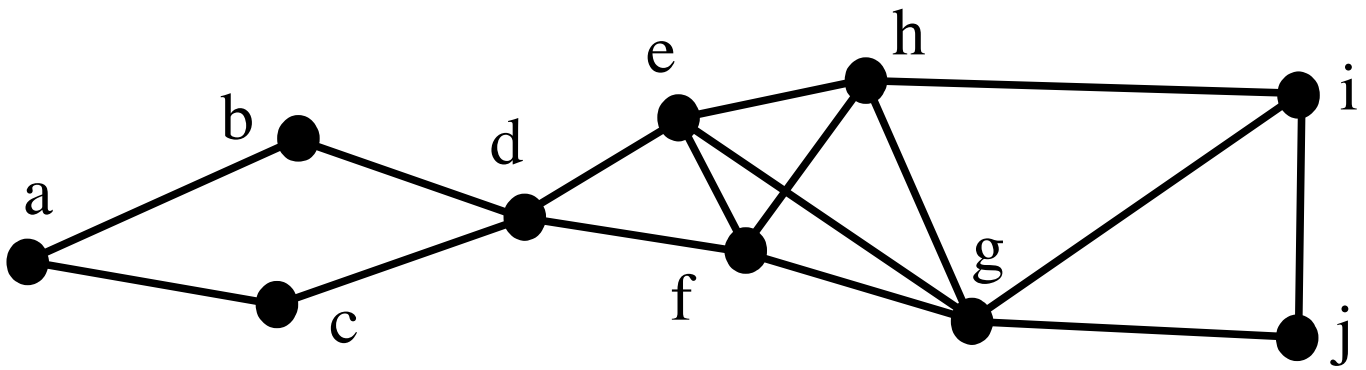
Ce n'est pas un problème de décision, ni de recherche, ni d'optimisation, ni d'énumération.

Différents types de problèmes : Dénombrement

Problème de dénombrement (ou comptage)

#CLIQUE
Donnée: Un graphe non-orienté $G = (S, A)$ et un entier k.
Question : Trouver le nombre de cliques de G de taille k.

Exemple d'une instance du problème #CLIQUE : le graphe G ci-dessous et l'entier $k = 3$



La réponse à la question du problème est **7** car il y a 7 cliques de taille 3

Relations de difficultés relatives entre types de problèmes

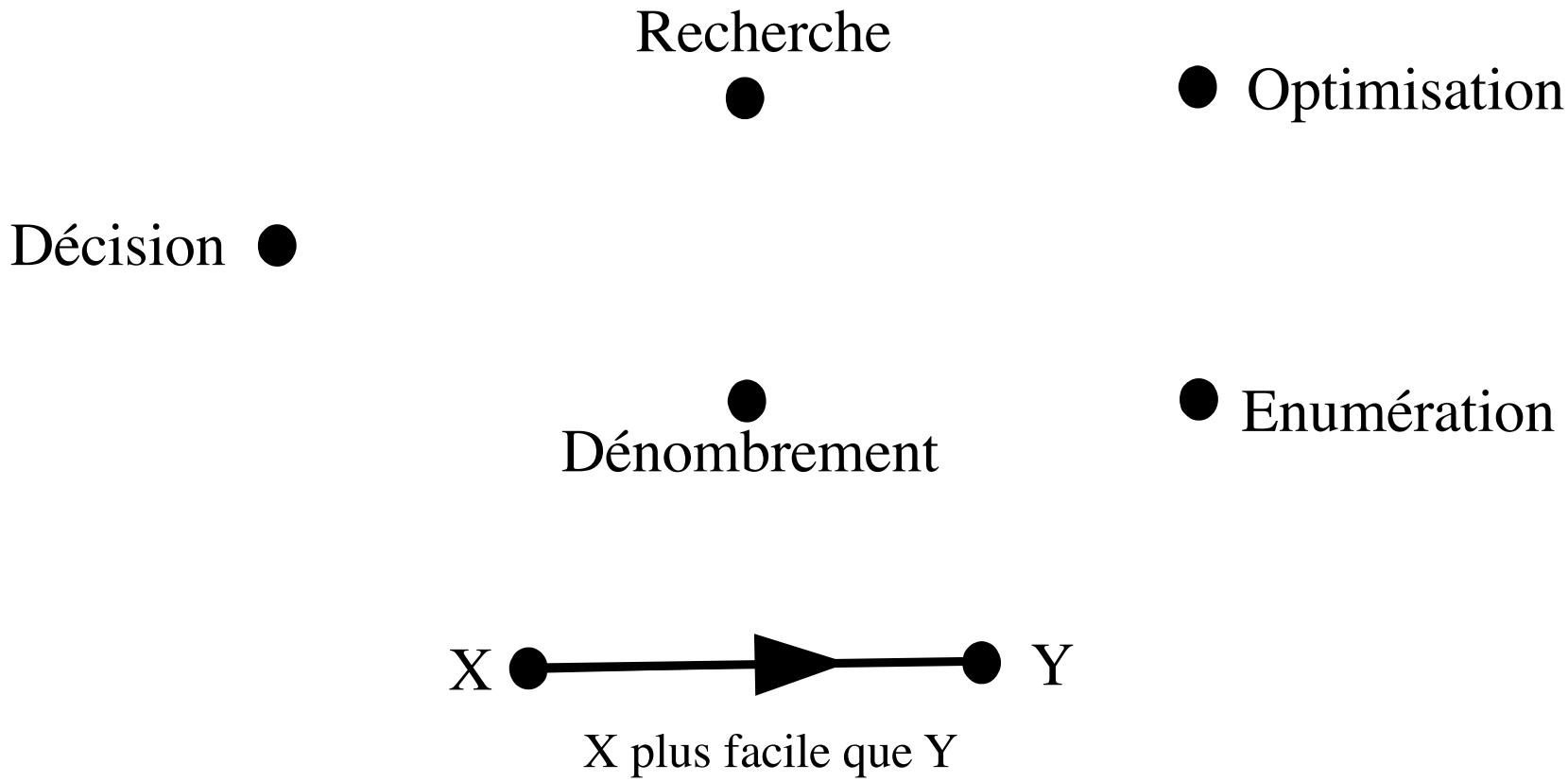
Difficultés relatives entre types de problèmes

- Il existe des relations entre ces différents types de problèmes en termes de difficulté
- Pour un même problème de décision on peut comparer la difficulté (sous la réserve des certains critères d'optimisation) :

Relations de difficultés relatives entre types de problèmes

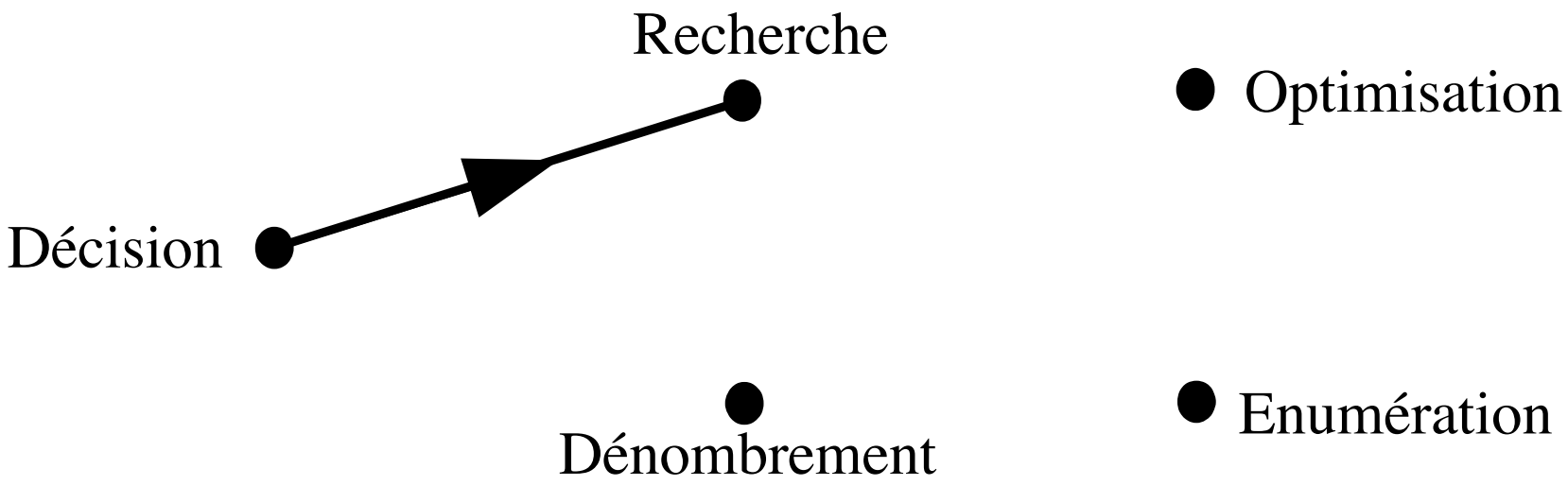
Difficultés relatives entre types de problèmes

- Il existe des relations entre ces différents types de problèmes en termes de difficulté
- Pour un même problème de décision on peut comparer la difficulté (sous la réserve des certains critères d'optimisation) :



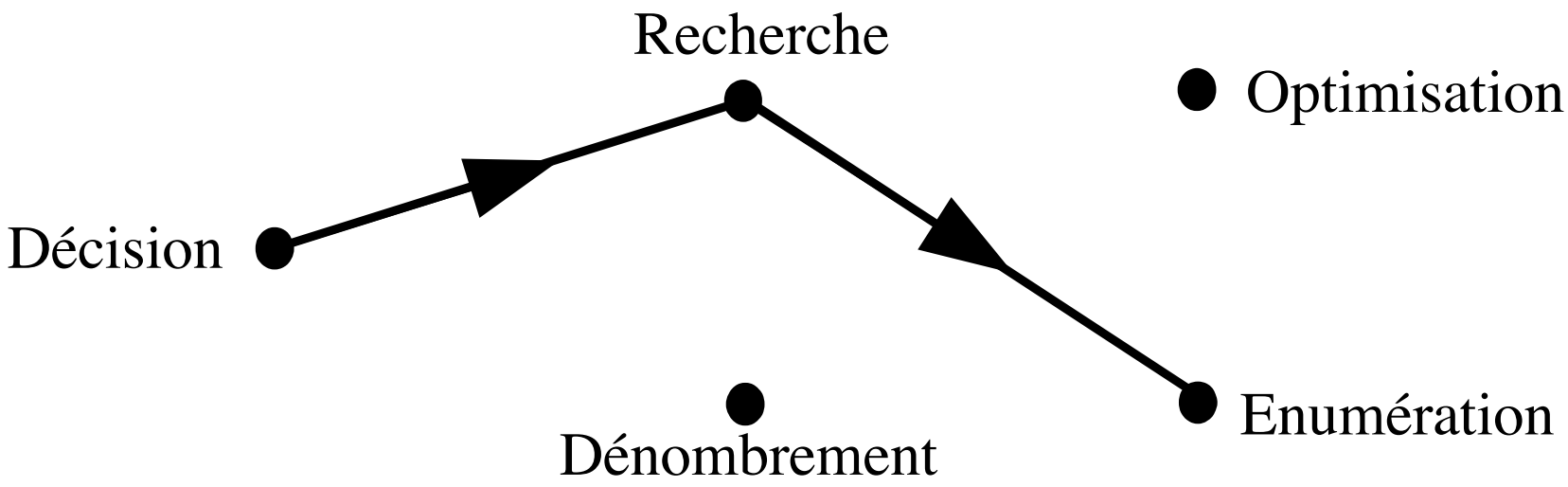
Relations de difficultés relatives entre types de problèmes

Problème de recherche plus difficile que le problème de décision (si on trouve une solution, on sait qu'il en existe une)



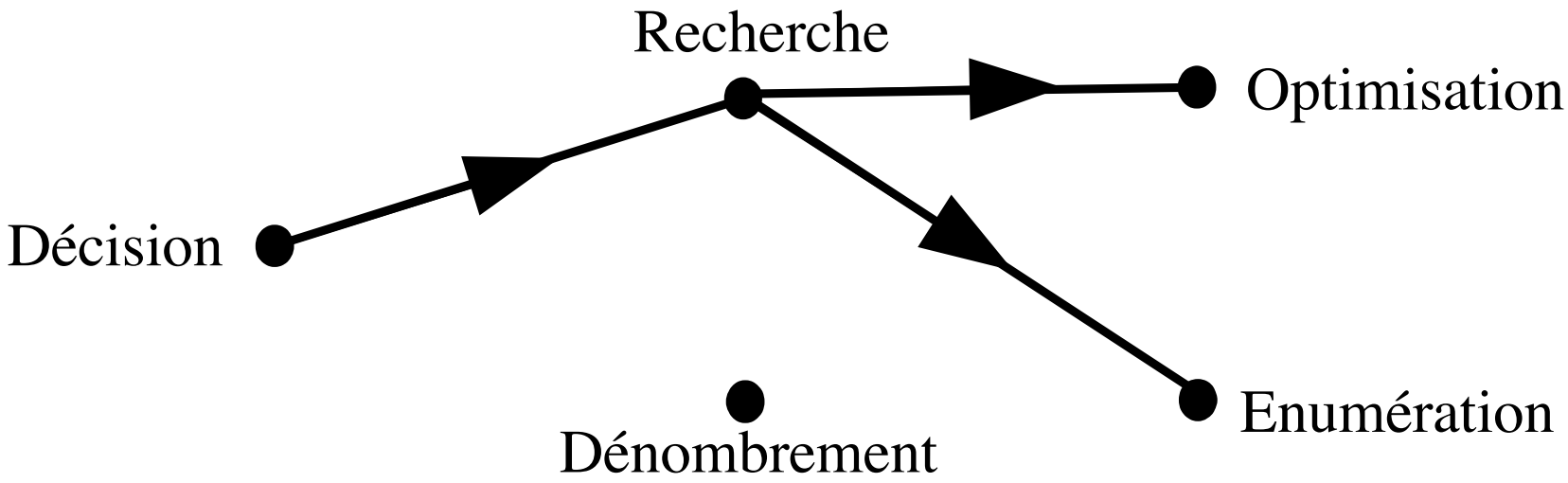
Relations de difficultés relatives entre types de problèmes

Problème d'énumération plus difficile que le problème de recherche (si on trouve toutes les solutions, on en trouve au moins une)



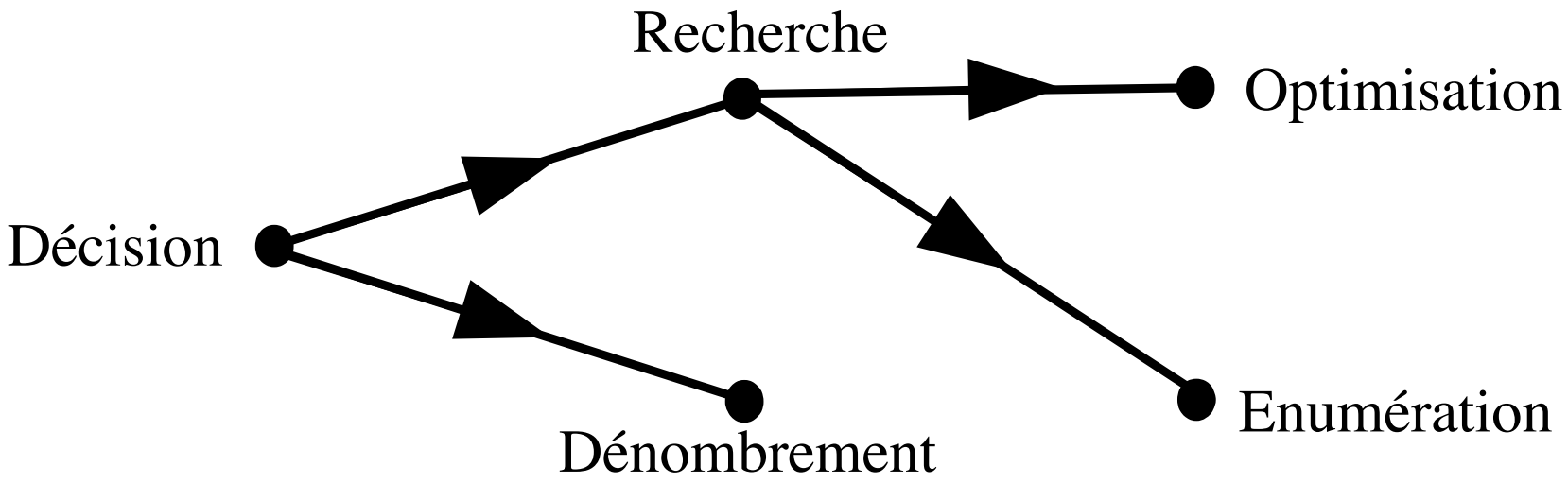
Relations de difficultés relatives entre types de problèmes

Problème d'optimisation plus difficile que le problème de recherche (si on connaît une solution qui maximise le critère d'optimisation, on peut savoir si c'est une solution du problème de recherche)



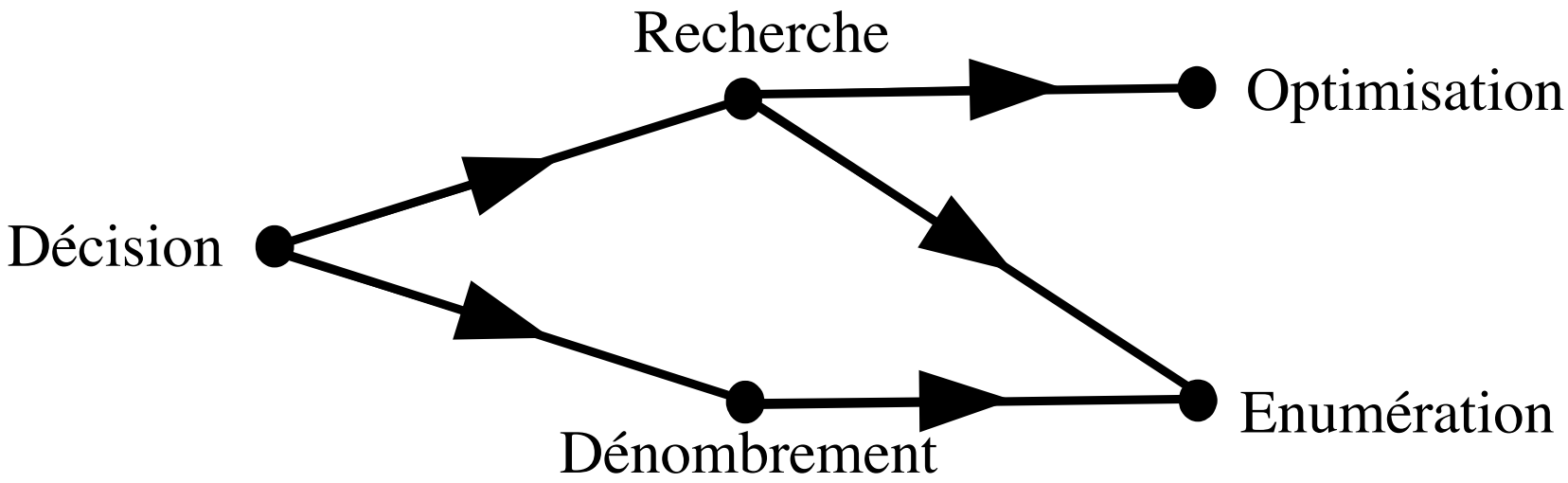
Relations de difficultés relatives entre types de problèmes

Dénombrement plus difficile que le problème de décision
(si on trouve le nombre de solutions, on sait s'il en existe une...)



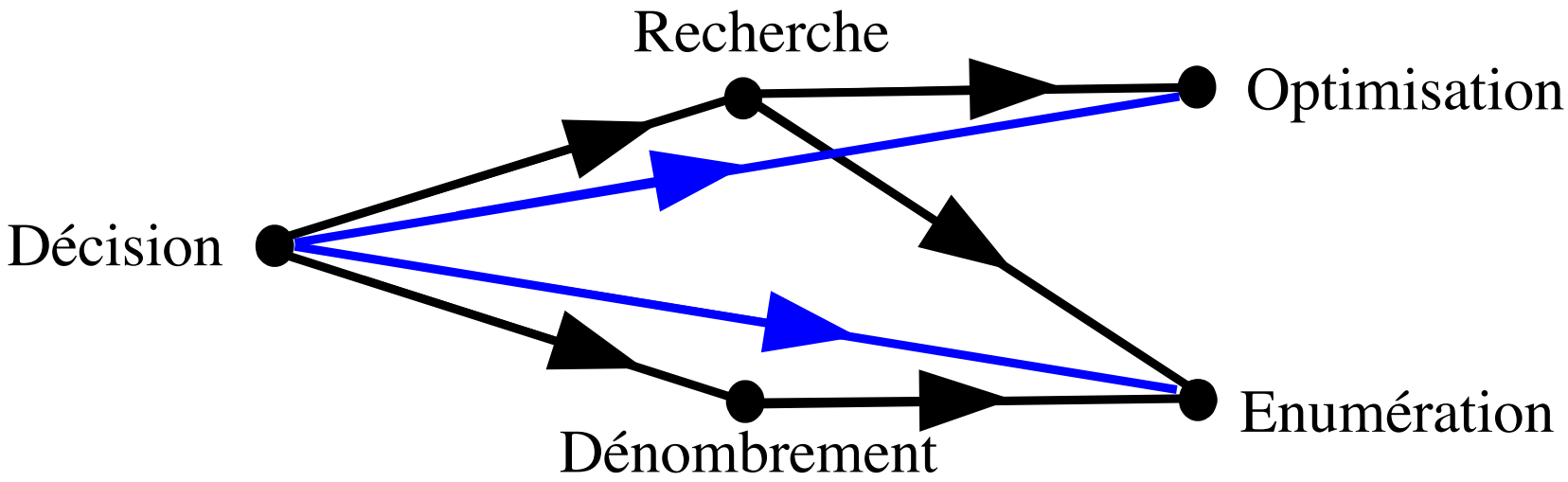
Relations de difficultés relatives entre types de problèmes

Énumération plus difficile que le problème de dénombrement associé (si on liste toutes les solutions, c'est "facile" de les compter...)



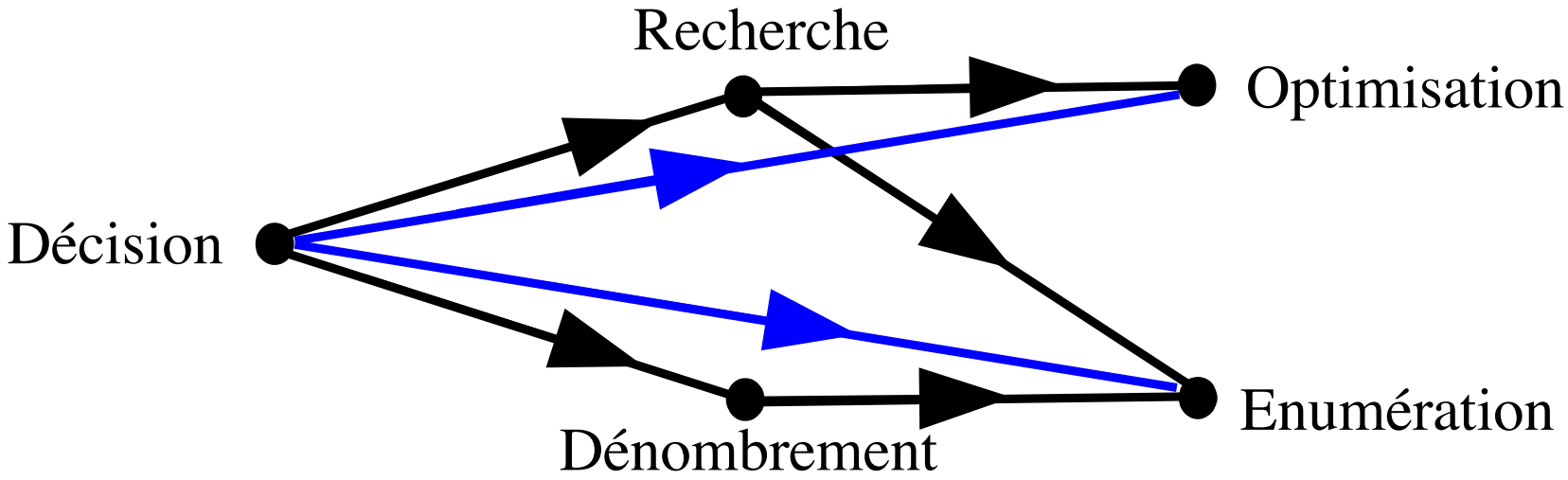
Relations de difficultés relatives entre types de problèmes

Remarque : ce graphe des difficultés est complétable par transitivité



Relations de difficultés relatives entre types de problèmes

Remarque : ce graphe des difficultés est complétable par transitivité

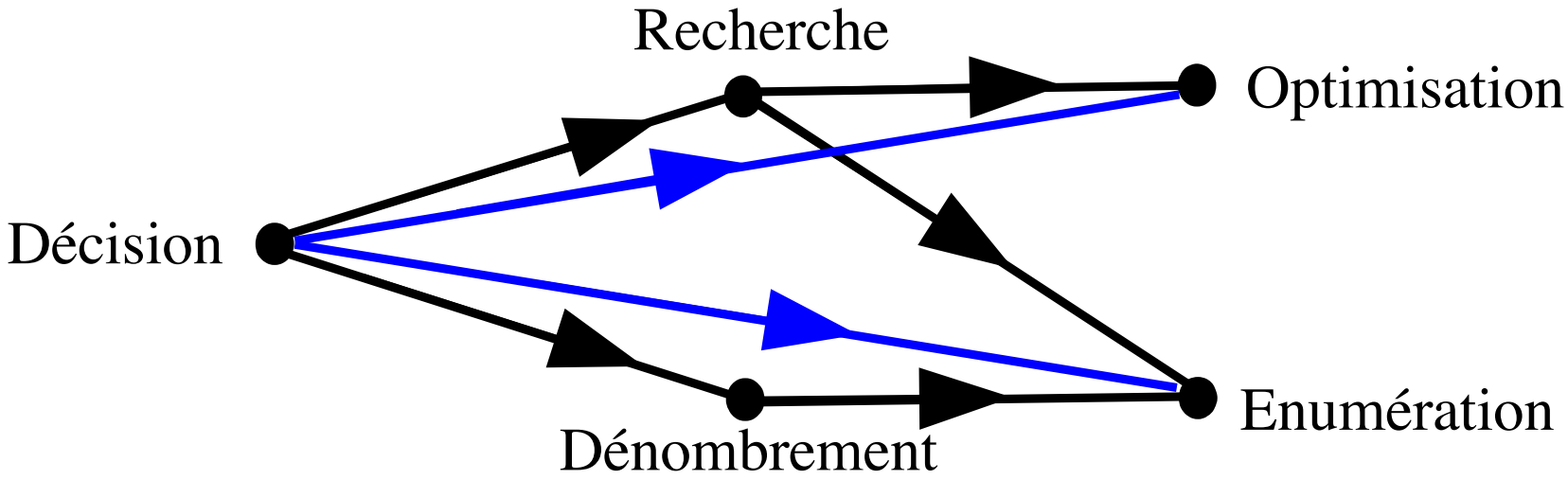


Dans la "vraie vie", quel est l'intérêt des problèmes de décision ?

Quel est l'intérêt de savoir qu'il existe une solution sans en disposer ?

Relations de difficultés relatives entre types de problèmes

Remarque : ce graphe des difficultés est complétable par transitivité



Dans la "vraie vie", quel est l'intérêt des problèmes de décision ?

Quel est l'intérêt de savoir qu'il existe une solution sans en disposer ?

Mais sur le plan théorique, c'est très important :

- si on arrive à montrer qu'un problème de décision est "difficile" alors on aura montré que les autres problèmes sont "difficiles"
- et les problèmes de décision sont *a priori* plus simples à étudier

Plan

Complexité d'un problème π (lire "Pi")

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Complexité d'un problème π (lire "Pi")

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Définition

Etant donnée $\mathcal{C}$ la classe d'algorithmes qui résolvent π, si $A \in \mathcal{C}$ et $T_A(I)$ est le nombre d'opérations fondamentales exécutées par l'algorithme A sur la donnée (instance) I, alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est l'ensemble des données du problème π de taille n.

(sans incidence sur le propos, ici T_A est définie de D_π vers $\mathbb{N}$ et non de $\mathbb{N}$ vers $\mathbb{R}^+$)

Complexité d'un problème π (lire "Pi")

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Définition

Etant donnée $\mathcal{C}$ la classe d'algorithmes qui résolvent π, si $A \in \mathcal{C}$ et $T_A(I)$ est le nombre d'opérations fondamentales exécutées par l'algorithme A sur la donnée (instance) I, alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est l'ensemble des données du problème π de taille n.

(sans incidence sur le propos, ici T_A est définie de D_π vers $\mathbb{N}$ et non de $\mathbb{N}$ vers $\mathbb{R}^+$)

Décryptage :

- $\max_{I \in D_\pi^n} T_A(I)$: capte le pire des cas
- $\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$: considère le meilleur algorithme

Complexité d'un problème

Remarques très importantes sur la complexité d'un problème :

Définition

Etant donnée ... alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est...

Complexité d'un problème

Remarques très importantes sur la complexité d'un problème :

Définition

Etant donnée ... alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est...

- La complexité d'un problème n'est pas nécessairement celle du meilleur algorithme connu pour le résoudre !

Complexité d'un problème

Remarques très importantes sur la complexité d'un problème :

Définition

Etant donnée ... alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est...

- La complexité d'un problème n'est pas nécessairement celle du meilleur algorithme connu pour le résoudre !
- **Seule la notion d'existence d'algorithme est considérée ici :** Il n'est pas nécessaire de connaître un algorithme particulier, mais tout simplement, d'en connaître l'existence (cf. $\exists A \in \mathcal{C}$) !

Complexité d'un problème

Remarques très importantes sur la complexité d'un problème :

Définition

Etant donnée ... alors **la complexité du problème** π est :

$$\min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I)$$

où D_π^n est...

- La complexité d'un problème n'est pas nécessairement celle du meilleur algorithme connu pour le résoudre !
- **Seule la notion d'existence d'algorithme est considérée ici :** Il n'est pas nécessaire de connaître un algorithme particulier, mais tout simplement, d'en connaître l'existence (cf. $\exists A \in \mathcal{C}$) !
- Résultat étonnant de Fellow et Langston vers 1985 : ils proposent une preuve dite *non constructive* de complexité en démontrant l'existence d'un algorithme de résolution d'une certaine complexité sans l'exhiber

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Parmi les rares résultats connus : le tri par comparaison
tri par comparaison : tri basé sur la comparaison entre éléments ($\neq$ tris par adressage)

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Parmi les rares résultats connus : le tri par comparaison
tri par comparaison : tri basé sur la comparaison entre éléments ($\neq$ tris par adressage)

Avant de voir la preuve : le problème du tri par comparaison

TRI PAR COMPARAISON
Donnée : Une séquence de valeurs $(x_1, x_2, \ldots x_n)$
Question : Trouver une permutation des valeurs dans l'ordre croissant

avec la contrainte de n'utiliser que des comparaisons 2 à 2 entre valeurs et donc sans hypothèse sur ces valeurs (sauf qu'elles sont comparables, cf. $\exists$ relation d'ordre)

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Parmi les rares résultats connus : le tri par comparaison
tri par comparaison : tri basé sur la comparaison entre éléments ($\neq$ tris par adressage)

Comment procède la preuve pour le tri par comparaison ?

1. on se rappelle que le tri par fusion est en $\Theta(n.log(n))$

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Parmi les rares résultats connus : le tri par comparaison
tri par comparaison : tri basé sur la comparaison entre éléments ($\neq$ tris par adressage)

Comment procède la preuve pour le tri par comparaison ?

1. on se rappelle que le tri par fusion est en $\Theta(n.log(n))$
2. on prouve qu'il n'existe pas d'algorithme de complexité inférieure

Complexité d'un problème : le tri par comparaison

Un constat : rares sont les problèmes dont la complexité est connue !

Une question : Comment déterminer la complexité d'un problèmes ?

1. Démontrer l'existence d'un algorithme et donner sa complexité
   (en général, pour cela, on exhibe un algorithme mais la preuve d'existence suffit..)
2. Démontrer qu'il n'existe pas d'algorithme de complexité inférieure
   tâche bien plus complexe car il existe théoriquement une infinité d'algorithmes pour résoudre un problème résoluble (calculable...)

Parmi les rares résultats connus : le tri par comparaison
tri par comparaison : tri basé sur la comparaison entre éléments ($\neq$ tris par adressage)

Comment procède la preuve pour le tri par comparaison ?

1. on se rappelle que le tri par fusion est en $\Theta(n.log(n))$
2. on prouve qu'il n'existe pas d'algorithme de complexité inférieure

Ce point 2 : c'est toute la difficulté de la preuve !

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A
- Exécutions possibles de A ?

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A
- Exécutions possibles de A ?

  $\rightarrow$ selon les différentes configurations de la donnée en entrée :

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A
- Exécutions possibles de A ?

  $\rightarrow$ selon les différentes configurations de la donnée en entrée :

  - configurations possibles en entrée : les $n!$ permutations possibles

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A
- Exécutions possibles de A ?

  $\rightarrow$ selon les différentes configurations de la donnée en entrée :

  - configurations possibles en entrée : les $n!$ permutations possibles
  - avec $n = 3$ et 3 valeurs possibles 5, 12 et 17, on a $n! = 3! = 6$ **possibilités** :

    $[5, 12, 17]$ ou $[5, 17, 12]$ ou $[12, 5, 17]$ ou $[12, 17, 5]$ ou $[17, 5, 12]$ ou $[17, 12, 5]$

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Arbre de décision associé à un algorithme de tri A ?

- Est défini sur une donnée $(x_1, x_2, \ldots x_n)$ en entrée
- C'est un arbre binaire (en fait une "arborescence binaire")
- Arbre qui représente toutes les exécutions possibles de A
- Exécutions possibles de A ?

  $\rightarrow$ selon les différentes configurations de la donnée en entrée :

  - configurations possibles en entrée : les $n!$ permutations possibles
  - avec $n = 3$ et 3 valeurs possibles 5, 12 et 17, on a $n! = 3! = 6$ **possibilités** :

    $[5, 12, 17]$ ou $[5, 17, 12]$ ou $[12, 5, 17]$ ou $[12, 17, 5]$ ou $[17, 5, 12]$ ou $[17, 12, 5]$

  $\rightarrow$ **les n! exécutions seront toutes différentes**

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Définition

Un **arbre de décision associé à un algorithme de tri** A sur une donnée $(x_1, x_2, \ldots x_n)$ est un arbre binaire tel que :

- **noeuds internes :** état courant avec comparaison entre éléments.

  Si $x_i < x_j$, l'exécution se poursuit sur le sous-arbre gauche, sinon, sur le droit.

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Définition

Un **arbre de décision associé à un algorithme de tri** A sur une donnée $(x_1, x_2, \ldots x_n)$ est un arbre binaire tel que :

- **noeuds internes :** état courant avec comparaison entre éléments.
  Si $x_i < x_j$, l'exécution se poursuit sur le sous-arbre gauche, sinon, sur le droit.
- **feuilles de l'arbre :** résultats des différentes exécutions de A.

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Définition

Un **arbre de décision associé à un algorithme de tri** A sur une donnée $(x_1, x_2, \ldots x_n)$ est un arbre binaire tel que :

- **noeuds internes :** état courant avec comparaison entre éléments.
  Si $x_i < x_j$, l'exécution se poursuit sur le sous-arbre gauche, sinon, sur le droit.
- **feuilles de l'arbre :** résultats des différentes exécutions de A.
- à toute exécution de A correspond **1 branche de l'arbre**
  (plusieurs données peuvent conduire potentiellement à la même exécution).

Complexité d'un problème : le tri par comparaison

Prouver qu'il n'existe pas d'algorithme de complexité inférieure ?

Via la notion d'**arbre de décision associé à un algorithme de tri**

Définition

Un **arbre de décision associé à un algorithme de tri** A sur une donnée $(x_1, x_2, \ldots x_n)$ est un arbre binaire tel que :

- **noeuds internes :** état courant avec comparaison entre éléments.
  Si $x_i < x_j$, l'exécution se poursuit sur le sous-arbre gauche, sinon, sur le droit.
- **feuilles de l'arbre :** résultats des différentes exécutions de A.
- à toute exécution de A correspond **1 branche de l'arbre**
  (plusieurs données peuvent conduire potentiellement à la même exécution).
- **nombre de tests** $x_i < x_j$ exécutés par A sur une donnée :
  c'est égal à la longueur de la branche correspondant à cette exécution de A

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

Rappel du tri à bulles

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

Rappel du tri à bulles

En entree : un tableau t de n entiers

```
int i,j;
for (i=0; i < n-1; i=i+1)

    /* descente en t[i] de l'element minimum entre t[i] et t[n-1] */
    for (j=n-1; j > i; j=j-1)

        if ( t[j] < t[j-1] ) permuter(j,j-1,t);
```

À chaque étape (i=0, puis i=1, puis i=2, jusqu'à i=n-2) :

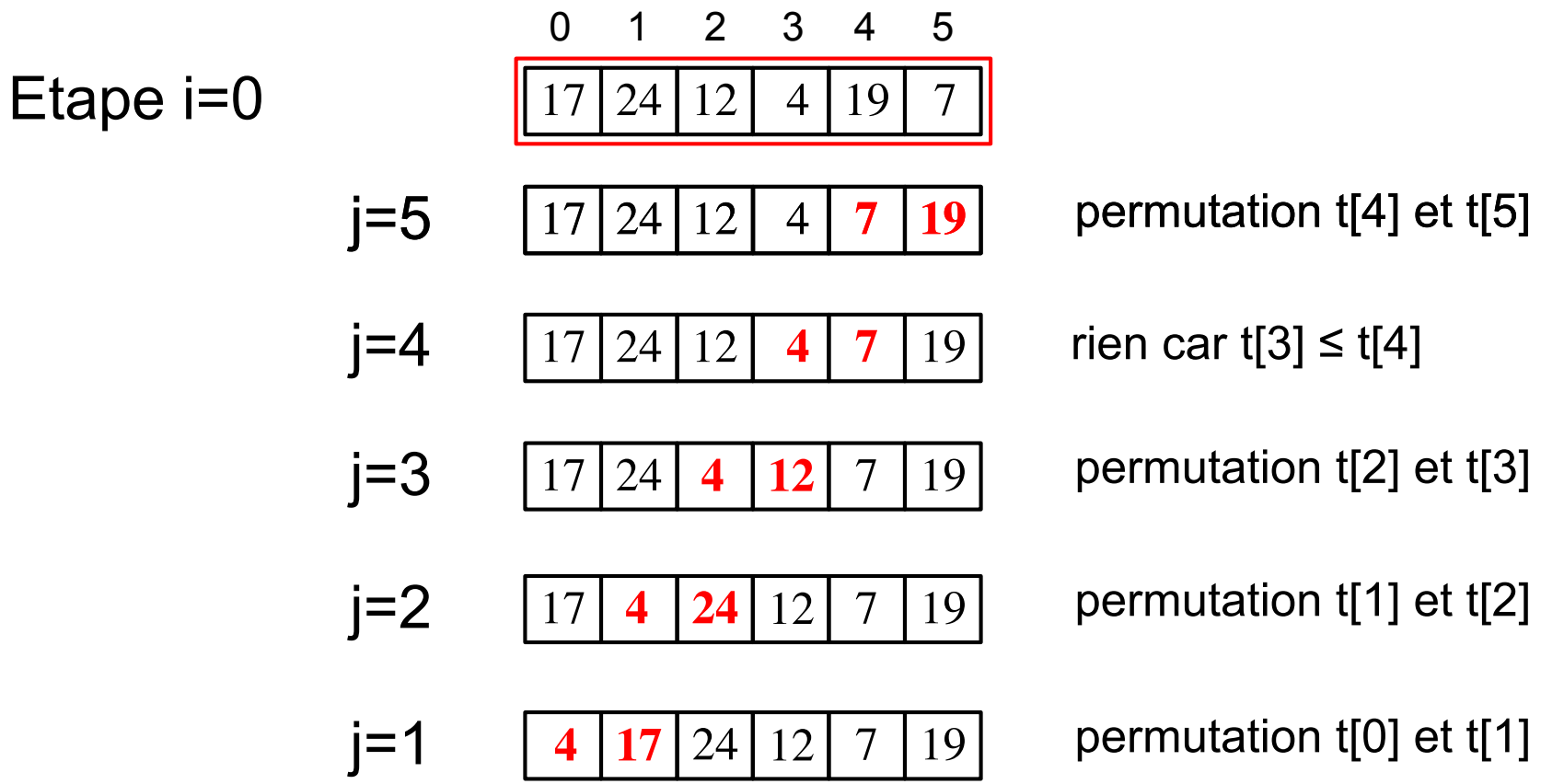
itération de j=n-1 jusqu'à j=i+1

pour positionner en t[i] l'élément minimum entre t[i] et t[n-1]

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

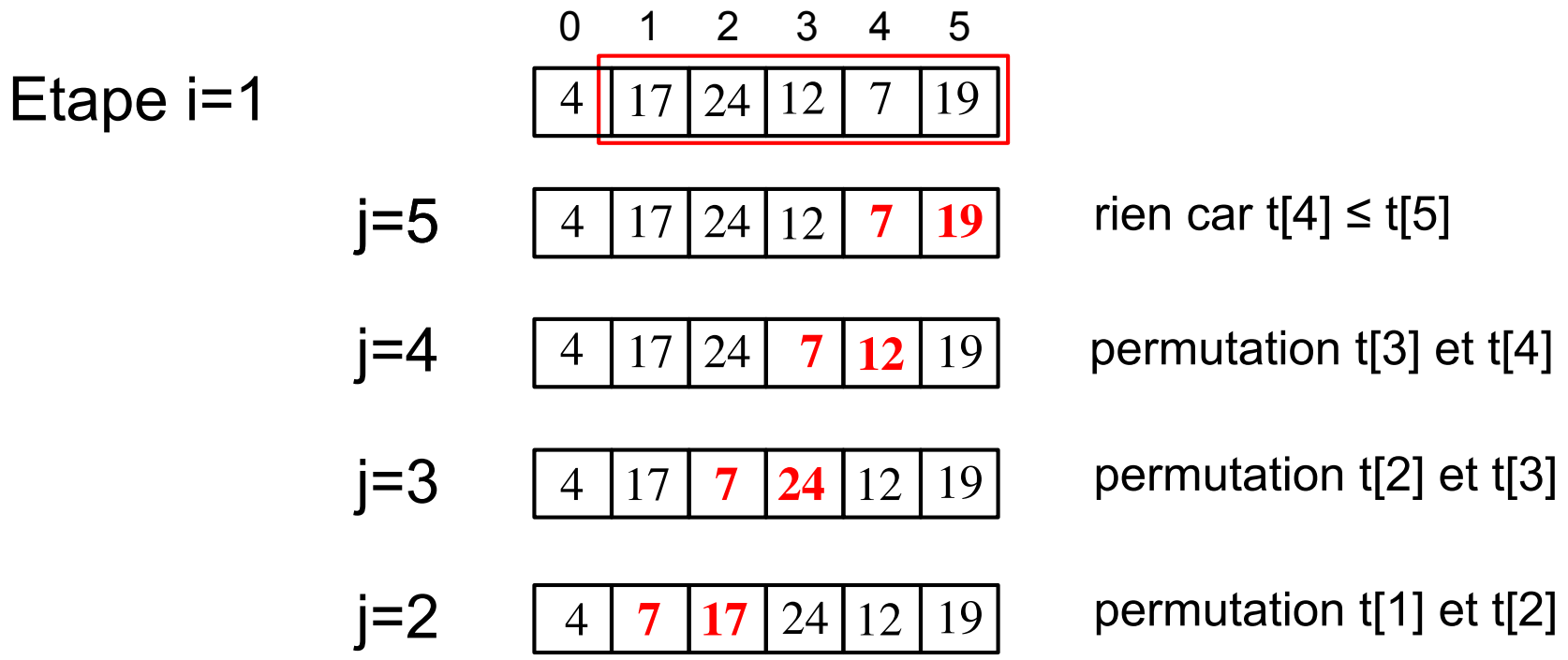
Rappel du tri à bulles



Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

Rappel du tri à bulles



Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

Rappel du tri à bulles

puis i=2, i=3 et i=4

Complexité d'un problème : le tri par comparaison

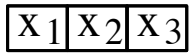
Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

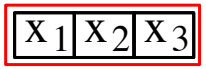


Racine : le tableau à trier

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

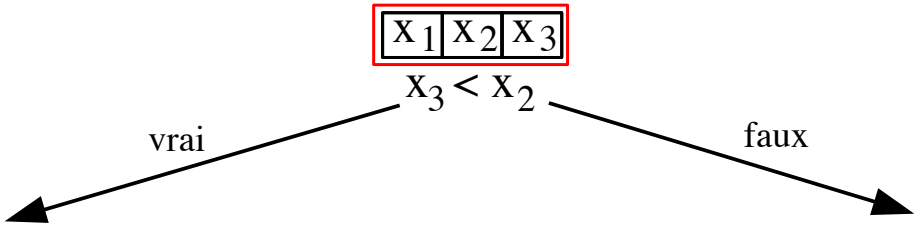


Tri à bulles : la 1^{ere} étape considère la section rouge

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

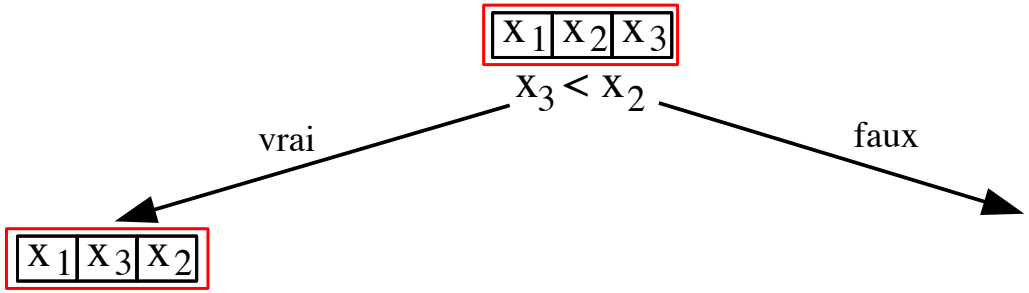


1^{ere} étape : comparaison $x_3 < x_2$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

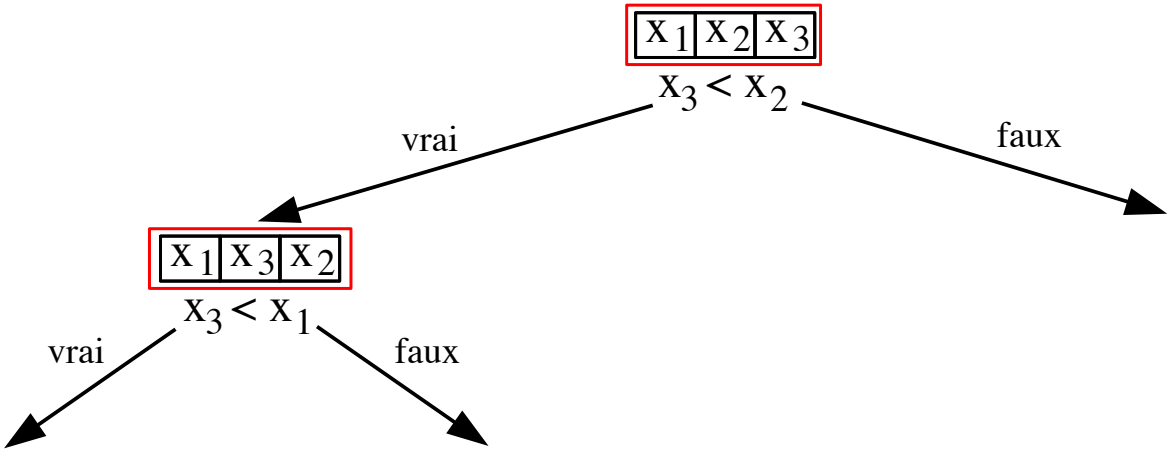


Test $x_3 < x_2$ **vrai :** permutation x_2 et x_3 puis poursuite branche gauche

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

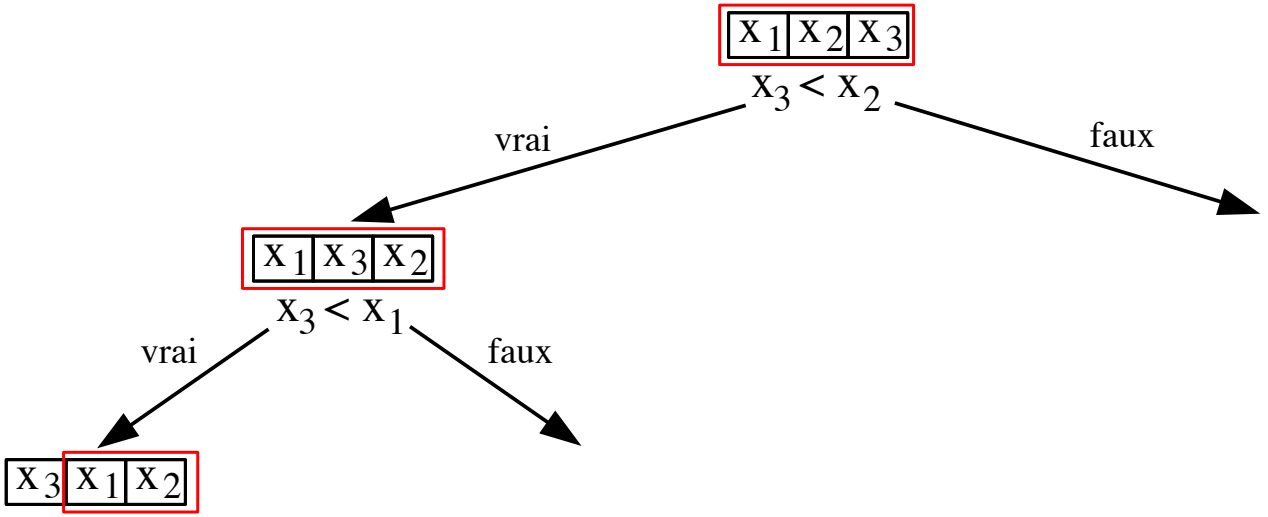


Encore 1^{ere} étape : comparaison $x_3 < x_1$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

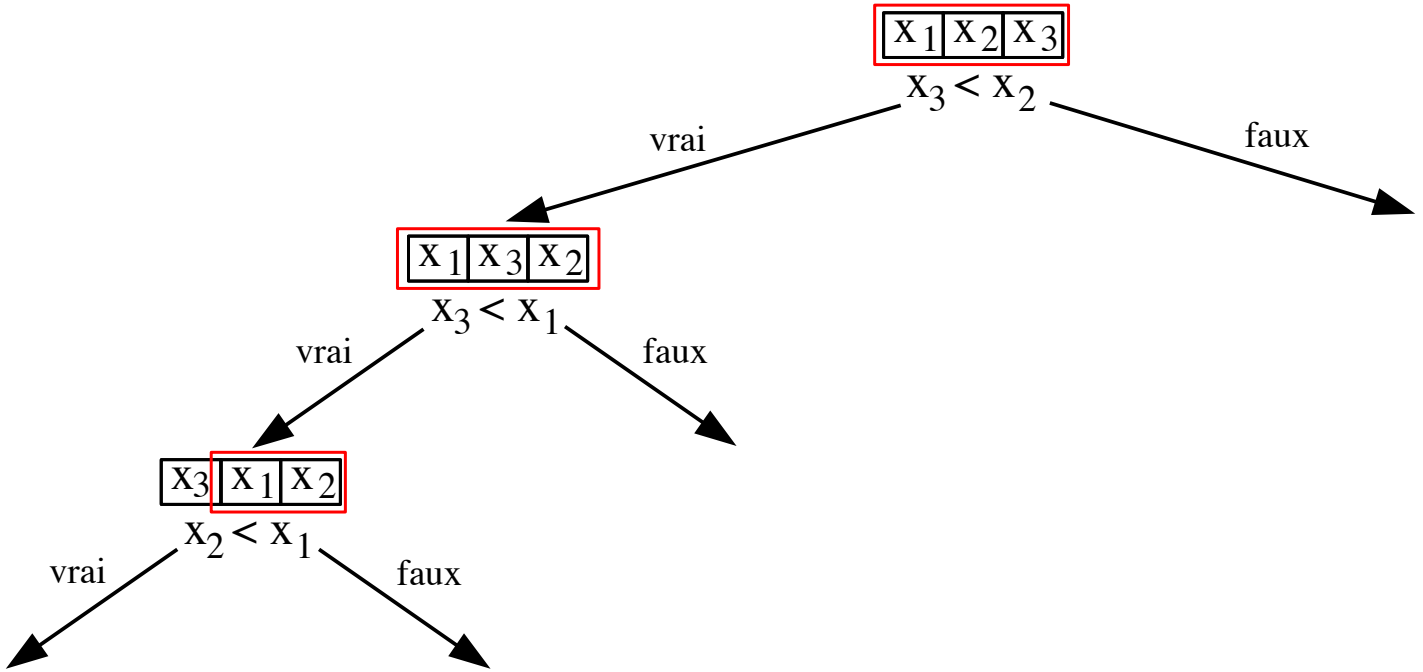


Test $x_3 < x_1$ **vrai :** permutation x_1 et x_3 puis poursuite branche gauche

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

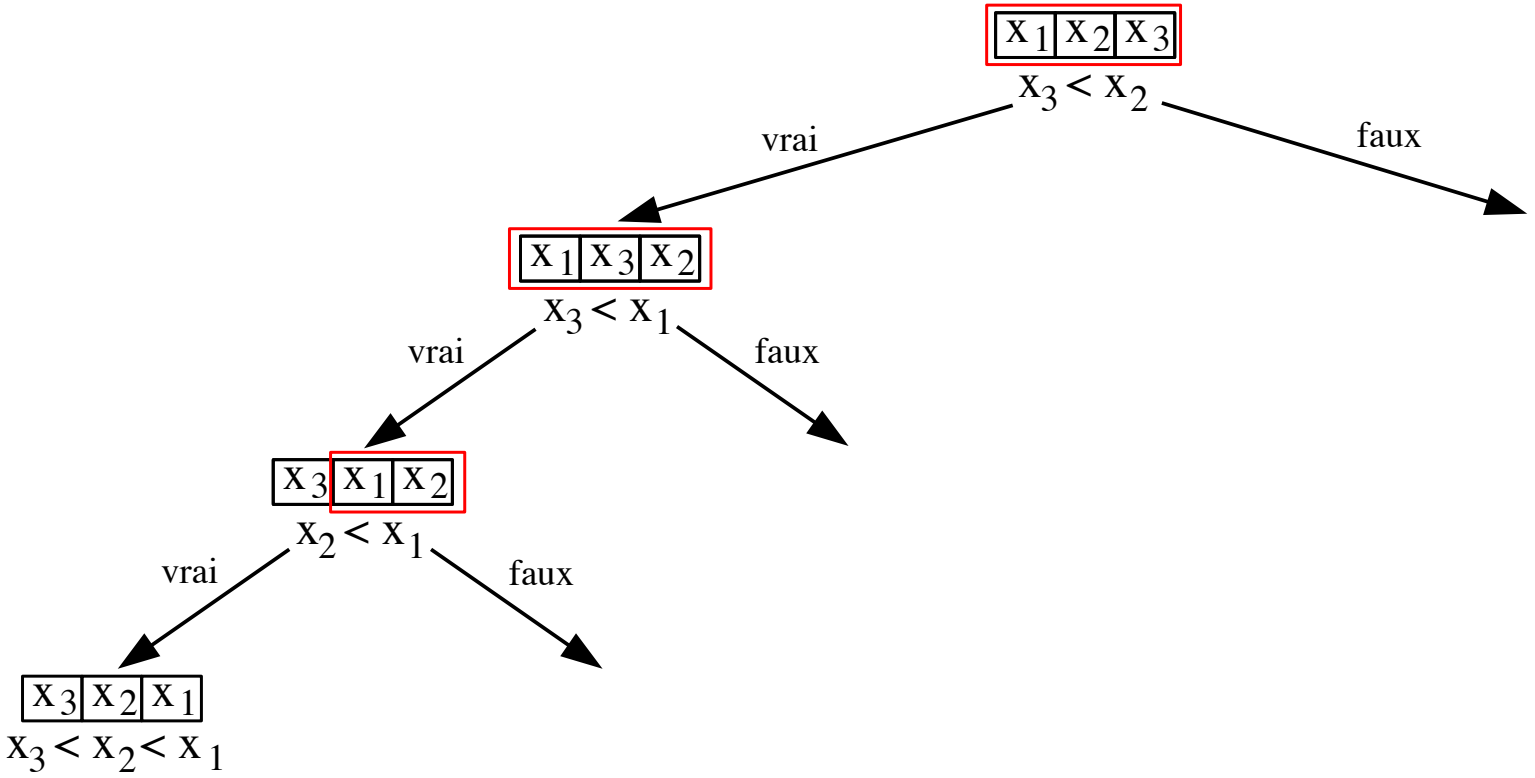


2eme étape : comparaison $x_2 < x_1$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

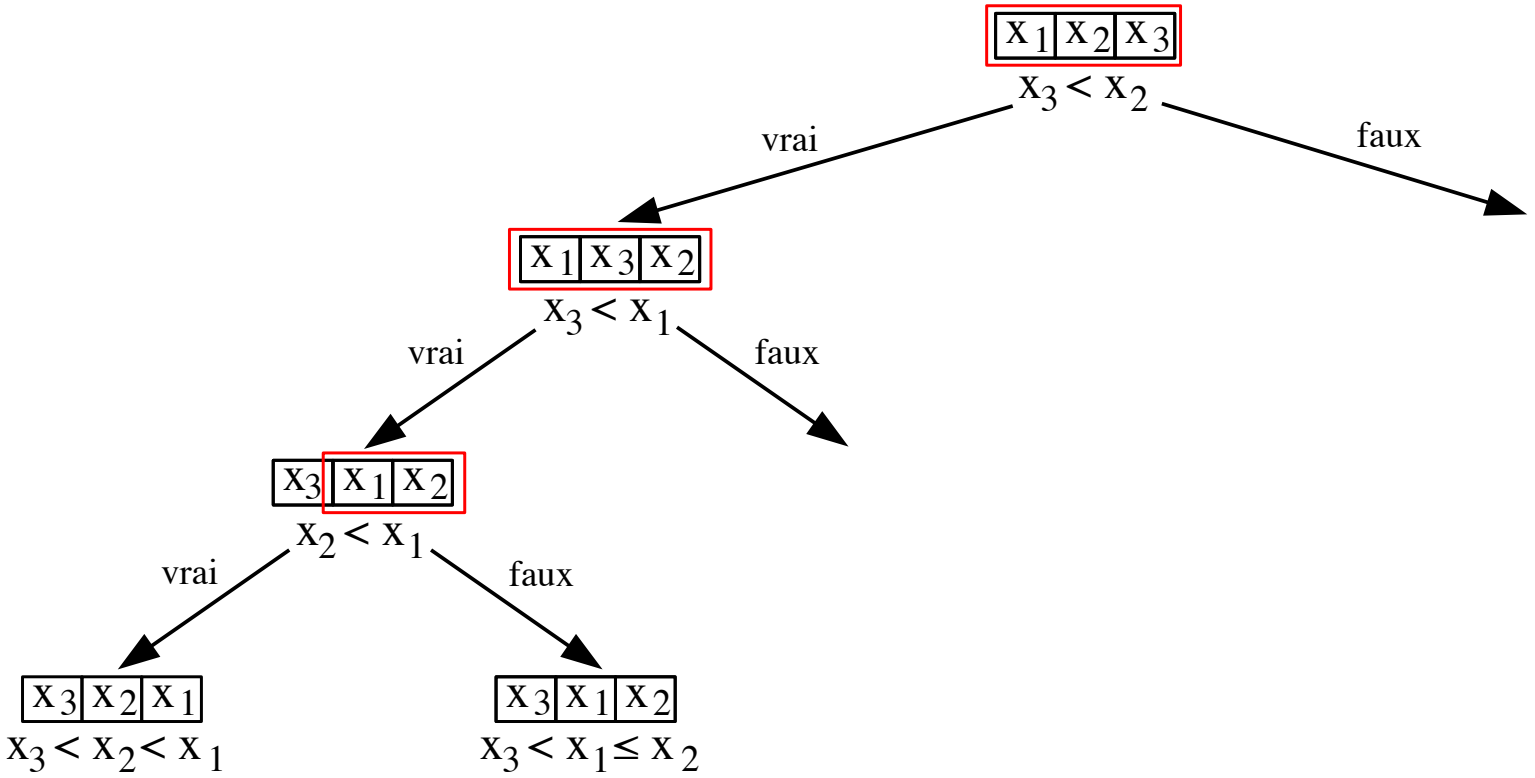


Test $x_2 < x_1$ vrai : permutation x_1 et x_2 puis arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

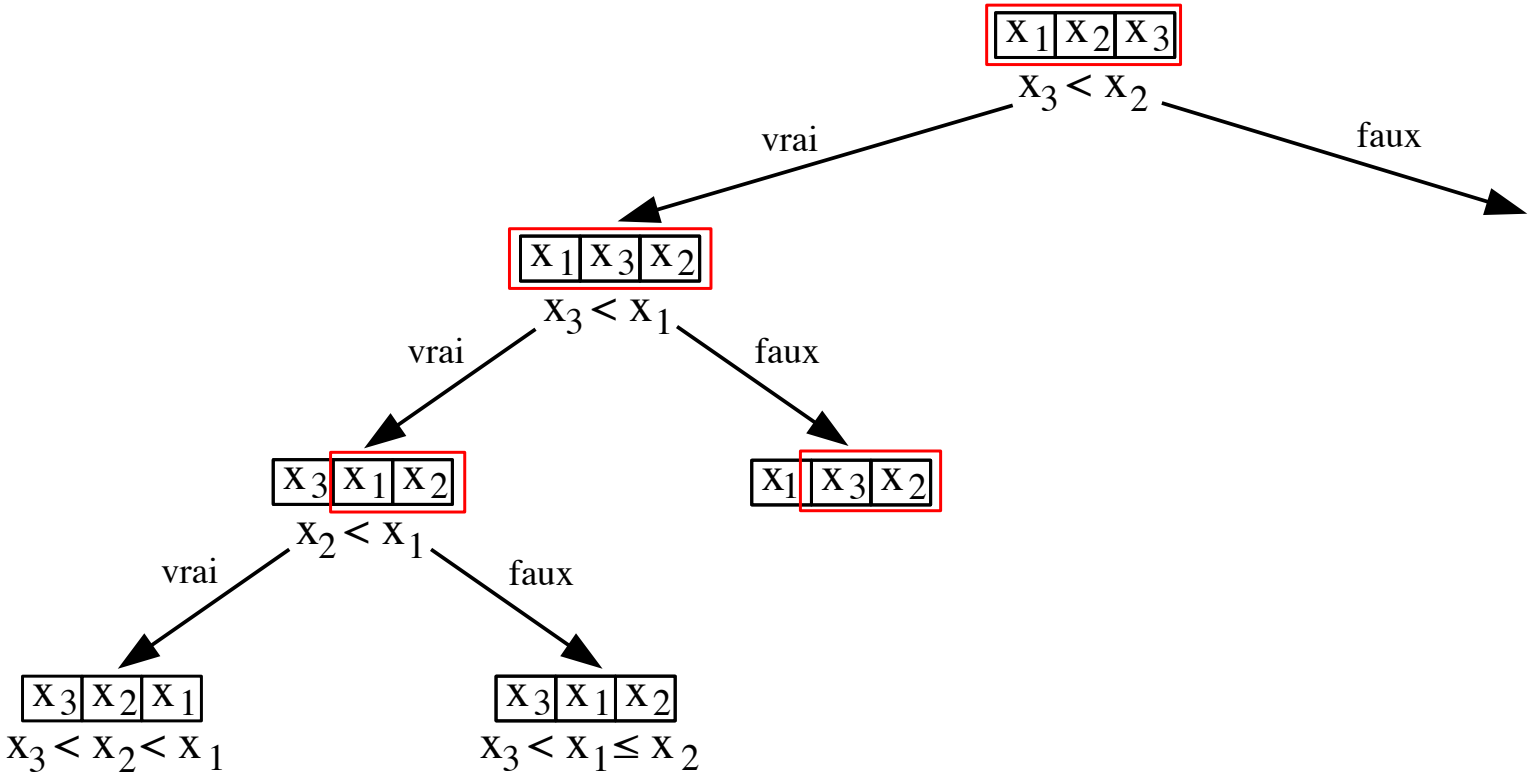


Test $x_2 < x_1$ faux : rien et arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

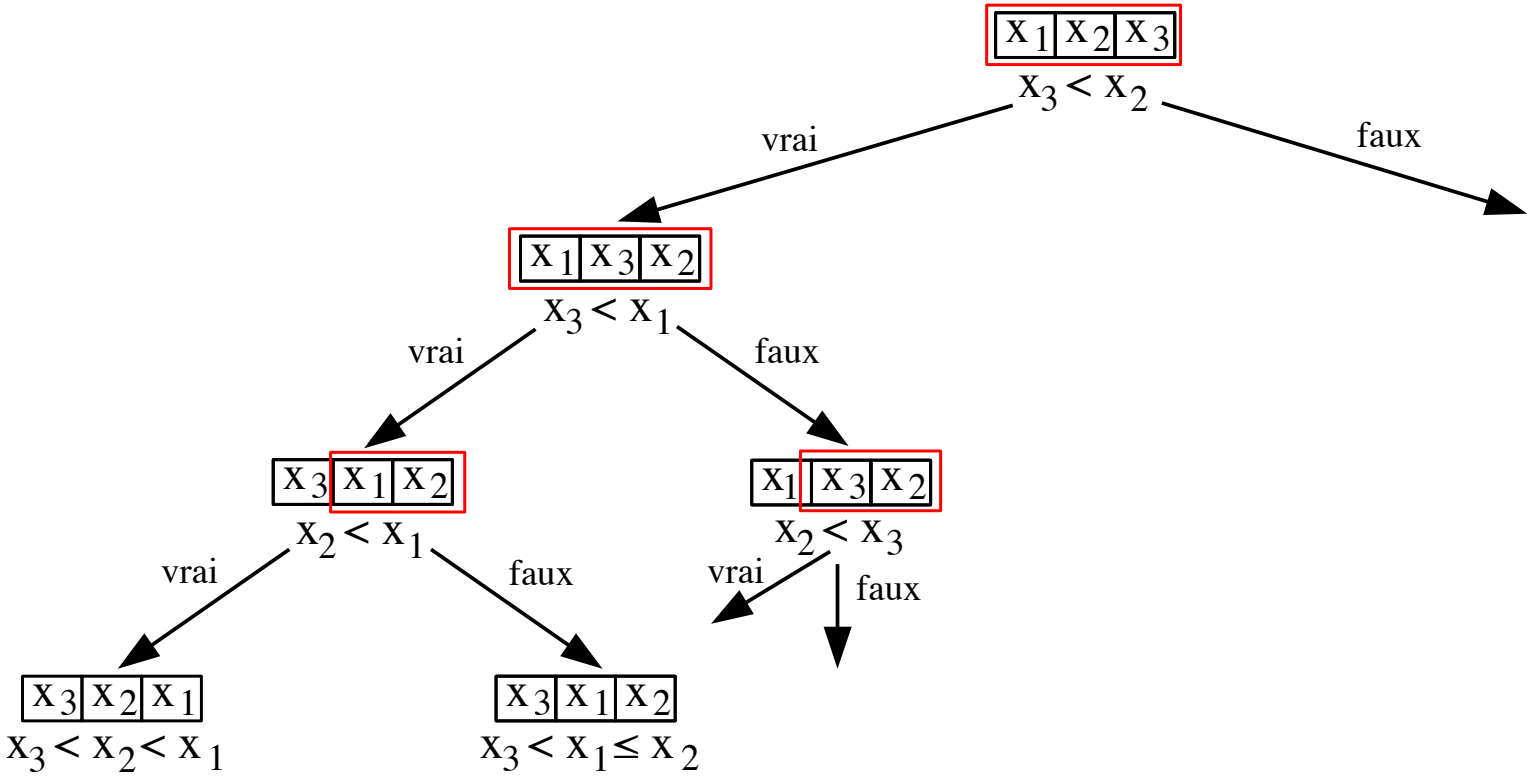


1^{ere} étape avec test $x_3 < x_1$ faux : rien et poursuite branche droite

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

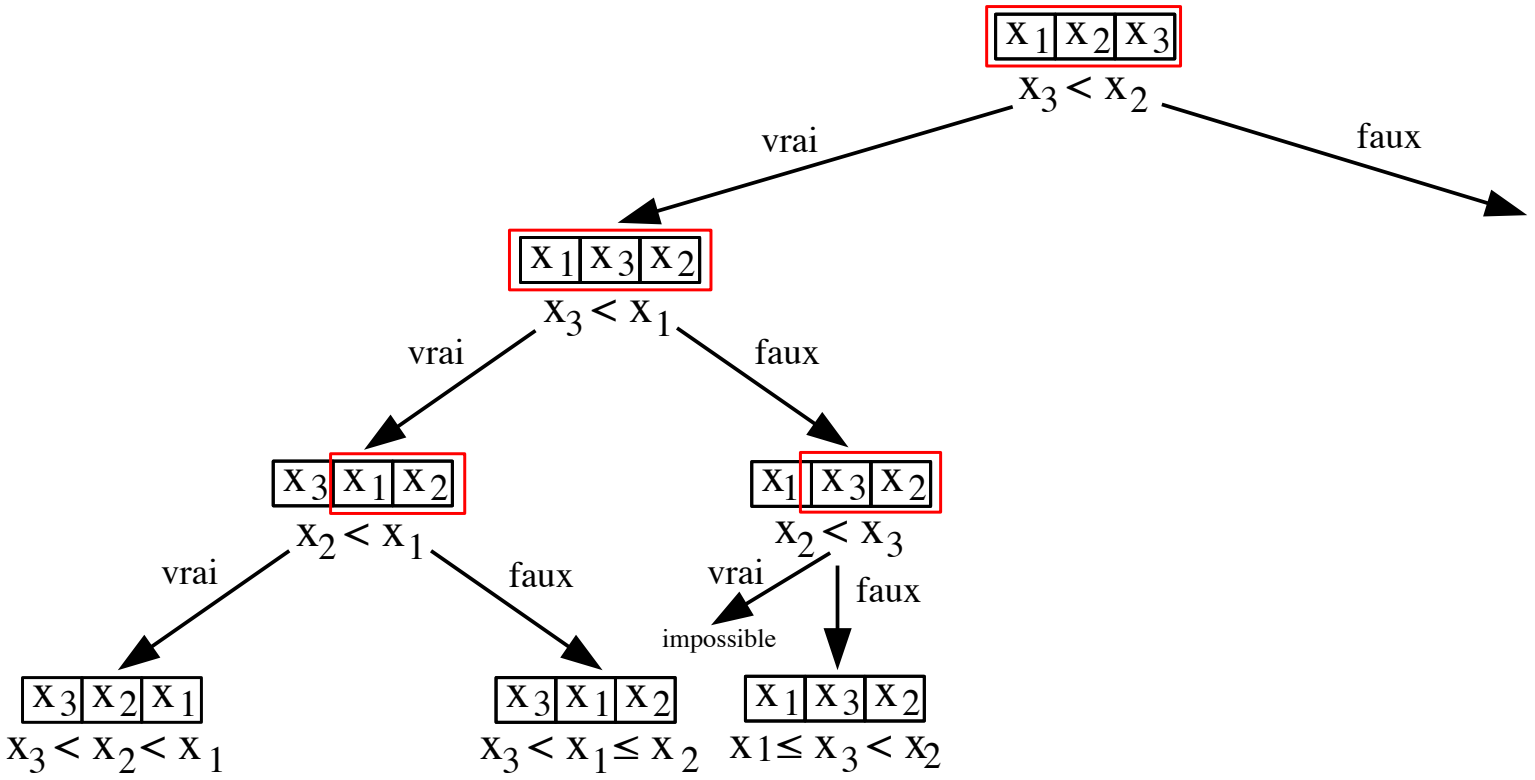


2eme étape : comparaison $x_2 < x_3$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

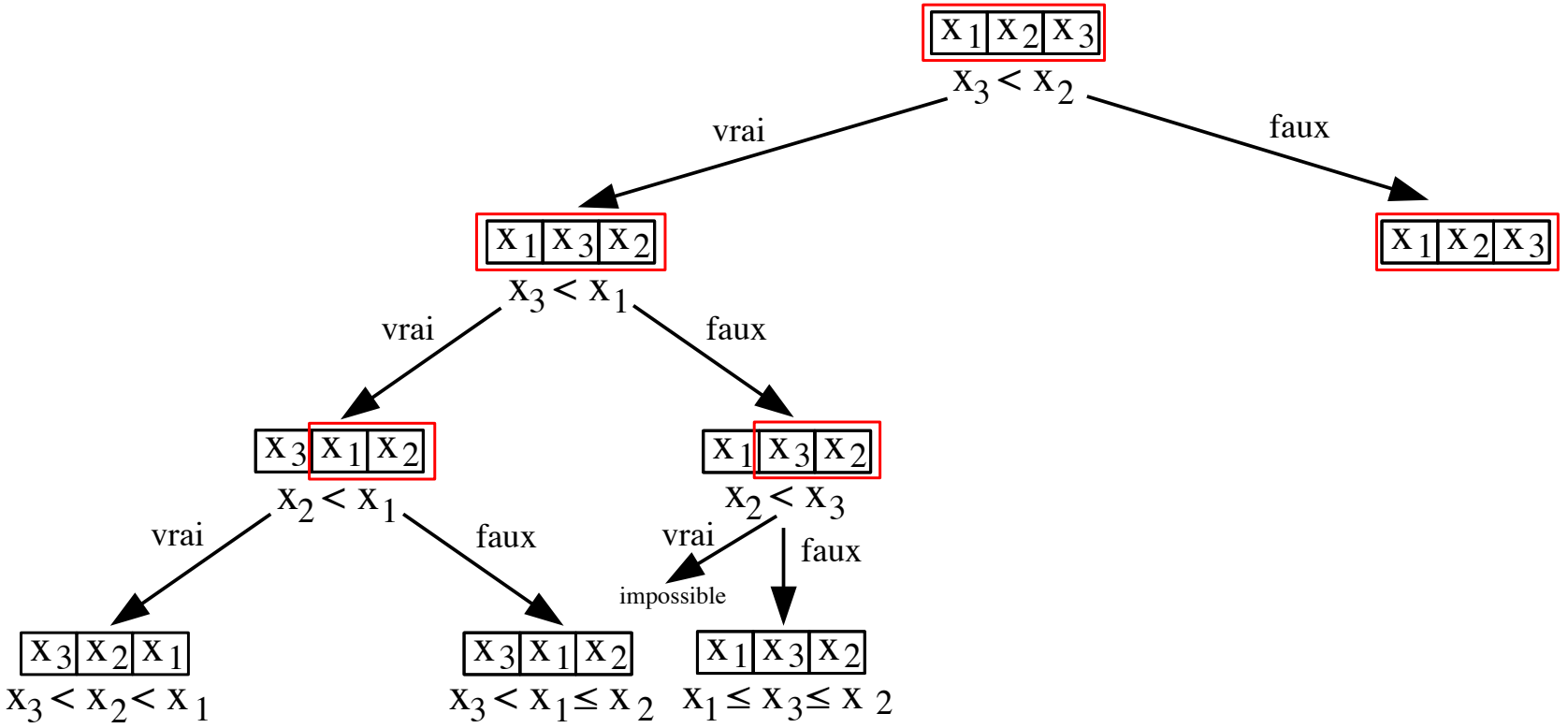


Test $x_2 < x_3$ faux : (cf. test racine) donc rien et arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

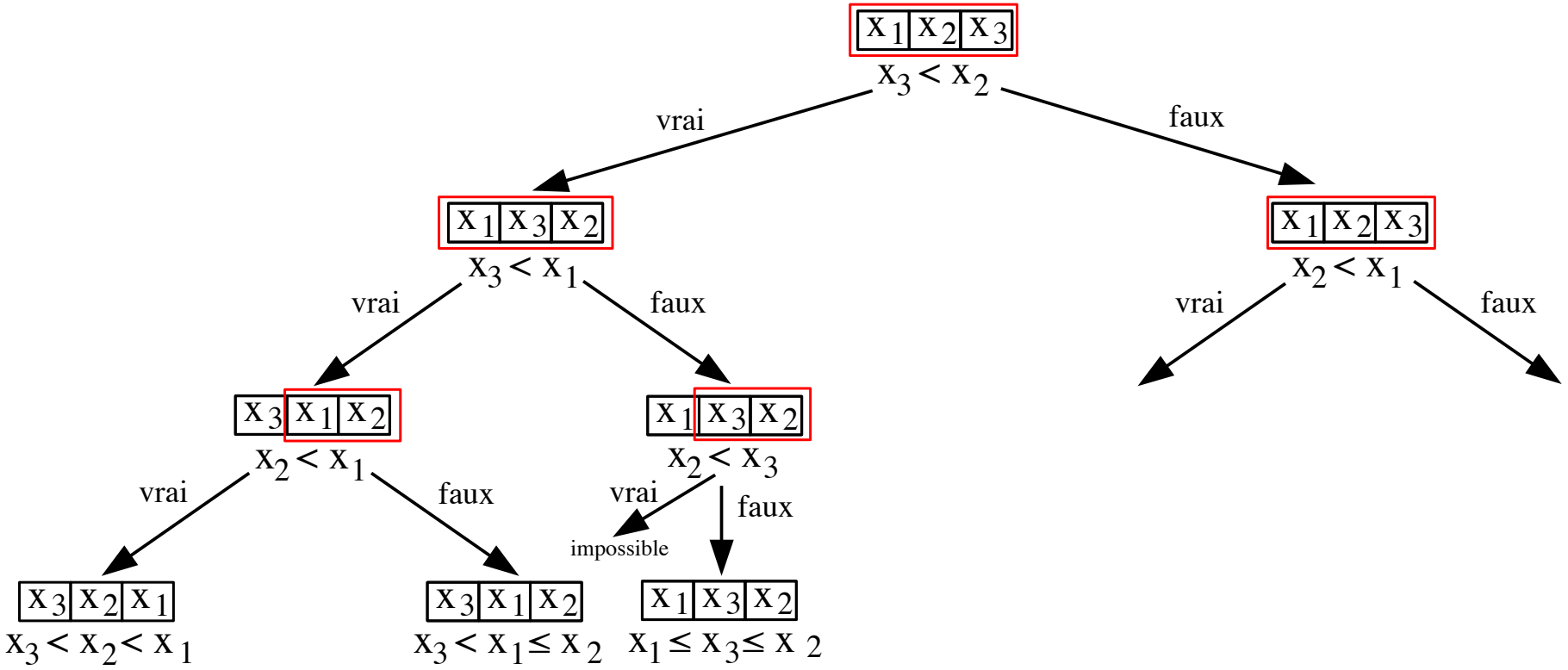


1^{ere} étape avec test $x_3 < x_2$ faux : poursuite exécution branche droite

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

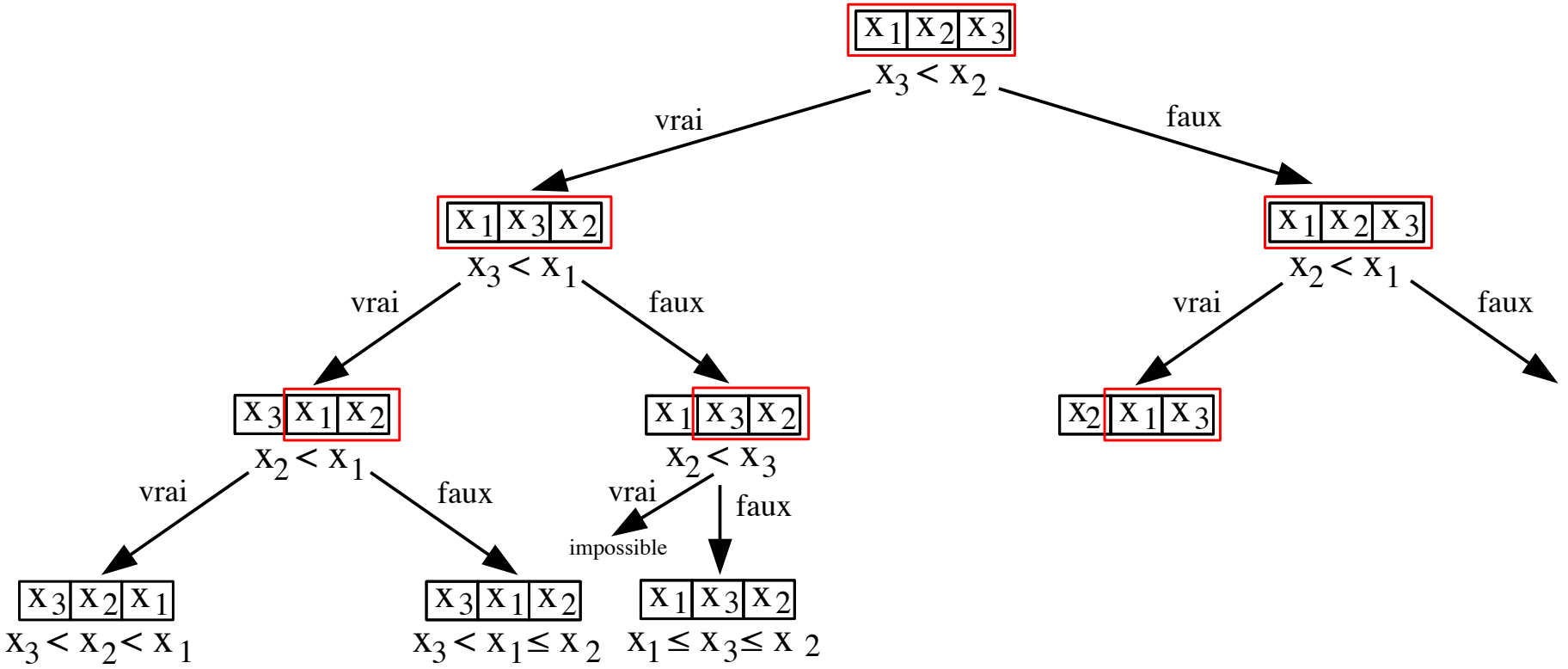


1^{ere} étape : comparaison $x_2 < x_1$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

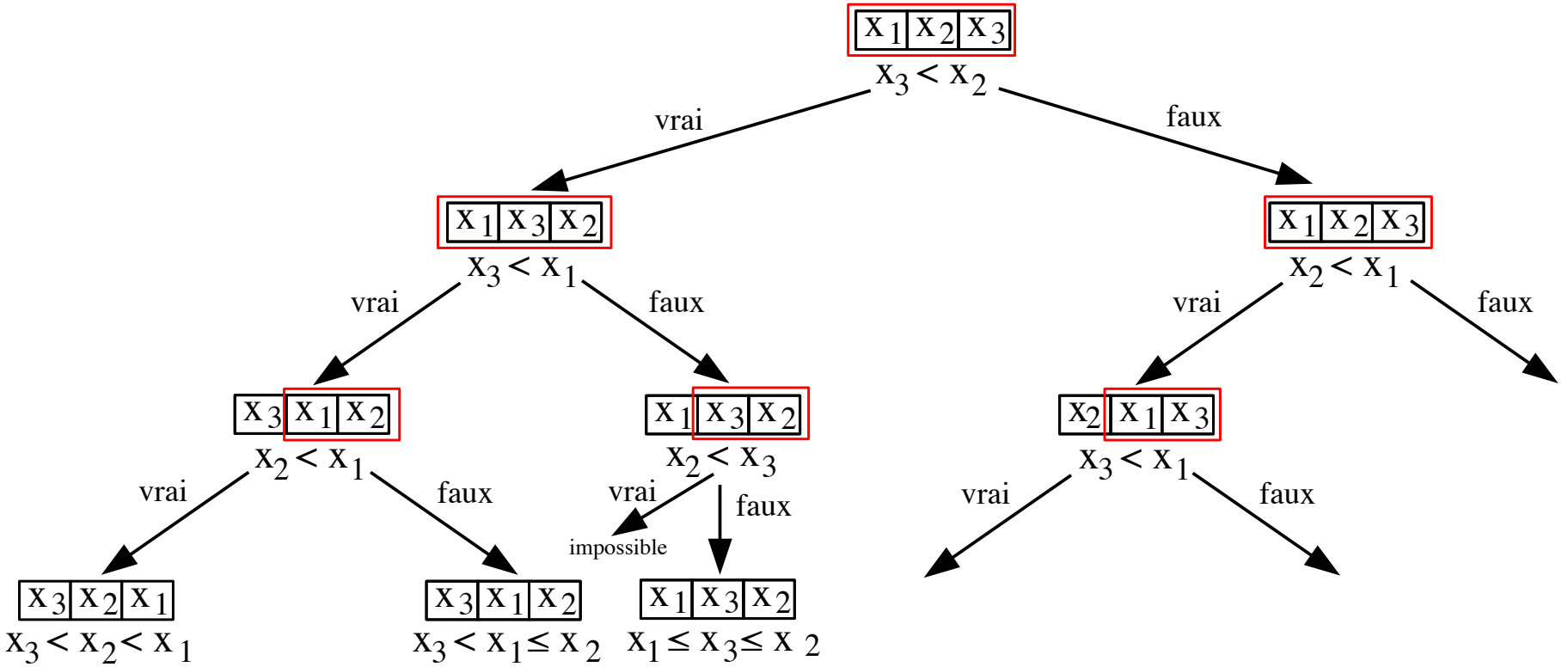


Test $x_2 < x_1$ vrai : permutation x_1 et x_2 et poursuite branche gauche

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

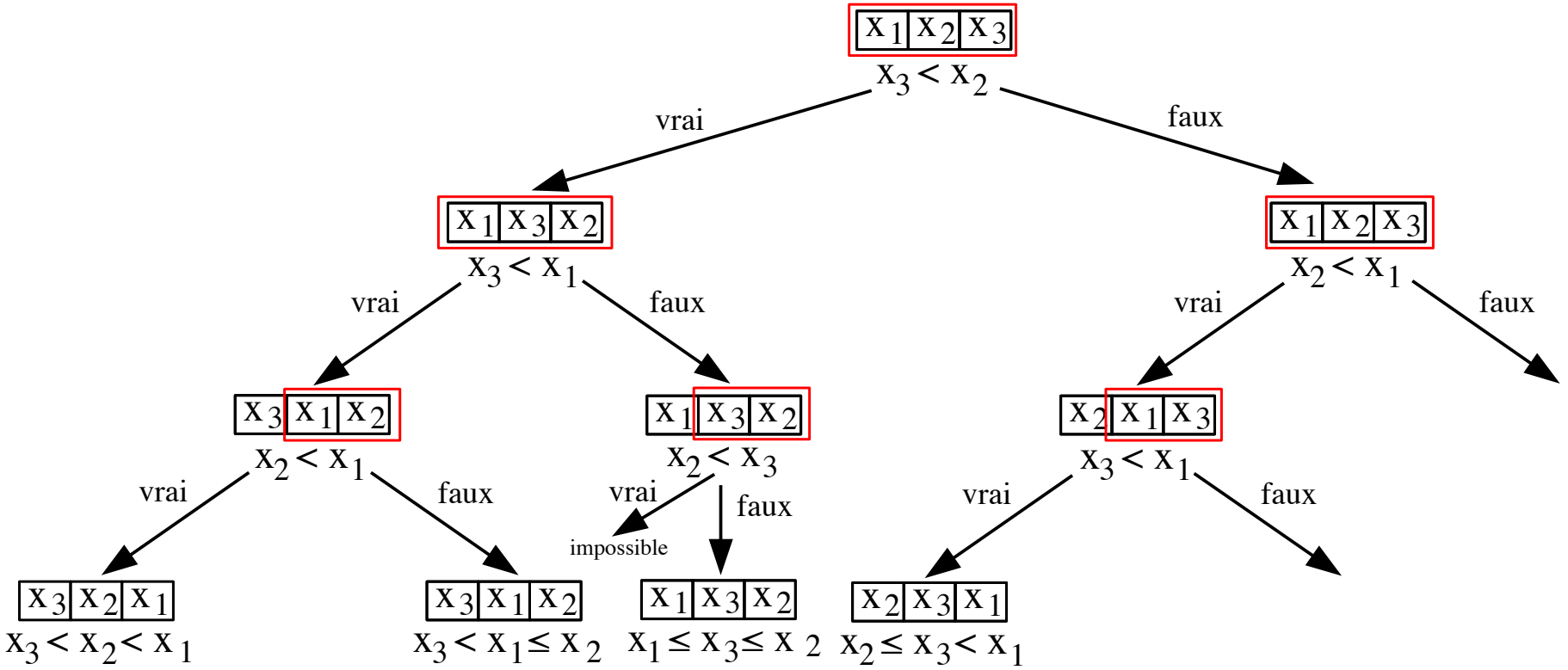


2eme étape : comparaison $x_3 < x_1$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

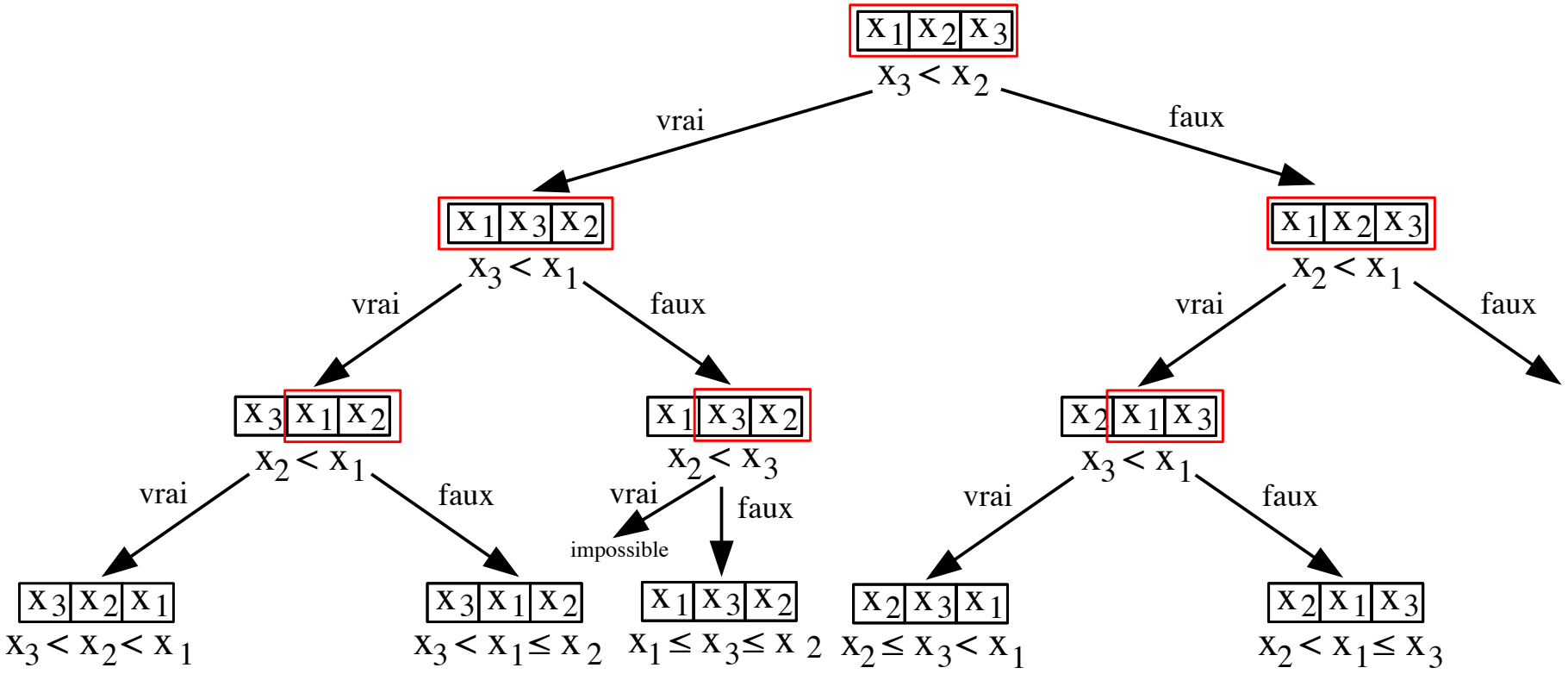


Test $x_3 < x_1$ vrai : permutation x_1 et x_3 et arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

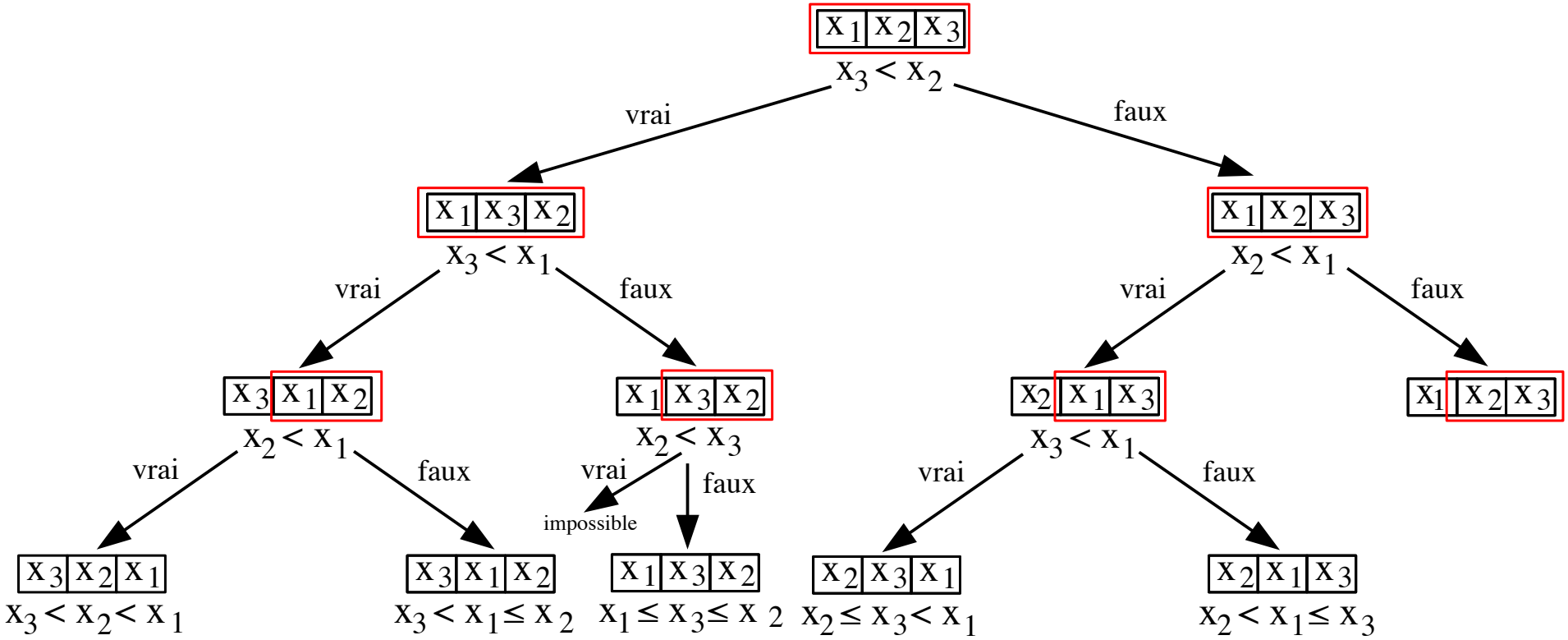


Test $x_3 < x_1$ faux : rien et arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

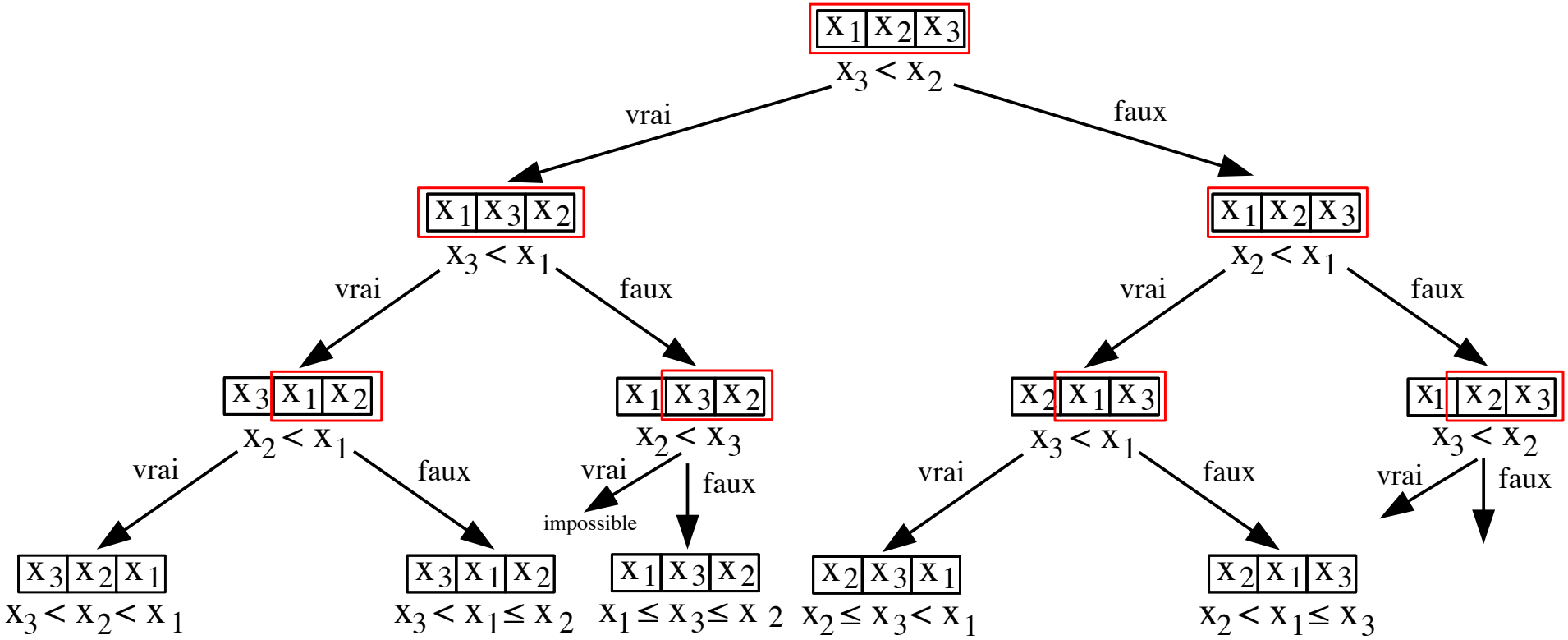


1^{ere} étape avec test $x_2 < x_1$ faux : poursuite exécution branche droite

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

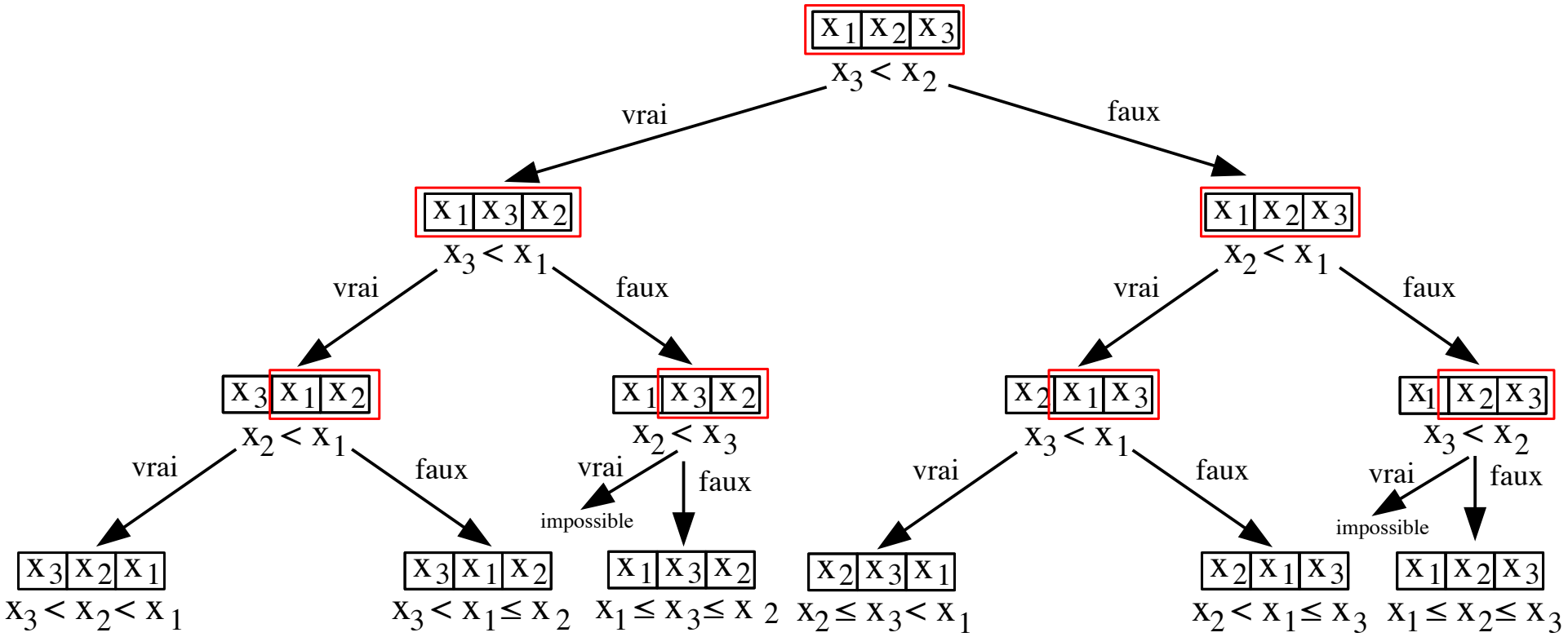


2^{eme} étape : comparaison $x_3 < x_2$ avec 2 possibilités

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles

En entrée : $[x_1, x_2, x_3]$ (on ne sait pas si $x_1 < x_2$, ni si $x_1 < x_3$, ni si $x_2 < x_3$)

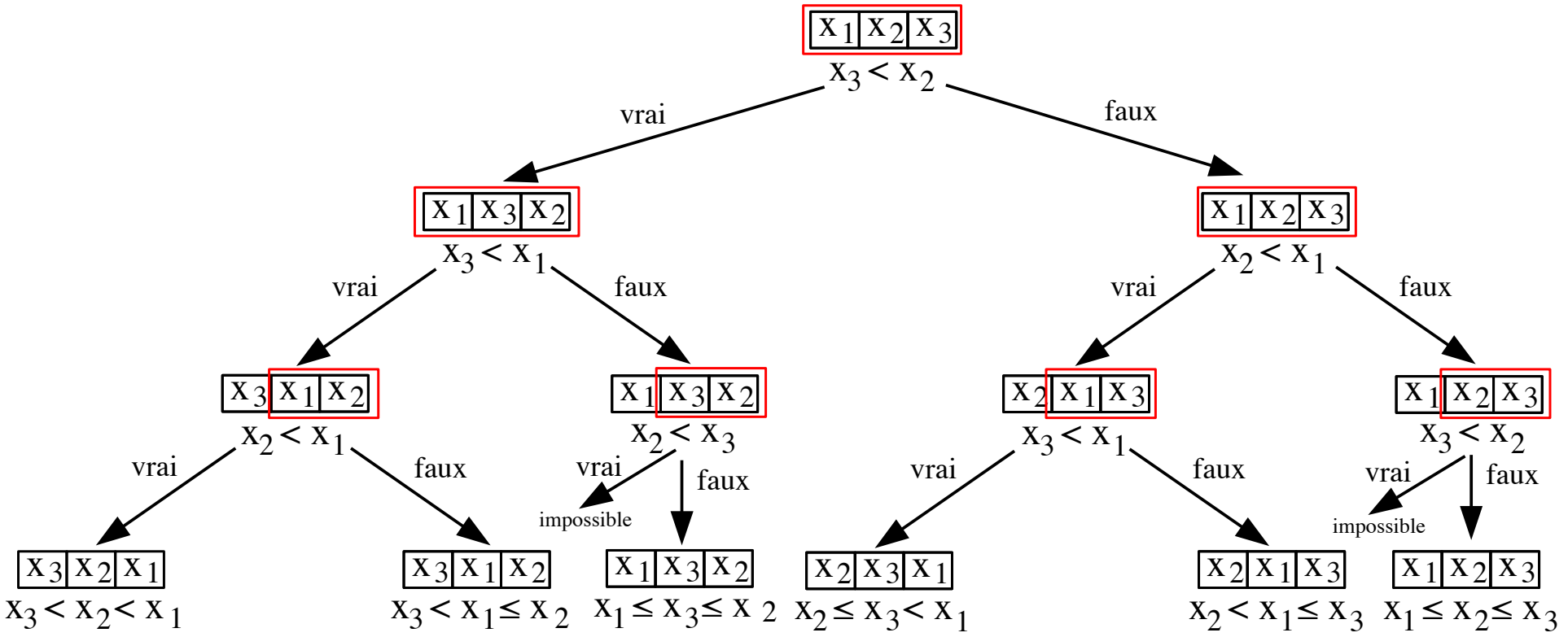


Test $x_2 < x_3$ faux : (cf. test racine) donc rien et arrêt exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles :

Toutes les exécutions possibles sont représentées

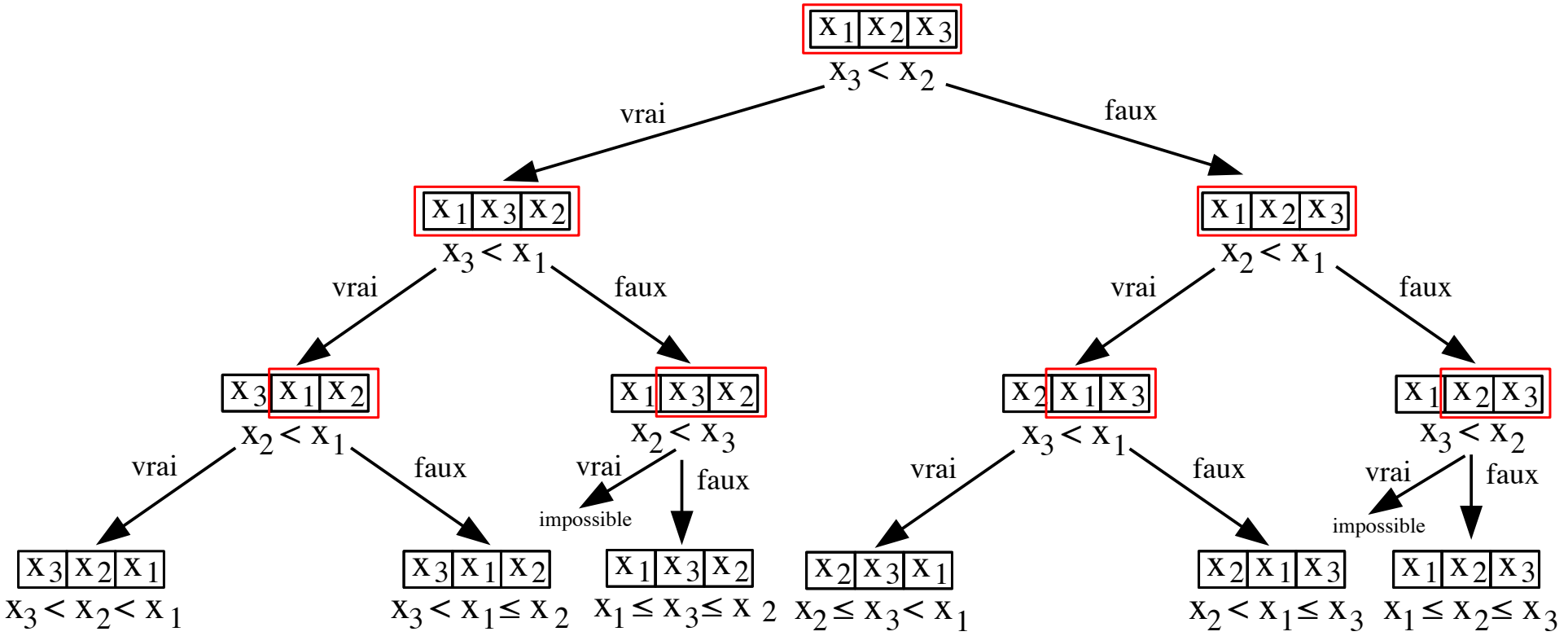


Chaque branche (jusqu'à une feuille) représente 1 exécution

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles :

Toutes les exécutions possibles sont représentées

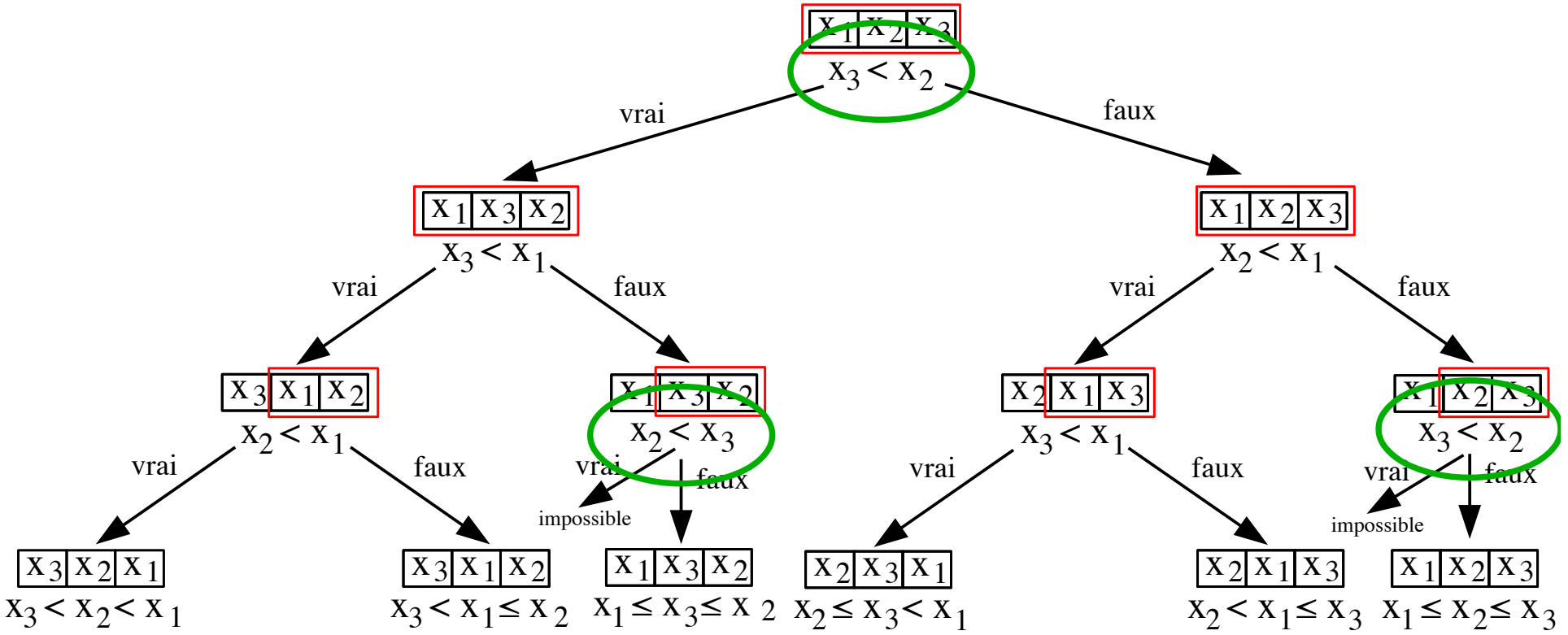


6 branches, 6 feuilles, 6 exécutions : 6 = 3! = 6 entrées possibles

Complexité d'un problème : le tri par comparaison

Arbre de décision associé au tri à bulles :

Toutes les exécutions possibles sont représentées



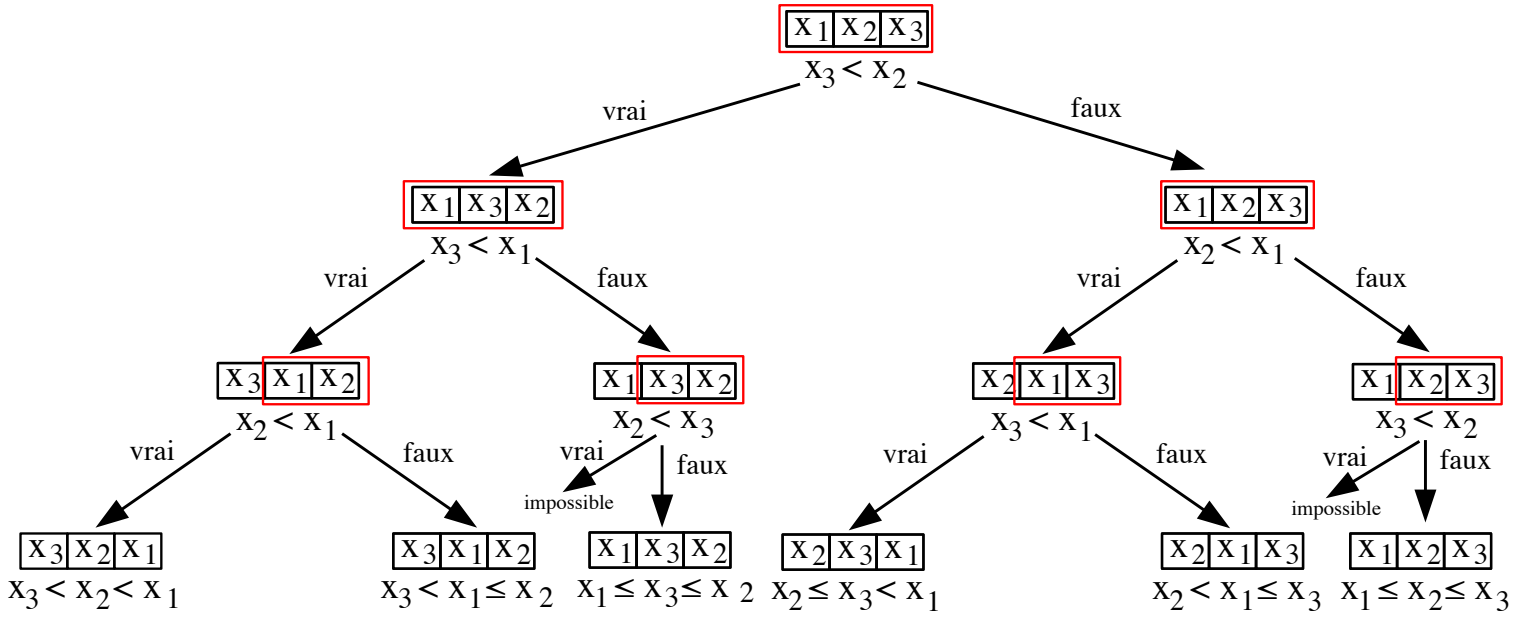
Comparaison entre x_2 et x_3 exécutée jusqu'à 2 fois : tri non optimal

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Complexité d'un problème : le tri par comparaison

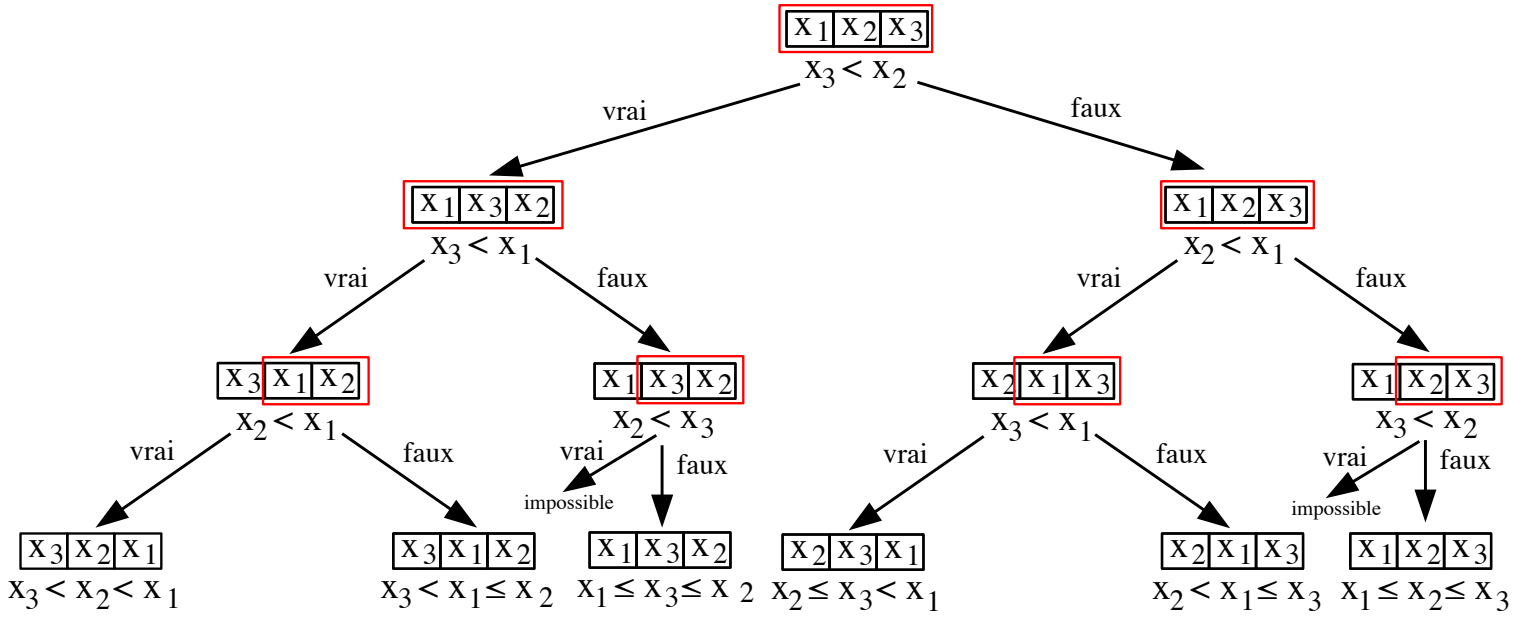
Pour borner la complexité : étude de la hauteur de ces arbres



Car le temps d'une exécution correspond à la longueur de sa branche

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres



Car le temps d'une exécution correspond à la longueur de sa branche

Pire des cas = plus longue branche

Complexité = taille plus longue branche = hauteur de l'arbre

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Lemme 1

L'arbre de décision associé à un algorithme de tri par comparaison sur n éléments possède exactement $n!$ feuilles

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Lemme 1

L'arbre de décision associé à un algorithme de tri par comparaison sur n éléments possède exactement $n!$ feuilles

Preuve :

- **Nombre maximum de feuilles :** à une feuille correspond une et une seule exécution, donc au moins un ordre initial.
  $\Rightarrow$ plus d'ordres initiaux en entrée que de feuilles

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Lemme 1

L'arbre de décision associé à un algorithme de tri par comparaison sur n éléments possède exactement $n!$ feuilles

Preuve :

- **Nombre maximum de feuilles :** à une feuille correspond une et une seule exécution, donc au moins un ordre initial.
  $\Rightarrow$ plus d'ordres initiaux en entrée que de feuilles
- **Nombre minimum de feuilles :** à un ordre initial, on ne peut associer qu'un chemin dans l'arbre, donc qu'une exécution, et donc qu'une seule feuille.
  $\Rightarrow$ plus de feuilles que d'ordres initiaux en entrée

Complexité d'un problème : le tri par comparaison

Pour borner la complexité : étude de la hauteur de ces arbres

Lemme 1

L'arbre de décision associé à un algorithme de tri par comparaison sur n éléments possède exactement $n!$ feuilles

Preuve :

- **Nombre maximum de feuilles :** à une feuille correspond une et une seule exécution, donc au moins un ordre initial.
  $\Rightarrow$ plus d'ordres initiaux en entrée que de feuilles
- **Nombre minimum de feuilles :** à un ordre initial, on ne peut associer qu'un chemin dans l'arbre, donc qu'une exécution, et donc qu'une seule feuille.
  $\Rightarrow$ plus de feuilles que d'ordres initiaux en entrée

$\Rightarrow$ **nombre de feuilles = nombre d'ordre initiaux =** $n!$

Complexité d'un problème : le tri par comparaison

Étude des propriétés des arborescences binaires (donc de ces arbres)

Lemme 2

Pour une arborescence binaire de hauteur h (h = nombre de niveaux -1)

2^h = nombre maximum de feuilles

Preuve : par induction sur h (mais vous l'avez déjà vu en licence...).

Complexité d'un problème : le tri par comparaison

Étude des propriétés des arborescences binaires (donc de ces arbres)

Lemme 2

Pour une arborescence binaire de hauteur h (h = nombre de niveaux -1)

2^h = nombre maximum de feuilles

Preuve : par induction sur h (mais vous l'avez déjà vu en licence...).

Remarque : donc, pour une hauteur h, on a au plus 2^h feuilles, d'où :

- pour toute arborescence binaire de hauteur h de f feuilles, on a $f \leq 2^h$
- et comme $f \leq 2^h \Rightarrow log_2(f) \leq log_2(2^h) = h$

Complexité d'un problème : le tri par comparaison

Étude des propriétés des arborescences binaires (donc de ces arbres)

Lemme 2

Pour une arborescence binaire de hauteur h (h = nombre de niveaux -1)

2^h = nombre maximum de feuilles

Preuve : par induction sur h (mais vous l'avez déjà vu en licence...).

Remarque : donc, pour une hauteur h, on a au plus 2^h feuilles, d'où :

- pour toute arborescence binaire de hauteur h de f feuilles, on a $f \leq 2^h$
- et comme $f \leq 2^h \Rightarrow log_2(f) \leq log_2(2^h) = h$

on peut donc en déduire la propriété suivante :

Propriété 1

La hauteur h d'une arborescence binaire de f feuilles vérifie $log_2(f) \leq h$

Complexité d'un problème : le tri par comparaison

L'arbre de décision associé à un algorithme de tri par comparaison de n éléments vérifie :

- Lemme 1 : son nombre de feuilles $f = n!$
- Propriété 1 : sa hauteur h vérifie $log_2(f) \leq h$

Complexité d'un problème : le tri par comparaison

L'arbre de décision associé à un algorithme de tri par comparaison de n éléments vérifie :

- Lemme 1 : son nombre de feuilles $f = n!$
- Propriété 1 : sa hauteur h vérifie $log_2(f) \leq h$

d'où :

Corollaire

La hauteur d'un arbre de décision pour le tri par comparaison de n éléments est minorée par $log_2(n!)$

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Rappel de la preuve de $log_2(n!) \in \Theta(n.log(n))$:

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Rappel de la preuve de $log_2(n!) \in \Theta(n.log(n))$:

On sait que $log_2(n!) = log_2(1) + log_2(2) + \cdots + log_2(n) = \Sigma_{i=1}^{i=n} log_2(i)$

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Rappel de la preuve de $log_2(n!) \in \Theta(n.log(n))$:

On sait que $log_2(n!) = log_2(1) + log_2(2) + \cdots + log_2(n) = \Sigma_{i=1}^{i=n} log_2(i)$

Majoration : $\Sigma_{i=1}^{i=n} log_2(i) \leq n.log_2(n)$ car $\forall i, 1 \leq i \leq n$, on a $log_2(i) \leq log_2(n)$

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Rappel de la preuve de $log_2(n!) \in \Theta(n.log(n))$:

On sait que $log_2(n!) = log_2(1) + log_2(2) + \cdots + log_2(n) = \Sigma_{i=1}^{i=n} log_2(i)$

Majoration : $\Sigma_{i=1}^{i=n} log_2(i) \leq n.log_2(n)$ car $\forall i, 1 \leq i \leq n$, on a $log_2(i) \leq log_2(n)$

Minoration : $\Sigma_{i=1}^{i=n} log_2(i) \geq \Sigma_{i=\lfloor n/2 \rfloor}^{i=n} log_2(i) \geq \frac{n}{2}.log_2(\frac{n}{2}) = \frac{n}{2}.(log_2(n) - 1)$

Complexité d'un problème : le tri par comparaison

Théorème

La complexité du tri par comparaison est $\Theta(n.log(n))$

Preuve :

- Le nombre de tests est minoré par $log_2(n!) \in \Theta(n.log(n))$ donc aucun tri n'exécute moins de tests
- La complexité du tri par fusion est $\Theta(n.log(n))$

La complexité du tri par comparaison est minorée par $\Theta(n.log(n))$ et il existe un algorithme de même complexité.

Rappel de la preuve de $log_2(n!) \in \Theta(n.log(n))$:

On sait que $log_2(n!) = log_2(1) + log_2(2) + \cdots + log_2(n) = \Sigma_{i=1}^{i=n} log_2(i)$

Majoration : $\Sigma_{i=1}^{i=n} log_2(i) \leq n.log_2(n)$ car $\forall i, 1 \leq i \leq n$, on a $log_2(i) \leq log_2(n)$

Minoration : $\Sigma_{i=1}^{i=n} log_2(i) \geq \Sigma_{i=\lfloor n/2 \rfloor}^{i=n} log_2(i) \geq \frac{n}{2}.log_2(\frac{n}{2}) = \frac{n}{2}.(log_2(n) - 1)$

donc on a : $\frac{n}{2}.(log_2(n) - 1) \leq log_2(n!) \leq n.log_2(n)$ *et donc* $log_2(n!) \in \Theta(n.log(n))$

Plan

Problème polynomial

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Problème polynomial

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Définition

Un problème π est dit **polynomial** pour le modèle de calcul associé à la classe $\mathcal{C}$ des algorithmes qui résolvent ce problème, s'il existe un algorithme de résolution de complexité polynomiale, i.e. :

$$\exists A \in \mathcal{C} \text{ tel que } \max_{I \in D_\pi^n} T_A(I) \in O(p(n)) \text{ où } p \text{ est un polynôme.}$$

Problème polynomial

On considère ici :

- Un problème π (décision, recherche, optimisation,...)
- L'ensemble ou classe $\mathcal{C}$ des algorithmes qui résolvent le problème π (dans un modèle de calcul donné)

Définition

Un problème π est dit **polynomial** pour le modèle de calcul associé à la classe $\mathcal{C}$ des algorithmes qui résolvent ce problème, s'il existe un algorithme de résolution de complexité polynomiale, i.e. :

$\exists A \in \mathcal{C}$ tel que $\max_{I \in D_\pi^n} T_A(I) \in O(p(n))$ où p est un polynôme.

Remarques :

- Présence d'une valeur dans un tableau ? En $O(n)$ donc polynomial
- Tri d'un tableau ? Polynomial déjà avec le *tri à bulles* car en $O(n^2)$

si on considère les modèles de calcul usuels déterministes (ex. langage C)

Classes de Complexité

Définition

Une **classe de complexité** est un ensemble de problèmes d'un certain type (ex. problèmes de décision) que l'on regroupe en fonction de leur complexité pour un modele de calcul donné (ex. Machines de Turing Déterministes).

Classes de Complexité

Définition

Une **classe de complexité** est un ensemble de problèmes d'un certain type (ex. problèmes de décision) que l'on regroupe en fonction de leur complexité pour un modele de calcul donné (ex. Machines de Turing Déterministes).

Définition

La **classe des problèmes polynomiaux** qui est notée **P** ou bien **PTIME**, est l'ensemble des problèmes de décision polynomiaux sur la classe des algorithmes définis sur des modèles de calcul déterministes (Assembleur, Python, Java, Pascal, C, Machines de Von Neumann, Machines de Turing Déterministes,...).

Classes de Complexité

Définition

Une **classe de complexité** est un ensemble de problèmes d'un certain type (ex. problèmes de décision) que l'on regroupe en fonction de leur complexité pour un modele de calcul donné (ex. Machines de Turing Déterministes).

Définition

La **classe des problèmes polynomiaux** qui est notée **P** ou bien **PTIME**, est l'ensemble des problèmes de décision polynomiaux sur la classe des algorithmes définis sur des modèles de calcul déterministes (Assembleur, Python, Java, Pascal, C, Machines de Von Neumann, Machines de Turing Déterministes,...).

Attention : pour les langages de programmation (Assembleur, Python, C, etc.), il faut faire l'hypothèse d'une exécution sur une machine à mémoire de taille non bornée...

Classes de Complexité

Définition

Un problème π est dit **exponentiel** s'il existe un algorithme de résolution de complexité exponentielle qui le résout et s'il n'existe pas d'algorithme de résolution de complexité inférieure à une complexité exponentielle, i.e. en supposant que $\mathcal{C}$ soit la classe d'algorithmes qui résolvent ce problème :

$$\exists c, k \in \mathbb{R}^{*+}, k > 1, \min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I) \geq c.k^n$$

Classes de Complexité

Définition

Un problème π est dit **exponentiel** s'il existe un algorithme de résolution de complexité exponentielle qui le résout et s'il n'existe pas d'algorithme de résolution de complexité inférieure à une complexité exponentielle, i.e. en supposant que $\mathcal{C}$ soit la classe d'algorithmes qui résolvent ce problème :

$$\exists c, k \in \mathbb{R}^{*+}, k > 1, \min_{A \in \mathcal{C}} \max_{I \in D_\pi^n} T_A(I) \geq c.k^n$$

Définition

La **classe des problèmes exponentiels** notée **EXPTIME** est l'ensemble des problèmes de décision qui peuvent être résolus en temps exponentiel par des algorithmes définis sur des modèles de calcul déterministes.

Modèles de calcul déterministes : Assembleur, Python, Java, Pascal, C, Machines de Von Neumann, Machines de Turing Déterministes,...

Attention : la classe des problèmes exponentiels contient les problèmes polynomiaux

Classes de Complexité : Quelques remarques

- **Notion de classes de problèmes :** permet de procéder à des comparaisons entre problèmes au niveau de leur difficulté par comparaisons entre classes : **PTIME** $\subseteq$ **EXPTIME**

  (tout problème résoluble en temps polynomial est résoluble en temps exponentiel)

Classes de Complexité : Quelques remarques

- **Notion de classes de problèmes :** permet de procéder à des comparaisons entre problèmes au niveau de leur difficulté par comparaisons entre classes : **PTIME** $\subseteq$ **EXPTIME**

  (tout problème résoluble en temps polynomial est résoluble en temps exponentiel)

- **Complexité en espace** : ressources en espace mémoire nécessaires pour la résolution d'un problème ; on peut définir des classes de complexité :

Classes de Complexité : Quelques remarques

- **Notion de classes de problèmes :** permet de procéder à des comparaisons entre problèmes au niveau de leur difficulté par comparaisons entre classes : **PTIME** $\subseteq$ **EXPTIME**

  (tout problème résoluble en temps polynomial est résoluble en temps exponentiel)

- **Complexité en espace** : ressources en espace mémoire nécessaires pour la résolution d'un problème ; on peut définir des classes de complexité :

Définition

La **classe des problèmes polynomiaux en espace**, notée **PSPACE**, est l'ensemble des problèmes de décision pour lesquels il existe un algorithme de résolution défini sur des modèles de calcul déterministes (Assembleur, Python, Java, Pascal, C, Machines de Von Neumann, Machines de Turing Déterministes,...) qui ne requiert qu'un espace de taille polynomiale pour les résoudre.

Classes de Complexité : Quelques remarques

- **Notion de classes de problèmes :** permet de procéder à des comparaisons entre problèmes au niveau de leur difficulté par comparaisons entre classes : **PTIME** $\subseteq$ **EXPTIME**
  (tout problème résoluble en temps polynomial est résoluble en temps exponentiel)

- **Complexité en espace** : ressources en espace mémoire nécessaires pour la résolution d'un problème ; on peut définir des classes de complexité :

Définition

La **classe des problèmes polynomiaux en espace**, notée **PSPACE**, est l'ensemble des problèmes de décision pour lesquels il existe un algorithme de résolution défini sur des modèles de calcul déterministes (Assembleur, Python, Java, Pascal, C, Machines de Von Neumann, Machines de Turing Déterministes,...) qui ne requiert qu'un espace de taille polynomiale pour les résoudre.

et il a été démontré que **PTIME** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME**

Plan

Quelques remarques (parfois historiques)

Un peu d'histoire :

- Notion d'**algorithme** :
  très ancienne car datant de l'antiquité (ex. algorithme d'Euclide)
- Notion de **programme** :
  ancienne car 19ème siècle (cf. Ada Lovelace avec la machine de Babbage)
- Aspects formels :
  introduits dans les années 1930 par entre autres Turing, Church, Kleene et Post (travaux sur la calculabilité)

Quelques remarques (parfois historiques)

Un peu d'histoire :

- Notion d'**algorithme** :
  très ancienne car datant de l'antiquité (ex. algorithme d'Euclide)
- Notion de **programme** :
  ancienne car 19ème siècle (cf. Ada Lovelace avec la machine de Babbage)
- Aspects formels :
  introduits dans les années 1930 par entre autres Turing, Church, Kleene et Post (travaux sur la calculabilité)
- Notion de **complexité des problèmes** : essort dans les années 1960
  - notion de problème "facile" (polynomial) ou classe **P** introduite par Cobham (1964) et Edmonds (1965)
  - classe **NP** en 1965 par Edmonds en termes de certificats polynomiaux (le temps de vérification d'une solution est polynomial).

Quelques remarques (parfois historiques)

La suite de l'histoire sur la complexité des problèmes

- Classe **NP** :
  - définition par certificats polynomiaux mais aussi :
  - problèmes polynomiaux sur modèles de calcul non-déterministes

    par exemple, les Machines de Turing non-déterministes

    (non-déterminisme : assimilable à un modèle de calcul pour lequel le nombre de processeurs ne serait pas borné i.e. "parallélisme non borné").

Quelques remarques (parfois historiques)

La suite de l'histoire sur la complexité des problèmes

- Classe **NP** :
  - définition par certificats polynomiaux mais aussi :
  - problèmes polynomiaux sur modèles de calcul non-déterministes

    par exemple, les Machines de Turing non-déterministes

    (non-déterminisme : assimilable à un modèle de calcul pour lequel le nombre de processeurs ne serait pas borné i.e. "parallélisme non borné").
- Classe **NP-C** : les problèmes NP-Complets
  - problèmes "les plus difficiles" de **NP**
  - un résultat fondamental : Cook en 1971 (et Levin au même moment) montrent qu'il en existe :

    Un théorème central (Cook-Levin) : "SAT est NP-Complet"
  - un autre résultat fondamental : Karp en 1972 montre que de nombreux problèmes combinatoires sont équivalents en termes de complexité (notion de *transformation* ou *réduction polynomiale* entre problèmes)

Quelques remarques (parfois historiques)

La suite de l'histoire : il reste beaucoup à écrire !

Dont la réponse à la fameuse question à 1 million de $:

$$\text{Est-ce que } \mathbf{P} \neq \mathbf{NP} \text{ ?}$$

À la fin de l'UE, vous serez en mesure de comprendre cette conjecture.

Quelques remarques (parfois historiques)

La suite de l'histoire : il reste beaucoup à écrire !

Dont la réponse à la fameuse question à 1 million de $:

$$\text{Est-ce que } \mathbf{P} \neq \mathbf{NP} \text{ ?}$$

À la fin de l'UE, vous serez en mesure de comprendre cette conjecture.

Mais avant, regard sur la suite :

- Chapitre 4 : La cadre formel (i.e. théorique)...
  ou comment parler rigoureusement de ces questions
- Chapitre 5 : La classe **P**
- Chapitre 6 et les autres : La classe **NP**, etc. par Enrico Porreca