

# Algorithmique et structures de données

## Cours INF TC1

Romain Vuillemot  
`romain.vuillemot@ec-lyon.fr`

Département Math-Info  
École Centrale de Lyon

2022-2023

## Sommaire

## Séances de cours de INF TC1

- ▶ Cours 1: introduction aux algorithmiques et structures de données  
*Hash maps, Piles, Files, Listes de priorité, Listes chaînées*
- ▶ Cours 2: stratégies de programmation  
*Arbres, Parcours Profondeur/Largeur, Diviser pour Reigner, Programmation Dynamique, Glouton*
- ▶ Cours 3: structure de données avancées  
*Graphes, Arbres couvrant, Parcours de Graphe, Recherche de Chemin*
- ▶ Cours 4: algorithmes avancés  
*Arbres de Recherche, Tri, Heuristiques*



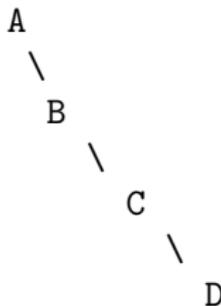
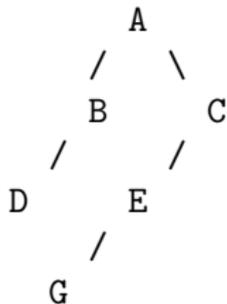
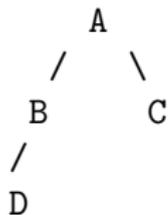
## Arbres binaires de recherche

## Concepts avancés

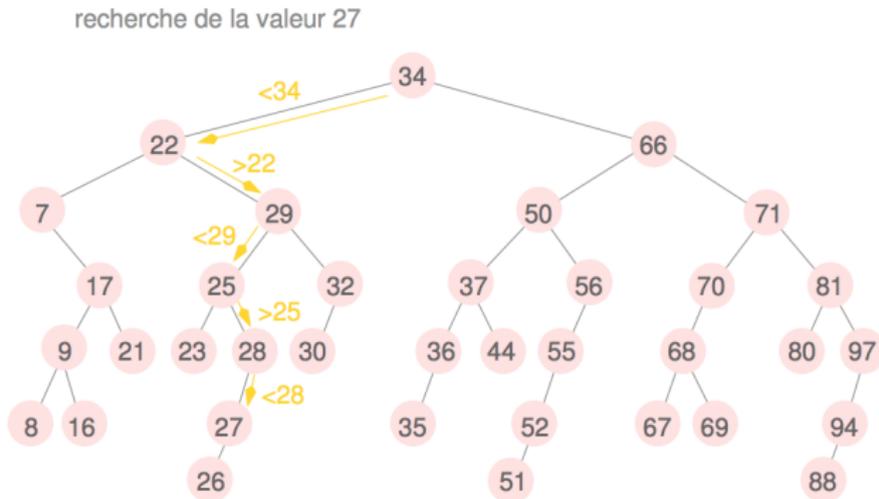
# Arbres de Recherche

Un **arbre binaire de recherche** est une structure de données composée de noeuds où chaque noeud a au plus 2 enfants ordonnés .

Très utile pour chercher des valeurs sans avoir à parcourir toute la liste (à condition d'avoir une relation d'ordre interne à l'arbre). Mais peuvent être déséquilibrés

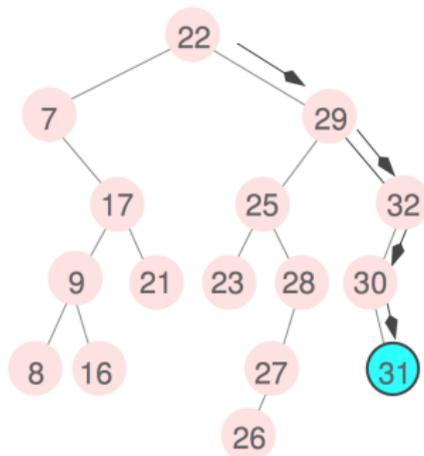
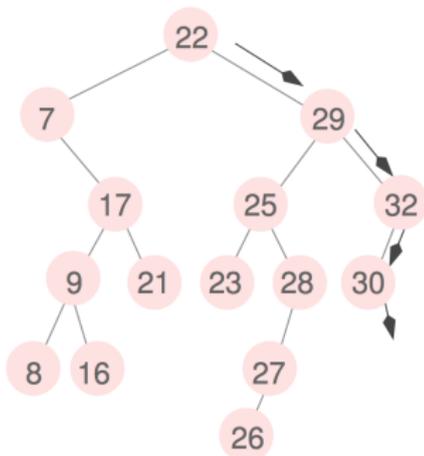


**Exemple (Abres binaires de recherche) :** la *recherche* d'une valeur consiste à parcourir une branche en partant de la racine, en descendant chaque fois sur le fils gauche ou sur le fils droit suivant si respectivement plus grande ou plus petite que la valeur cherchée.

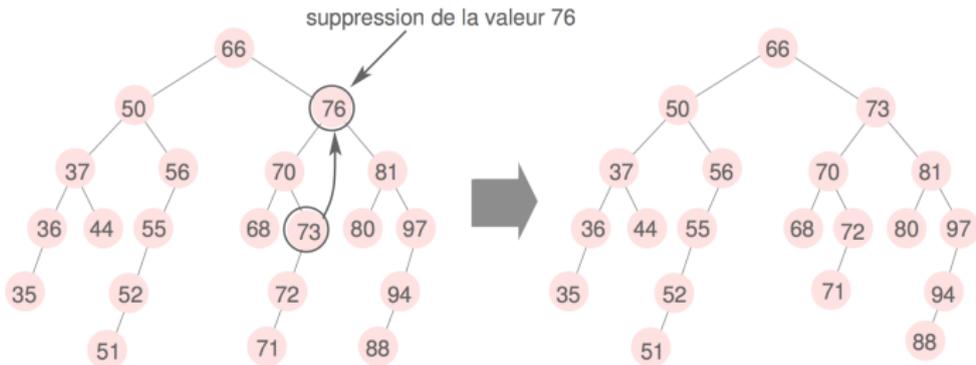


## Concepts avancés Recherche

**Exemple (Arbres binaires de recherche) :** l'insertion d'une valeur est identique que pour la recherche et s'insère la ou on s'est arrêté dans le recherche.



**Exemple (Arbres binaires de recherche) :** la *suppression* dépend de la structure de l'arbre.



## Autre arbres de recherche

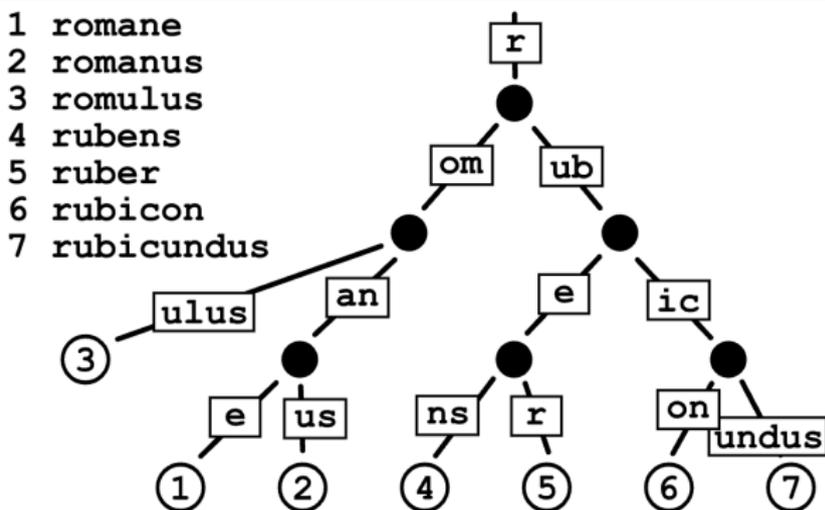
- ▶ Dépendent des applications !<sup>1</sup>
- ▶ *Radix Tree*<sup>2</sup> pour comparer 2 chaînes de caractères
- ▶ **Arbres AVL** dont le but est d'équilibrer les arbres afin qu'ils gardent une hauteur logarithmique
- ▶ **B-Tree d'ordre M**. Chaque nœud a au plus  $m$  enfants. La racine a au moins 2 enfants (si ce n'est pas une feuille). Toutes les feuilles sont au même niveau. très utilisé dans les grands systèmes de fichiers où de gros blocs de données sont manipulés. C'est une généralisation des arbres binaires de recherche.
- ▶ **B+ -Tree**. Est similaire à un B-Tree mais où les nœuds ne contiennent que des clés
- ▶ **Red-Black Binary Tree**. Est un type d'arbre binaire équilibré, où chaque nœud possède une information supplémentaire : une couleur (rouge ou noir). Toutes les feuilles ont un enfant NIL (noir). Les nœuds rouges n'ont pas d'enfants. Tout chemin vers des nœuds NIL possède le même nombre de nœuds noirs.

<https://www.cs.usfca.edu/~galles/visualization/BTree.html>

<sup>1</sup><http://pageperso.lif.univ-mrs.fr/~francois.denis/algoMPCI/chap1.pdf>

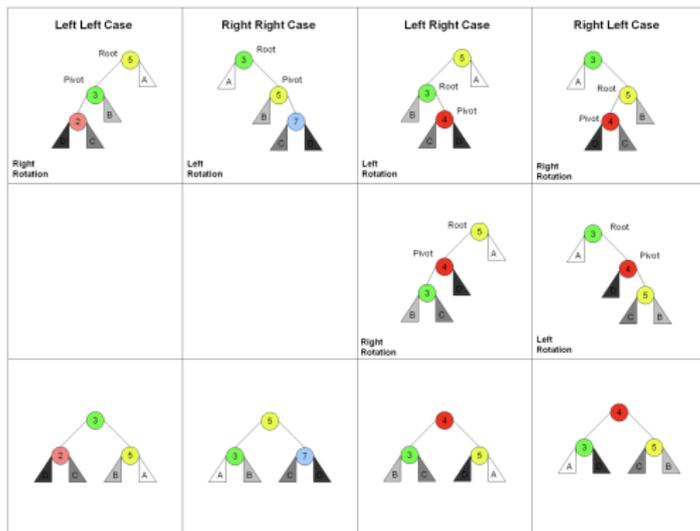
<sup>2</sup>[https://en.wikipedia.org/wiki/Radix\\_tree](https://en.wikipedia.org/wiki/Radix_tree)

**Radix Tree.** Une structure de données compacte permettant de représenter un ensemble de mots adaptée pour la recherche



[https://fr.wikipedia.org/wiki/Arbre\\_radix](https://fr.wikipedia.org/wiki/Arbre_radix)

**Arbres AVL.** Un arbre binaire dont la différence des sous-arbres ne dépasse pas 1. Un arbre est dit AVL si et seulement si  $H(g) - H(d) \leq 1$ . Il faut donc après chaque insertion/suppression d'assurer que la propriété AVL est respectée. Pour cela on utilise une méthode de *rotation*.



Tri

Un algorithme de **tri** consiste à mettre les éléments qui appartiennent à une liste dans un certain ordre.

Autrement dit étant donné un tableau d'entiers  $T$  (entrée), trier  $T$  dans l'ordre croissant (sortie). Un tri a plusieurs propriétés :

- ▶ **Relation d'ordre** : permet de comparer deux éléments. Par exemple ordre lexicographique (ordre de l'alphabet)
- ▶ **Stabilité** : plusieurs tris donnent le même résultat indépendamment des conditions initiales
- ▶ **In situ** : demandent de la place supplémentaire ou pas pour faire les changements
- ▶ **En ligne (online)** : la solution en cours de construction a une certaine validé

Même si problème simple et connu, de nombreuses complexités et structures de données différentes. Les tris permettent aussi de préparer à d'autres algorithmes.

# Concepts avancés

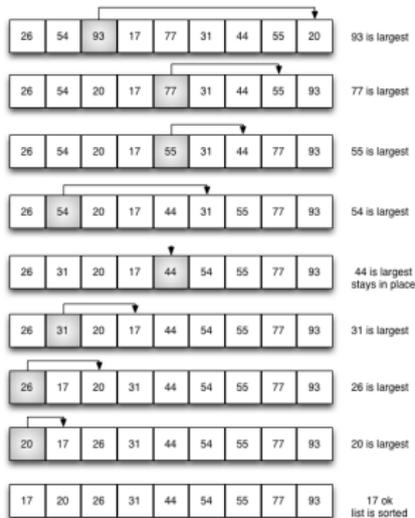
## Comparaison des complexités

### Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
Mergesort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Timsort	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Heapsort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Tree Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(n)$
Shell Sort	$\Omega(n \log(n))$	$\theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
Bucket Sort	$\Omega(n+k)$	$\theta(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$\Omega(nk)$	$\theta(nk)$	$O(nk)$	$O(n+k)$
Counting Sort	$\Omega(n+k)$	$\theta(n+k)$	$O(n+k)$	$O(k)$
Cubesort	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$

<http://bigocheatsheet.com/>

**Tri par sélection (Selection Sort)** parcourt toute la liste pour identifier le maximum (minimum), le place à la fin (au début), et recommence.



Il s'agit de la méthode de tri la plus simple et naïve.. et donc la méthode la moins performante. Complexité de  $\mathcal{O}(n^2)$

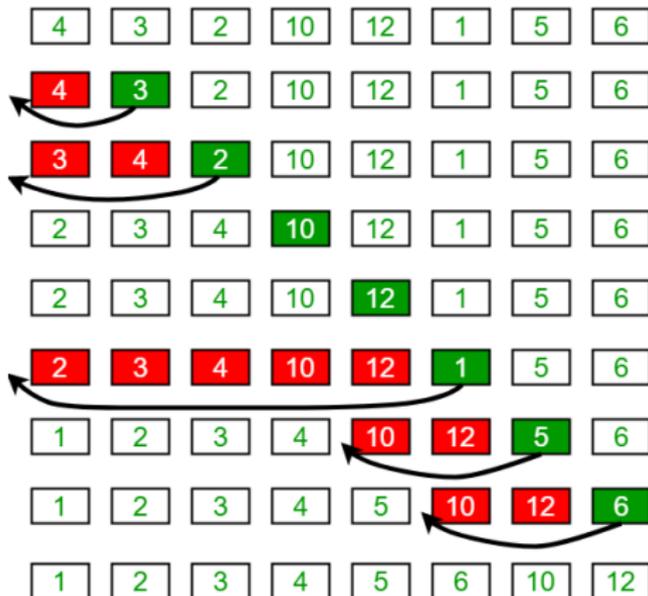
## Trip par sélection (Selection Sort)

```
def selectionSort(alist):  
    for l in range(len(alist)-1,0,-1):  
        positionOfMax=0  
  
        for location in range(1, l+1):  
            if alist[location]>alist[positionOfMax]:  
                positionOfMax = location  
  
        temp = alist[l]  
  
        alist[l] = alist[positionOfMax]  
  
        alist[positionOfMax] = temp  
  
alist = [54,26,93,17,77,31,44,55,20]  
selectionSort(alist)  
print(alist)
```

Il s'agit de la méthode de tri la plus simple et naïve.. et donc la méthode la moins performante. Complexité de  $\mathcal{O}(n^2)$

**Tri par insertion (Insertion Sort)** : similaire à la façon dont on classe les cartes, on parcourt de gauche à droite et on positionne le nouvel élément non trié dans la sous-liste (triée).

Insertion Sort Execution Example



Complexité de  $\mathcal{O}(n^2)$

## Tri par insertion (Insertion Sort)

```
def insertionSort(arr):  
  
    # Parours du tableau  
    for i in range(1, len(arr)):  
  
        key = arr[i]  
  
        # Déplace les éléments  
        j = i-1  
        while j >=0 and key < arr[j] :  
            arr[j+1] = arr[j]  
            j -= 1  
        arr[j+1] = key  
  
arr = [12, 11, 13, 5, 6]  
insertionSort(arr)  
for i in range(len(arr)):  
    print("%d" %arr[i])
```

## Trip par insertion binaire (Binary Insertion Sort).

---

**Algorithm 2:** The Binary Insertion Sort

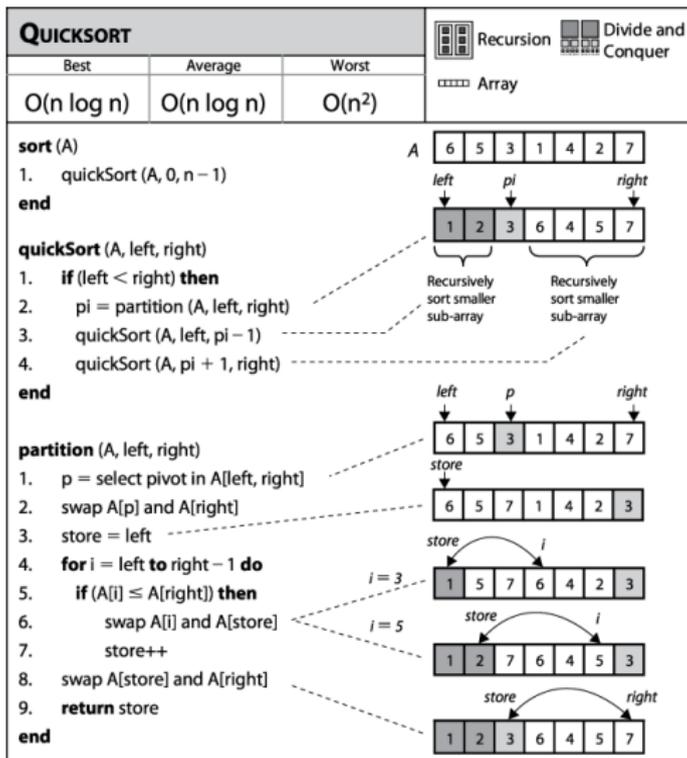
---

```
Data:  $x = [x_1, x_2, \dots, x_n]$  - the input array  
Result:  $x$  is sorted so that  $x[i] \leq x[i+1]$  for all  $i = 1, 2, \dots, n-1$   
for  $i \leftarrow 2, 3, \dots, n$  do  
    // Find where to insert  $x[i]$  so that  $x[1:i]$  is sorted  
    // non-decreasingly.  
     $j \leftarrow \text{BINARY-SEARCH}(x[1:(i-1)], x[i])$   
     $k \leftarrow i - 1$   
    while  $k > j$  do  
        // Swap  $x[i]$  with greater elements  
        Swap  $x[k]$  and  $x[k-1]$   
         $k \leftarrow k - 1$   
    end  
end  
  
// Binary search returns the index of  $b$  of the position  $b$   
// should be at in the sorted array  $a_1 \leq a_2 \leq \dots \leq a_m$ .  
function BINARY-SEARCH( $[a_1, a_2, \dots, a_m], b$ ):  
     $low \leftarrow 1$   
     $high \leftarrow m$   
    while  $low < high$  do  
         $mid \leftarrow \frac{low+high}{2}$   
        if  $a_{mid} = b$  then  
            | return  $mid$   
        else if  $a_{mid} > b$  then  
            |  $low \leftarrow mid + 1$   
        else  
            |  $high \leftarrow mid - 1$   
        end  
    end  
    // If we've come this far,  $b$  isn't in the array, but  
    // its index should be  $low$  ( $= high$ ).  
    return  $low$   
end
```

Complexité de  $\mathcal{O}(n^2)$  (toujours autant de déplacements d'éléments)



## Tri rapide (Quick Sort)



## Tri rapide (Quick Sort)

```
def quicksort(t):
    if t == []:
        return []
    else:
        pivot = t[0]
        t1 = []
        t2 = []
        for x in t[1:]:
            if x < pivot:
                t1.append(x)
            else:
                t2.append(x)
        return quicksort(t1) + [pivot] + quicksort(t2)

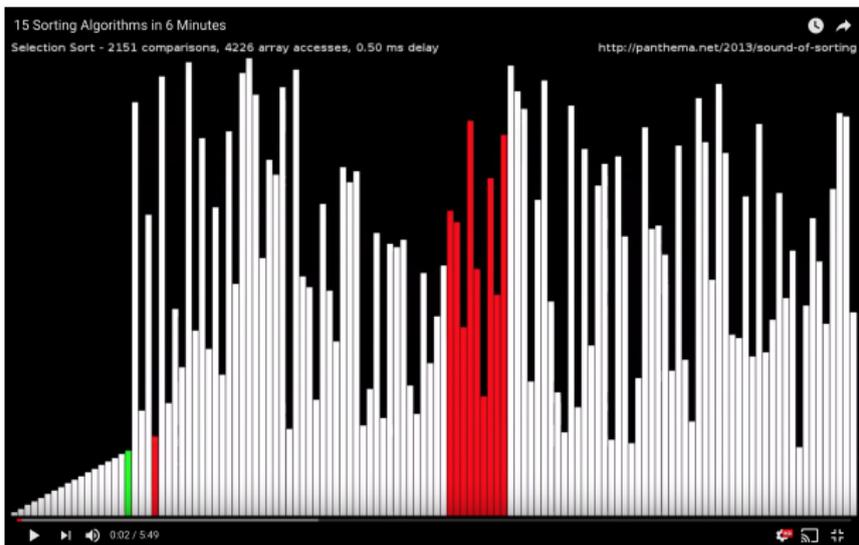
quicksort([1,4, 5])
```

Comparaison des complexités des méthodes de tri

## Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$

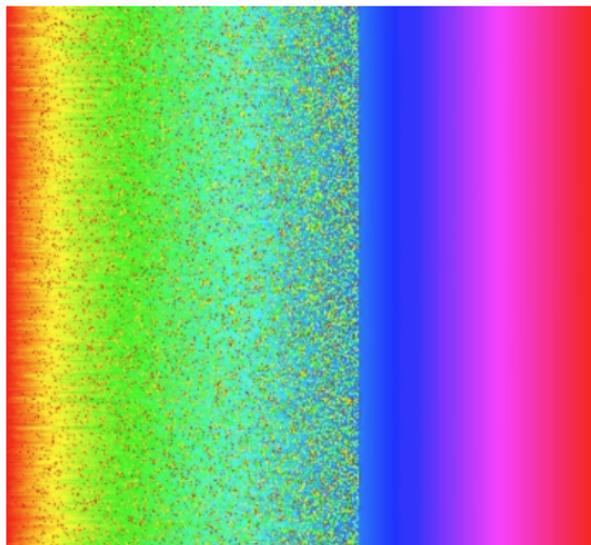
*Comparaison des méthodes de tris*



*15 Sorting Algorithms in 6 Minutes*

<https://www.youtube.com/watch?v=kPRA0W1kECg>

*Comparaison des méthodes de tris*



*Sorting visualizations*

<https://imgur.com/gallery/RM3w1>

# Concepts avancés

## Tri

 <b>Play All</b>	 Insertion	 Selection	 Bubble	 Shell	 Merge	 Heap	 Quick	 Quick3
 Random								
 Nearly Sorted								
 Reversed								
 Few Unique								

*Sorting Algorithms Animations*

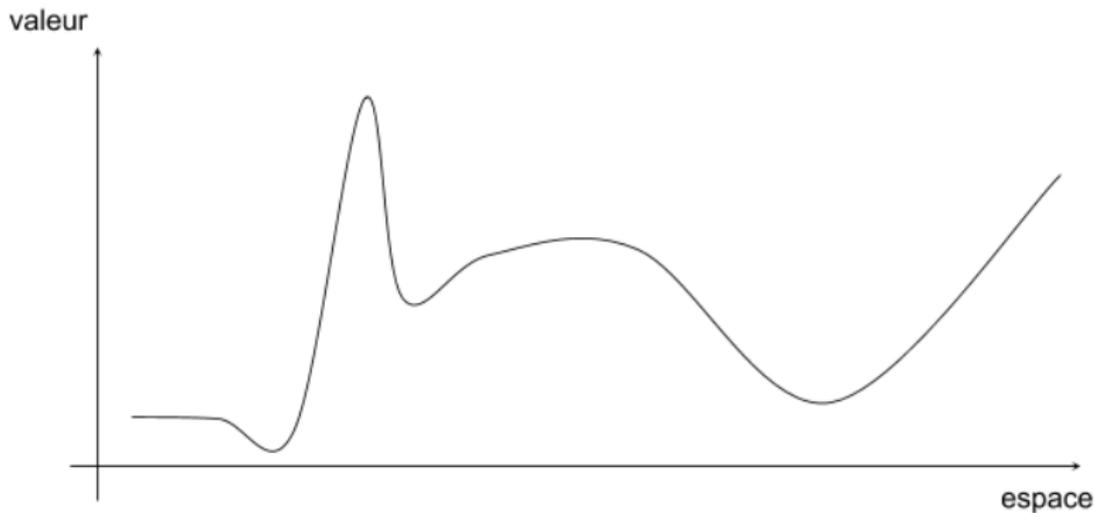
<https://www.toptal.com/developers/sorting-algorithms>

# Heuristiques

## Concepts avancés

### Exemples d'algorithmes

Quel algorithme permet de trouver le minimum de cette fonction ?



**Calcul de Pi.** Estimation de la surface de 1/4 de cercle ( $r=1$ )  $A = \frac{\pi}{4}$

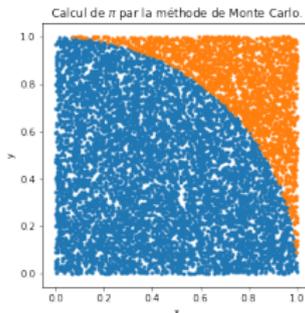
$$\lim_{n \rightarrow \infty} \frac{4}{n} \sum_{i=1}^n Y_i = 4\mathbb{E}(Y_0) = \pi \quad (1)$$

```
N = 10000
x = np.random.random(size=N)
y = np.random.random(size=N)

z = (x**2+y**2)<1 # on garde les points dans le 1/4 de cercle

pi_est = 4*z.sum()/N

print('Estimation de pi: {}'.format(pi_est))
```



(rappel) Un algorithme est un **ensemble d'instructions, non-ambiguë, permettant de résoudre un problème.**

- ▶ Communicable
- ▶ Efficace
- ▶ Complet, termine et correct
- ▶ Déterministe

(rappel) Un algorithme est un **ensemble d'instructions, non-ambiguë, permettant de résoudre un problème.**

- ▶ Communicable
- ▶ Efficace
- ▶ Complet, termine et correct
- ▶ Déterministe

De nos jours, les programmes informatiques n'ont pas ces propriétés et la fonction d'algorithme est *relâchée*

(rappel) Un algorithme est un **ensemble d'instructions, non-ambiguë, permettant de résoudre un problème.**

- ▶ Communicable
- ▶ Efficace
- ▶ Complet, termine et correct
- ▶ Déterministe

De nos jours, les programmes informatiques n'ont pas ces propriétés et la fonction d'algorithme est *relâchée*

- ▶ **Non-Communicable** → **Modèle d'apprentissage**

(rappel) Un algorithme est un **ensemble d'instructions, non-ambiguë, permettant de résoudre un problème.**

- ▶ Communicable
- ▶ Efficace
- ▶ Complet, termine et correct
- ▶ Déterministe

De nos jours, les programmes informatiques n'ont pas ces propriétés et la fonction d'algorithme est *relâchée*

- ▶ **Non-Communicable** → **Modèle d'apprentissage**
- ▶ **Non-Efficace** → **Heuristique**
- ▶ **Non-Complet, termine et correct** → **Heuristique**
- ▶ **Non-Déterministe** → **Heuristique**

(rappel) Un algorithme est un **ensemble d'instructions, non-ambiguë, permettant de résoudre un problème.**

Pourquoi doit-on relâcher ces contraintes et étendre la définition ?

- ▶ Les problèmes actuels sont de plus en plus **complexes**
- ▶ Algorithmes ne sont qu'une (sous-)partie de la solution
- ▶ Volumes de données de plus en plus grands
  - ▶ Nécessite de l'optimisation
  - ▶ ..mais une opportunité pour gagner en performances et permettre de nouvelles applications/nouveaux usages
- ▶ Besoin d'approches génériques qui fonctionnent dans plusieurs contextes

A noter que parfois le problème n'est pas connu (rôle de la Recherche et R&D de définir le problème, le formaliser, proposer une solution, etc.).

## Concepts avancés

# Algorithmes Probabilistes

Un algorithme est dit **probabiliste** si lors de son exécution il utilise une source de hasard et donc deux exécutions donnent lieu à des résultats différents

### Question :

*Connaissez-vous un algorithme probabiliste?*

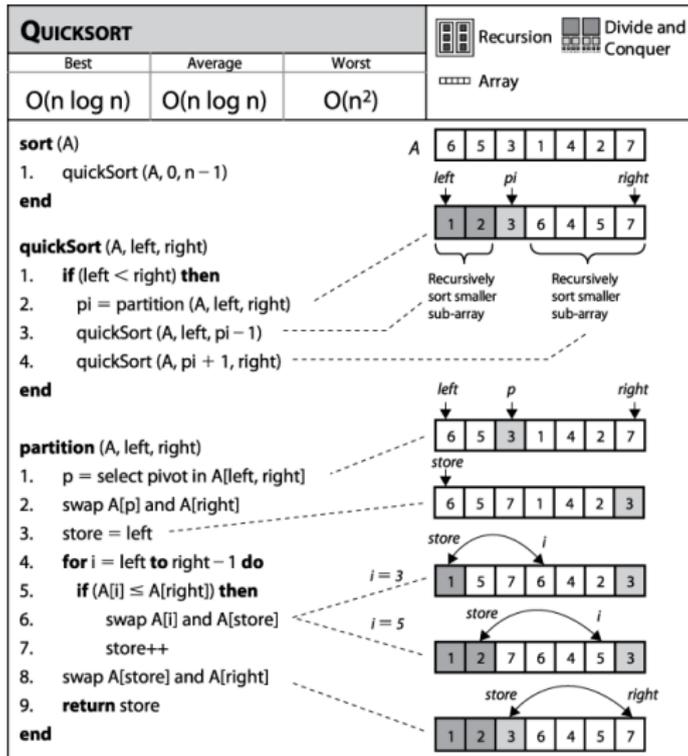
Un algorithme est dit **probabiliste** si lors de son exécution il utilise une source de hasard et donc deux exécutions donnent lieu à des résultats différents

### Question :

*Connaissez-vous un algorithme probabiliste?*

- ▶ Quicksort peut avoir une composante aléatoire (un choix du pivot aléatoire permet souvent d'éviter le pire des cas, car si le choix du pivot est connu alors un *adversaire* peut en profiter)
- ▶ Les tris peuvent être *instables* car des valeurs identiques peuvent être ordonnées différemment (pour un même jeu de données initial)

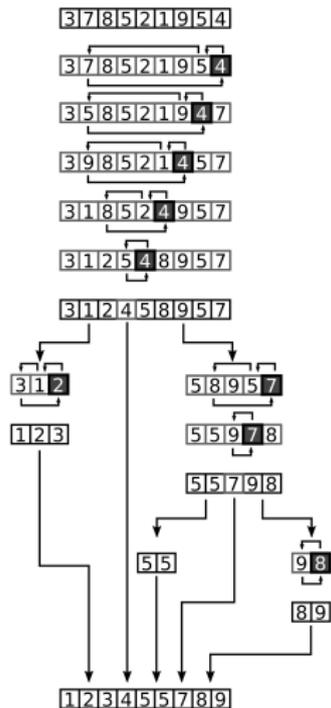
### Trip rapide (Quick Sort)



# Concepts avancés

## Algorithmes Probabilistes

### Trip rapide (Quick Sort) pire des cas



### Quick Sort (Randomized)

```
import random
def randomized_quicksort(L):
    if len(L) == 1 or len(L) == 0:
        return L

    # CHOIX DU PIVOT i ?...
    pivot = L[i]
    smaller = [elt for elt in L if elt < pivot]
    larger = [elt for elt in L if elt >= pivot]
    return quicksort(smaller) + [pivot] + quicksort(larger)
```

Comment choisir le pivot?

### Quick Sort (Randomized)

```
import random
def randomized_quicksort(L):
    if len(L) == 1 or len(L) == 0:
        return L

    # CHOIX DU PIVOT i ?...
    pivot = L[i]
    smaller = [elt for elt in L if elt < pivot]
    larger = [elt for elt in L if elt >= pivot]
    return quicksort(smaller) + [pivot] + quicksort(larger)
```

Comment choisir le pivot?

```
# randint upper bound is inclusive
i = random.randint(len(L) - 1)
```

## Concepts avancés

# Algorithmes Probabilistes

Deux types d'approches aléatoires :

- ▶ **"Las Vegas"** → donne toujours une bonne valeur correcte quel que soit le moment d'exécution de l'algorithme
- ▶ **"Monte Carlo"** → donne toujours une valeur mais pas forcément correcte à tout instant

**Pros** : Meilleures performances dans certains cas; Permet d'éviter d'être attaqués par des cas adverses.

**Cons** : Algorithmes peuvent être instables; Génération de nombres *vraiment* aléatoires est difficile.

→ Il existe des méthodes de *de-randomisation* qui est un compromis espace et temps (voir automates déterministes/non-déterministes).

Une **heuristique** permet de fournir *rapidement* un résultat approximatif

Il s'agit de faire un compromis :

- ▶ Optimalité
- ▶ Complétude on n'étudie pas toutes les solutions
- ▶ Confiance : l'algorithme fournir un interval de confiance de la solution

Une **heuristique** permet de fournir *rapidement* un résultat approximatif

Il s'agit de faire un compromis :

- ▶ Optimalité
- ▶ Complétude on n'étudie pas toutes les solutions
- ▶ Confiance : l'algorithme fournir un interval de confiance de la solution

Question :

*Connaissez-vous des heuristiques ?*

Une **heuristique** permet de fournir *rapidement* un résultat approximatif

Il s'agit de faire un compromis :

- ▶ Optimalité
- ▶ Complétude on n'étudie pas toutes les solutions
- ▶ Confiance : l'algorithme fournir un interval de confiance de la solution

Question :

*Connaissez-vous des heuristiques ?*

- ▶ Heuristiques gloutonnes
- ▶ Recherche A\*
- ▶ Algorithmes glouton (propose une solution qui n'est pas optimale globale pour un problème difficile/NP-Complet)

## Concepts avancés

# Algorithme A\*

A-étoile (A\*) intuition :

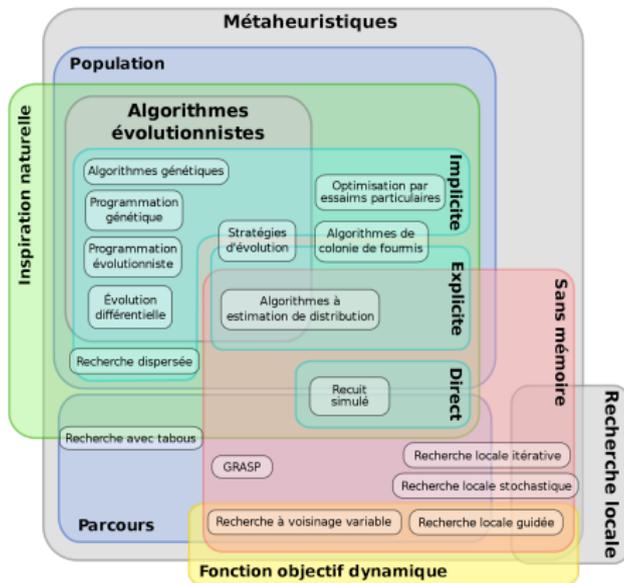
- ▶ on a une information supplémentaire sur la destination
- ▶ définition d'une heuristique qui permet d'organiser la liste d'attente des sommets à parcourir.

```
def heuristic(a, b):  
    # Distance de Manhattan  
    return abs(a.x - b.x) + abs(a.y - b.y)
```



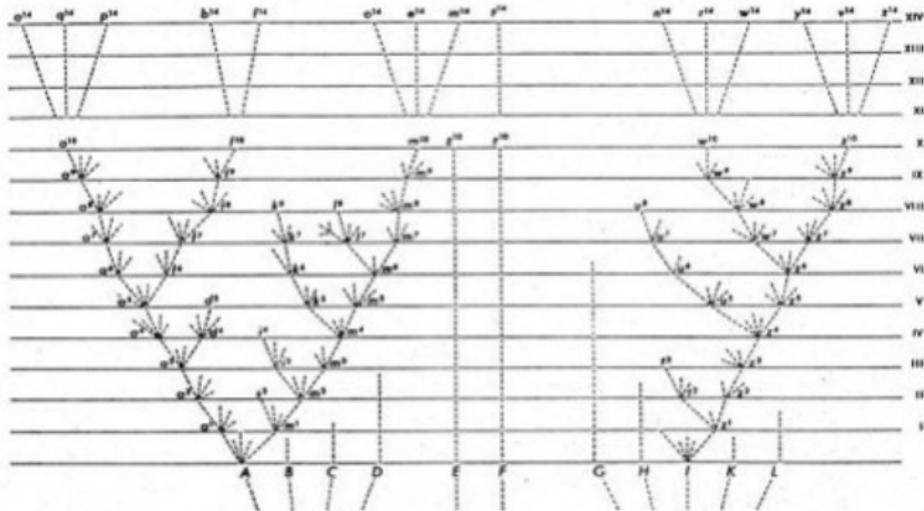
## Concepts avancés Heuristiques

Une **meta-heuristique** est une heuristique permettant l'optimisation de plusieurs ou de tout une gamme de problème sans adaptation spécifique.



# Concepts avancés Heuristiques

**Exemple de meta-heuristique : Bio-inspirées.** Utilisation de la métaphore *génétique* afin de résoudre un problème d'optimisation. Chaque algorithme suit une évolution (processus stochastique)



**Exemple de meta-heuristique :** Bio-inspirées. Utilisation de la métaphore "génétique" afin de résoudre un problème d'optimisation.

a) Construction et évaluation d'une population initiale

b) Jusqu'à atteindre un critère d'arrêt :

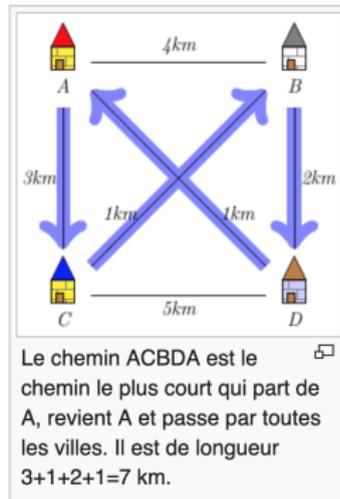
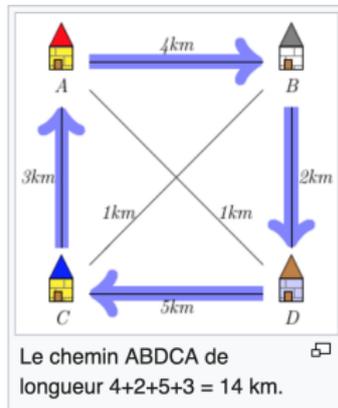
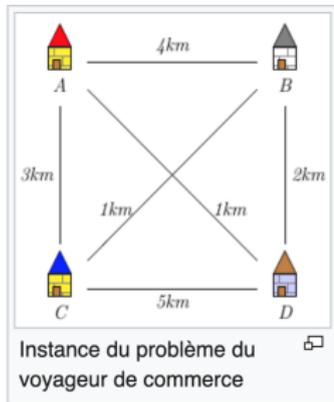
1. Sélection d'une partie de la population,
2. Reproduction des individus sélectionnés,
3. Mutation de la descendance,
4. Évaluation du degré d'adaptation de chaque individu,
5. Remplacement de la population initiale par une nouvelle population.

Application : TSP, jeux vidéos, ..

# Concepts avancés

## Heuristiques

### Problème du voyageur de commerce (formulation)



[https://fr.wikipedia.org/wiki/Probl%C3%A8me\\_du\\_voyageur\\_de\\_commerce](https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_voyageur_de_commerce)

Problème du voyageur de commerce (heuristique)

Nearest Neighbour (NN)

- [1] Choisir un sommet  $v_1 \in V$   
Poser  $k \leftarrow 1$
  
  - [2] Tant que  $k < n$ 
    - $k \leftarrow k + 1$
    - choisir  $v_k$  dans  $V \setminus \{v_1, v_2, \dots, v_{k-1}\}$   
qui minimise  $c_{v_{k-1}, v_k}$
- Retourner le tour  $(v_1, v_2, \dots, v_n)$

Complexité  $O(n^2)$

## Exemple



<https://www.askforgametask.com/tutorial/machine-learning-algorithm-flappy-bird/>

<https://www.youtube.com/watch?v=qv6UVOQ0F44>

## Concepts avancés

# Heuristiques

Question :

*Que faire dans les cas difficiles ?*

## Concepts avancés

# Heuristiques

Question :

*Que faire dans les cas difficiles ?*

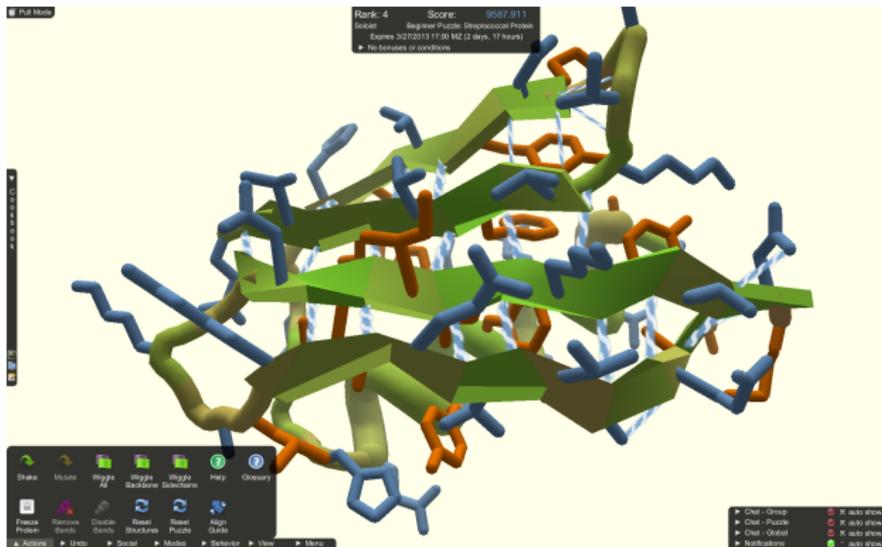
Interaction avec l'humain ! (ex : Crowdsourcing, Gamification)

# Concepts avancés Heuristiques

## Question :

*Que faire dans les cas difficiles ?*

Interaction avec l'humain ! (ex : Crowdsourcing, Gamification)



<https://fold.it/portal/>

# Automates

Concepts avancés  
Cas d'étude

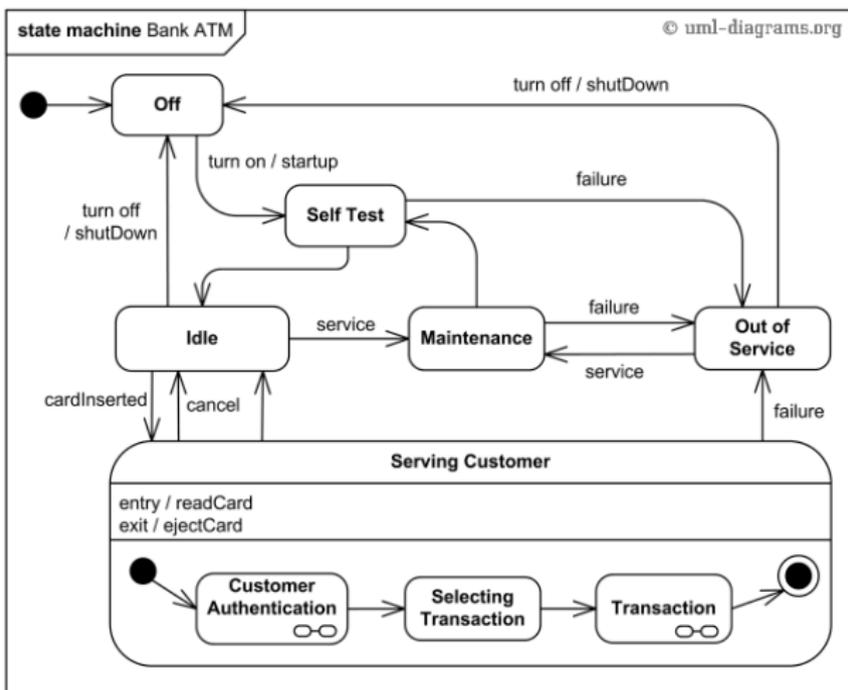
Question :

*Comment modéliseriez-vous un problème de distributeur de billet ?*



# Concepts avancés

## Banque (vue UML)

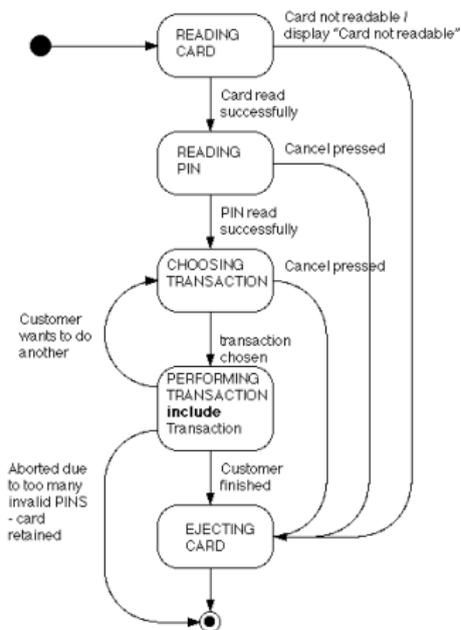


<https://www.uml-diagrams.org/bank-atm-uml-state-machine-diagram-example.html>

# Concepts avancés

## Banque (vue UML)

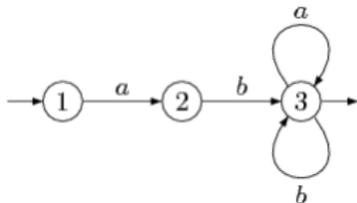
State-Chart for One Session



<http://www.math-cs.gordon.edu/courses/cs211/ATMExample/Statecharts.html>

Un **automate** est un outil de calcul qui possède un nombre fini d'**états**, qui possède un état initial, courant et final

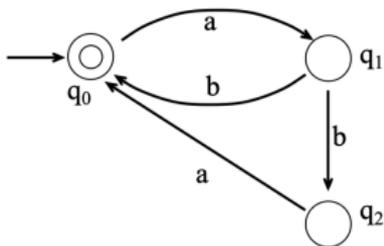
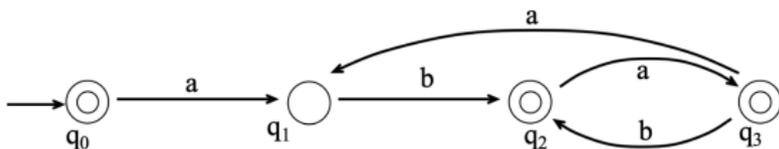
- ▶ Un automate lit un mot en entrée et ensuite passe d'un état à un autre
- ▶ Peut être vu comme un graphe étiqueté qui indique les transitions possibles
- ▶ Représentation graphique où les sommets sont des nœuds et transitions des arcs



Question :

*Que fait cet automate ?*

**Automates déterministes vs non-déterministes** : soit chaque état possède au plus une transition pour chaque symbole d'entrée (et même exactement une dans le cas où l'automate est complet); soit un symbole d'entrée peut étiqueter une, plusieurs ou aucune transition pour un état donné.



## Concepts avancés

# Automates

Formellement un automate  $A = (\Sigma, S, s_0, \delta, F)$  est composé de :

- ▶  $\Sigma$ , un ensemble fini, non vide de lettres qui est l'alphabet d'entrée,
- ▶  $S$ , un ensemble fini, non vide d'états,
- ▶  $s_0$ , l'état initial, élément de  $S$ . Dans un automate non déterministe,  $s_0$  peut être un ensemble d'états.
- ▶  $\delta$ , la fonction de transition d'états:  $\delta : S \times \Sigma \rightarrow S$
- ▶  $F$  est l'ensemble des états terminaux, un sous-ensemble (éventuellement vide) de  $S$ .

Types d'automates

- ▶ Automates (à états) finis déterministes  
(*Deterministic Finite Automata – DFA*)
- ▶ Automates (à états) finis non déterministes  
(*Non-deterministic Finite Automata – NFA*)
- ▶ Automates à pile  
(*Pushdown Automata – PDA*)

Concepts avancés  
Automates

Question :

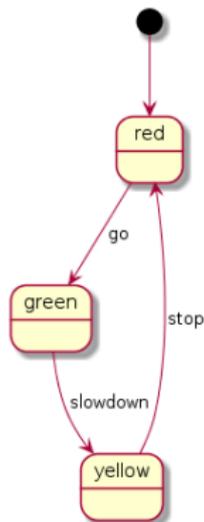
*Construire l'automate correspondant à un feu de signalisation ? ( $\Sigma$  ?  $S$  ?  
 $s_0$  ?  $\delta$  ?  $F$  ?)*

## Concepts avancés

# Automates

### Question :

Construire l'automate correspondant à un feu de signalisation ? ( $\Sigma$  ?  $S$  ?  
 $s_0$  ?  $\delta$  ?  $F$  ?)



## Concepts avancés

# Automate en Python

```
class DFA:
    def __init__(self, alphabet):
        self.states = []
        self.transitions = {}
        self.init = None
        self.finals = []
        self.alphabet = ""
        for s in alphabet:
            if s not in self.alphabet:
                self.alphabet += s

    def add_state(self, state, final = False):
        if state in self.states:
            print("error : state '" + state + "' already exists.")
            return
        self.transitions[state] = []
        self.states.append(state)
        if final:
            self.finals.append(state)

    def valid_symbol(self, symbol):
        if symbol not in self.alphabet: return False
        return True

    ...
```

## Concepts avancés

# Automate en Python (Module)

```
from automaton import *

class TrafficLight(Automaton):

    go = Event(" red", " green")
    slowdown = Event(" green", " yellow")
    stop = Event(" yellow", " red")

crossroads = TrafficLight(initial_state=" red")
assert crossroads.state == " red"

crossroads.go()
assert crossroads.state == " green"

crossroads.slowdown()
assert crossroads.state == " yellow"

crossroads.event(" stop")
assert crossroads.state == " red"

crossroads = TrafficLight(initial_state=" red")
crossroads.stop()

automaton.InvalidTransitionError:
The specified event 'Event(source_states=('yellow',), dest_state='red')'
is invalid in current state 'red'.
```

<https://pypi.org/project/python-automaton/>

## Concepts avancés

# Langage reconnu

Un **langage reconnu** par un automate  $A$  est l'ensemble des mots  $m \in \Sigma^*$  tels que  $\delta(p_0, m)$  est un état final

Construire les automates qui valident :

1. Le langage des mots contenant au plus une fois la lettre  $a$
2. Le langage des mots contenant un nombre pair de fois la lettre  $a$
3. Le langage des mots admettant  $aba$  pour sous-mot

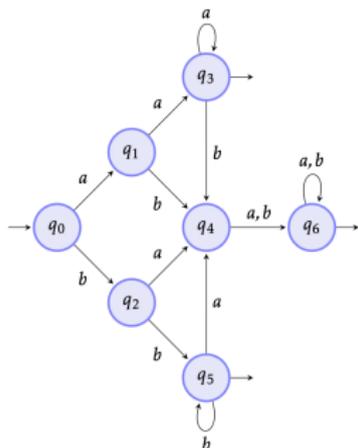
## Concepts avancés

# Langage reconnu

Un **langage reconnu** par un automate A est l'ensemble des mots  $m \in \Sigma$  tels que  $\delta(p_0, m)$  est un état final

### Question :

*Quel langage reconnaît cet automate ?*



## Détermination d'un automate

Il est possible de **rendre déterministe** tout automate non-déterministe en construisant itérativement un automate déterministe équivalent dont un état  $q$  peut correspondre à plusieurs états  $q_1, q_2$

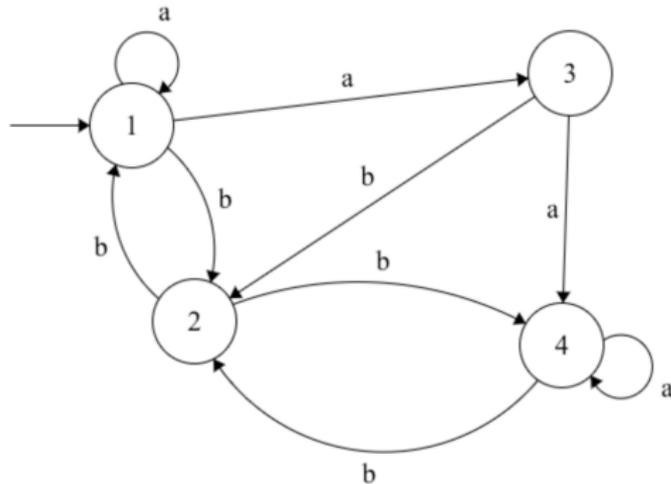
## Concepts avancés

### Détermination d'un automate

Il est possible de **rendre déterministe** tout automate non-déterministe en construisant itérativement un automate déterministe équivalent dont un état  $q$  peut correspondre à plusieurs états  $q_1, q_2$

### Question :

*Quel est l'automate déterministe équivalent à cet automate ?*



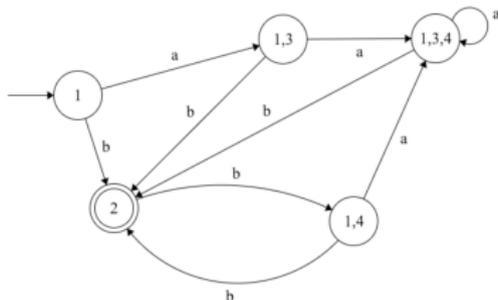
## Concepts avancés

# Détermination d'un automate

Il est possible de **rendre déterministe** tout automate non-déterministe en construisant itérativement un automate déterministe équivalent dont un état  $q$  peut correspondre à plusieurs états  $q_1, q_2$

Question :

*Solution*



Est-il complet (pour toute lettre existe-t-il une transition?)<sup>3</sup>

<sup>3</sup><http://www.momirandum.com/automates-finis/Determinisationdunautomate.html>

# Problèmes Difficiles et Recherche

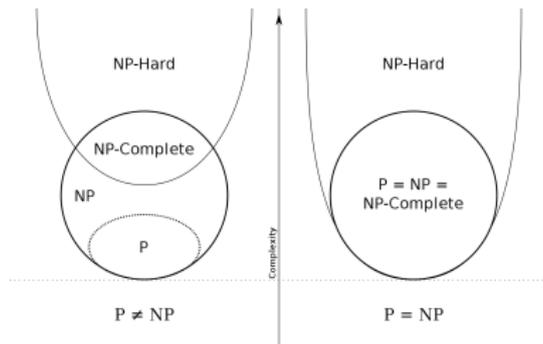
## Concepts avancés

# Problemes NP-Complets

Un problème est dit de classe **P** si il est résoluble en un temps polynomial  $O(n^k)$  par une machine de Turing déterministe.

Un problème est dit de classe **NP** (non déterministe polynomial) ne possède pas de solution en temps polynomial, mais une solution non-déterministe avec une machine de Turing (ses solutions peuvent cependant être vérifiés par une machine de Turing déterministe)

Un problème est dit **NP-Hard** est une classe de problèmes vers lesquels tous les problèmes NP peuvent être réduits



## Exemple

### Exemples de problèmes NP-Complets

- ▶ Voyageur de commerce
- ▶ Sac à dos
- ▶ Coloration de graphe
- ▶ Recherche de chemin
- ▶ Emploi du temps
- ▶ ...

[https://en.wikipedia.org/wiki/List\\_of\\_NP-complete\\_problems](https://en.wikipedia.org/wiki/List_of_NP-complete_problems)

Probleme SAT<sup>4</sup> (Boolean **S**atisfability Problem)

- ▶ Un des problème le plus étudié en Informatique
- ▶ Une formule booléenne est constituée de variables  $a, b, ..$  avec des valeurs (True/1, False/0) et des opérateurs (Not) et and ( $\wedge$ ) / or ( $\vee$ ).
- ▶ *Existe-t-il des valeurs pour les variables afin de satisfaire une formule booléenne ? (la rendre égale à True)*

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Boolean\\_satisfiability\\_problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem)

<sup>5</sup><https://www.cs.toronto.edu/~sacook/homepage/1971.pdf>

#### Probleme SAT<sup>4</sup> (Boolean **S**atisfability Problem)

- ▶ Un des problème le plus étudié en Informatique
- ▶ Une formule booléenne est constituée de variables  $a, b, ..$  avec des valeurs (True/1, False/0) et des opérateurs (Not) et and ( $\wedge$ ) / or ( $\vee$ ).
- ▶ *Existe-t-il des valeurs pour les variables afin de satisfaire une formule booléenne ? (la rendre égale à True)*

#### Exemples

- ▶  $a \text{ AND NOT } b$  est satisfaisable pour les valeurs  $a = \text{TRUE}$  et  $b = \text{FALSE}$
- ▶  $a \text{ AND NOT } a$  n'est pas satisfaisable

SAT est le premier problème a avoir été démontré NP-Complet (Théorème de Cook–Levin)<sup>5</sup>

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Boolean\\_satisfiability\\_problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem)

<sup>5</sup><https://www.cs.toronto.edu/~sacook/homepage/1971.pdf>

Notes :

- ▶ Un problème  $X$  de classe  $\mathbf{P}$  est réductible à un problème  $Y$  de la même classe si il existe un algorithme polynomiale permettant de ramener  $X$  à  $Y$
- ▶ Si  $X$  est réductible à un problème de la classe  $\mathbf{P}$  alors il est de classe  $\mathbf{P}$  lui-même.
- ▶ Beaucoup de problèmes  $\mathbf{NP}$  sont réductibles au problème SAT et sont dit NP-complets (et donc inutile de chercher une solution en temps polynomial)
- ▶  $\mathbf{P} = \mathbf{NP}$  est le problème on résolu le plus important en informatique (1M de dollar)<sup>6</sup>

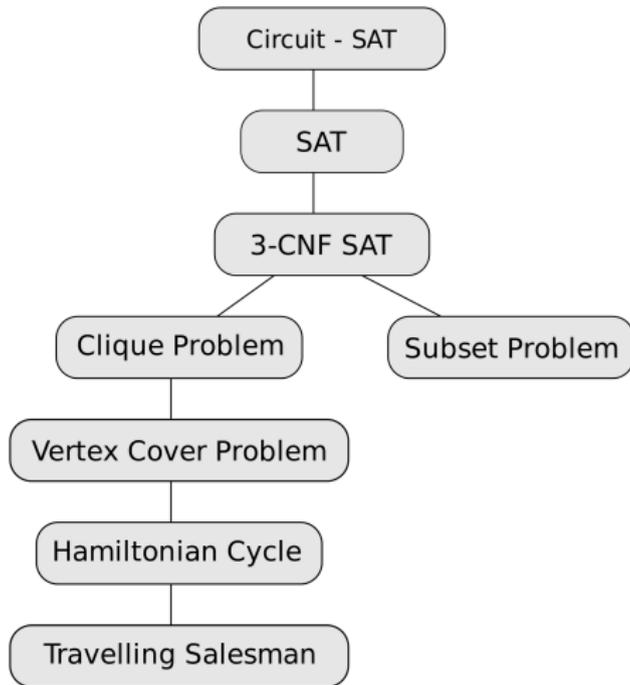
Pour résoudre ces problèmes soit approches vues précédemment (randomisation, approximation, heuristique, ..), soit réduction.

---

<sup>6</sup><http://www.claymath.org/millennium-problems>

# Concepts avancés

## Exemple de réduction

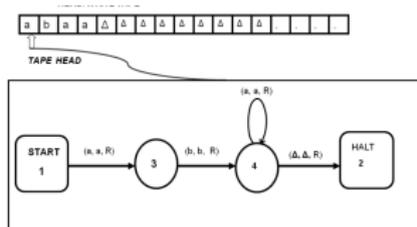


## Concepts avancés Machine de Turing

Une Machine de Turing est un modèle abstrait de calcul composé comme suit :

1. **Un ruban infini** divisé en cases consécutives (qui contiennent des symboles issus d'un alphabet fini)
2. **Une tête de lecture/écriture** qui se déplace vers la gauche ou la droite du ruban
3. **Un registre d'état** qui mémorise l'état en cours
4. **Une table d'actions** qui permet l'écriture sur le ruban

Imaginé par Alan Turing en 1936 (avant l'ordinateur). Encore très utilisée en informatique théorique et calcul de complexité.



# Concepts avancés

## Machine de Turing

Exemple de table infinie

Ancien état	Symbole écrit	Mouvement	Nouvel état
a	0	Droite	b
b	1	Droite	a

Exécution de la Machine  
infinie

Etape	Etat	Ruban
1	a	0
2	b	01
3	a	010
4	b	0101
5	a	01010
6	b	010101
7	a	0101010
8	b	01010101
...	...	01010101...

# Concepts avancés

## Machine de Turing

Exemple de table infinie

Ancien état	Symbole écrit	Mouvement	Nouvel état
a	0	Droite	b
b	1	Droite	a

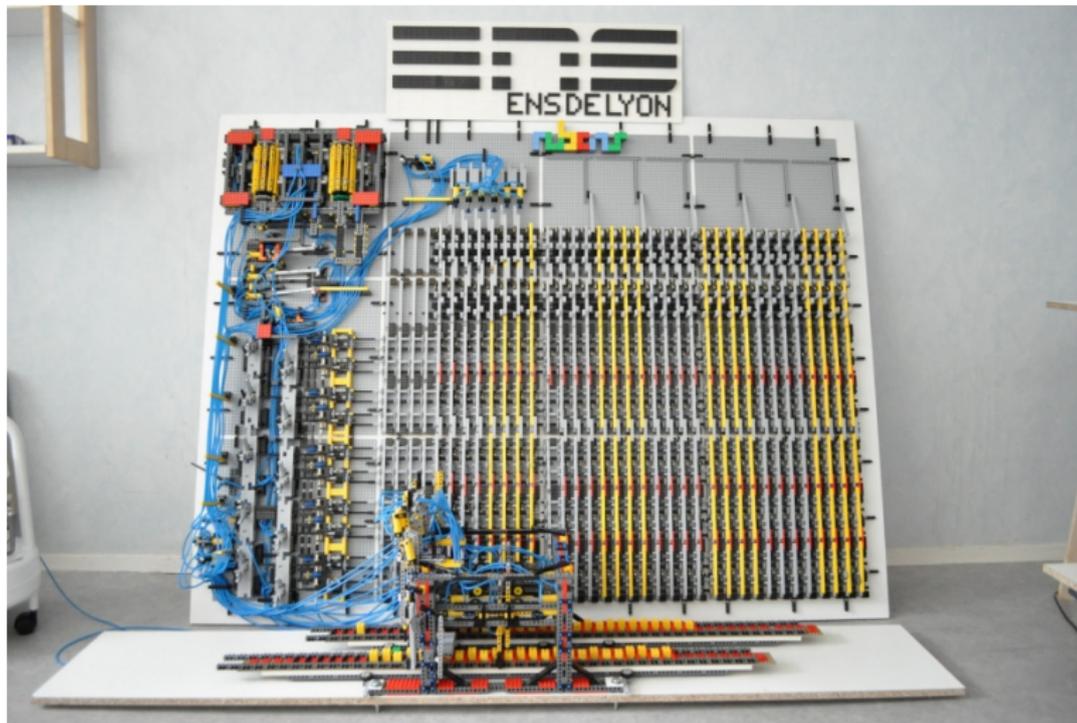
Exécution de la Machine  
infinie

Etape	Etat	Ruban
1	a	0
2	b	01
3	a	010
4	b	0101
5	a	01010
6	b	010101
7	a	0101010
8	b	01010101
...	...	01010101...

Cette machine permet de créer le nombre infini 010101010101... ce qui correspond à une boucle infinie.

# Concepts avancés

## Machine de Turing



Il existe de nombreux autres types de structure de données, algorithmes !

- ▶ Graphes avancés
- ▶ Programmation parallèle
- ▶ Cryptographie, compression
- ▶ Programmation fonctionnelle
- ▶ Combinatoire
- ▶ Architecture et arithmétique des ordinateurs
- ▶ Preuve et vérification de programmations
- ▶ Systèmes d'Information (Réseau, Bases de données, ..)
- ▶ Apprentissage, apprentissage profond
- ▶ Visualisation de données et Interfaces Homme/Machine
- ▶ Traitement de langage naturel (NLP)
- ▶ Interpréteur/compilateur<sup>7</sup>
- ▶ ...

---

<sup>7</sup><http://aosabook.org/en/500L/a-python-interpreter-written-in-python.html>

# Concepts avancés

## Parcours ingénieur ECL



Aux 12 unités d'enseignement en sciences pour l'ingénieur et en sciences économiques et humaines du tronc commun s'ajoutent :

★ **Une unité d'enseignement de langues vivantes**

★ **Une unité d'enseignement professionnelle**

Ce socle de base très large permet aux élèves d'acquies l'ouverture, les connaissances et les compétences qui définissent l'ingénieur centralien. Il existe aussi la possibilité d'un double diplôme ingénieur/architecte.

Les objectifs pédagogiques de chaque UE se déclinent en termes de :

- ✓ **Acquisition d'un savoir académique**
- ✓ **Acquisition de connaissances sur le contexte**
- ✓ **Acquisition de compétences de l'ingénieur (problèmes complexes, méthodologies...)**

25% de l'enseignement dans ces UE correspond à un « travail en autonomie » des élèves : travail en groupe ou individuel - obligatoire et contrôlé - où l'initiative de l'élève est clé.

**Transversalité, ouverture, cohérence, nouveauté, langue anglaise, sont les mots-clés qui caractérisent ce semestre.**

\*La dimension des semestres se fait conformément au système LMD, pour faciliter la reconnaissance au niveau international.  
 La notation SS correspond au 1<sup>er</sup> semestre après le Bac, le tronc commun se divise dans les semestres 05, 06, 07.  
 \*\*UE : unité d'enseignement  
 \*\*\*SPU/SEHS : Sciences pour l'ingénieur - Sciences économiques, humaines et sociales.  
 \*\*\*\*Des options de parcours peuvent être faites à l'étranger ou en France

La **Recherche en informatique** s'intéresse aux problèmes ouverts en informatique, mais aussi aux applications et à la dissémination de résultats (monde économique, société, ..)

Les principaux centres de Recherche en France :

<https://www.inria.fr/>



<https://www.cnrs.fr/>



## Concepts avancés

# Défis de Recherche en Informatique

A Lyon :

- ▶ LIRIS (LYON1/INSA/ECL/LYON2)
- ▶ LIP (ENS)
- ▶ CITI (INSA)

A noter que ces laboratoires sont des UMR (Unités Mixtes de Recherche) avec le CNRS et regroupés autour de la FIL (Fédération Informatique Lyon) <https://fil.cnrs.fr/>

## Concepts avancés

# Défis de Recherche en Informatique

A Lyon :

- ▶ LIRIS (LYON1/INSA/ECL/LYON2)
- ▶ LIP (ENS)
- ▶ CITI (INSA)

A noter que ces laboratoires sont des UMR (Unités Mixtes de Recherche) avec le CNRS et regroupés autour de la FIL (Fédération Informatique Lyon) <https://fil.cnrs.fr/>

Laboratoire LIRIS <https://liris.cnrs.fr/>

- ▶ *Laboratoire d'InfoRmatique en Image et Systèmes d'information*
- ▶ Penta-tutelles (CNRS/LYON1/INSA/ECL/LYON2)
- ▶ 347 membres
- ▶ A ECL 24 membres (8 Enseignant/Chercheurs du Département Math/Info)

Equipes **Imagine** (LIRIS) <https://projet.liris.cnrs.fr/imagine/>

- ▶ Analyse d'image (segmentation de région), apprentissage
- ▶ Application à la compréhension de contenu, reconnaissance d'objets, augmentation de contenu.

Equipes **Imagine** (LIRIS) <https://projet.liris.cnrs.fr/Imagine/>

- ▶ Analyse d'image (segmentation de région), apprentissage
- ▶ Application à la compréhension de contenu, reconnaissance d'objets, augmentation de contenu.

Equipes **SICAL** (LIRIS) <https://projet.liris.cnrs.fr/sical>

- ▶ Interaction Homme/Machine, Visualisation (données, réalité virtuelle/augmentée), Adaptation, Serious Games
- ▶ Application aux méthodes de prototypage, amélioration de la collaboration, exploration visuelle de données.

## Concepts avancés Recherche vs Ingénierie



<https://slideplayer.com/slide/9471758/>

### La recherche :

- ▶ Plusieurs types de Recherche: appliquée, fondamentale, pluri-disciplinaire
- ▶ Identification et la définition du problème lui-même..
- ▶ Communication (Articles, conférences, séminaires)
- ▶ Formation (Master Recherche, Doctorale)
- ▶ S'intéresse à des problèmes souvent ouverts et difficiles<sup>8</sup> mais impact<sup>9</sup>

### L'ingénierie

- ▶ Résoudre des contraintes
- ▶ Trouver des solutions et mobiliser des compétences existantes
- ▶ Interagir avec une équipe, communiquer

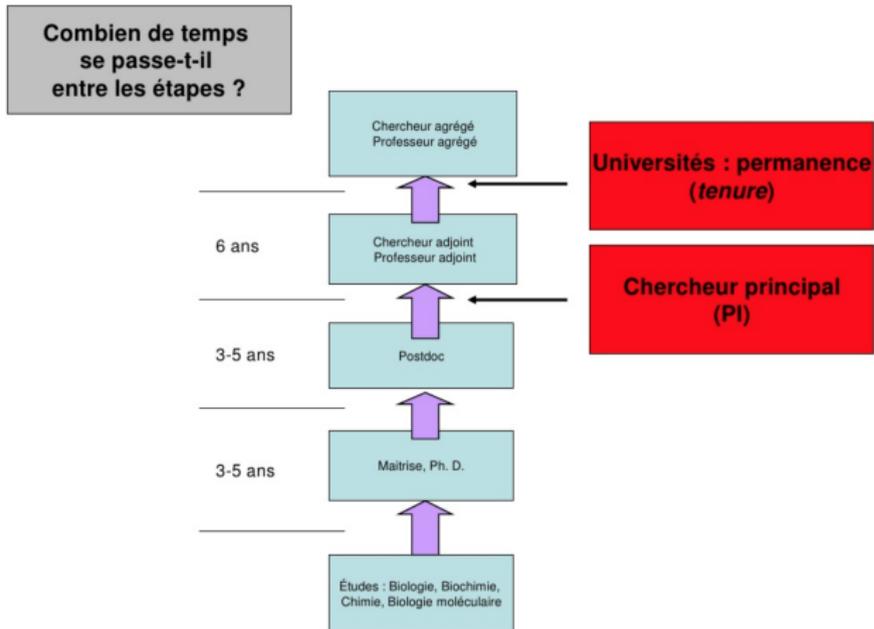
---

<sup>8</sup><http://www.claymath.org/millennium-problems>

<sup>9</sup><https://www.inria.fr/actualite/actualites-inria/palmares-des-prix-inria-2018>

# Concepts avancés Cursus de Recherche

## Devenir chercheur



THE END