Programmation récursive : Diviser pour régner

Benjamin Monmege benjamin.monmege@univ-amu.fr

Récursivité

Définition de suites récurrentes

Exemple de la factorielle:

$$u_0 = 1 \qquad \text{et} \qquad \forall n \geq 1 \quad u_n = n \times u_{n-1}$$

On définit u_n en fonction de u_{n-1}

- Programmation itérative en Python...

```
def factorielle(n: int) -> int:
    assert n >= 0
    resultat = 1
    for i in range(1, n+1):
        resultat *= i
    return resultat
```

Définition de suites récurrentes

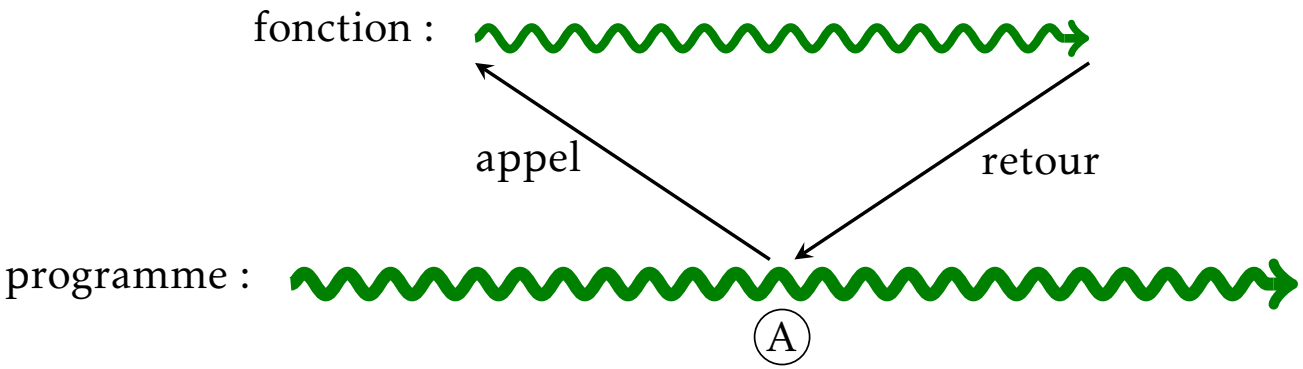
Exemple de la factorielle:

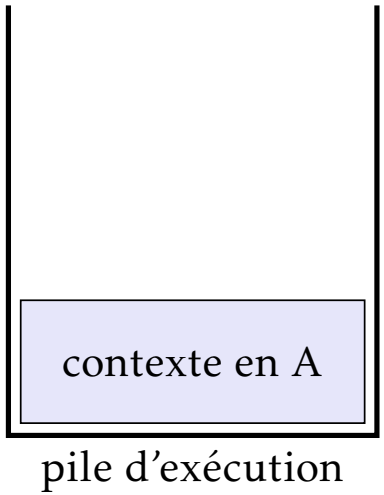
$$u_0 = 1 \qquad \text{et} \qquad \forall n \geq 1 \quad u_n = n \times u_{n-1}$$

- Plus naturel en récursif

```
def factorielle(n: int) -> int:
    assert n >= 0
    if n == 0:
        return 1
    return n * factorielle(n - 1)
```

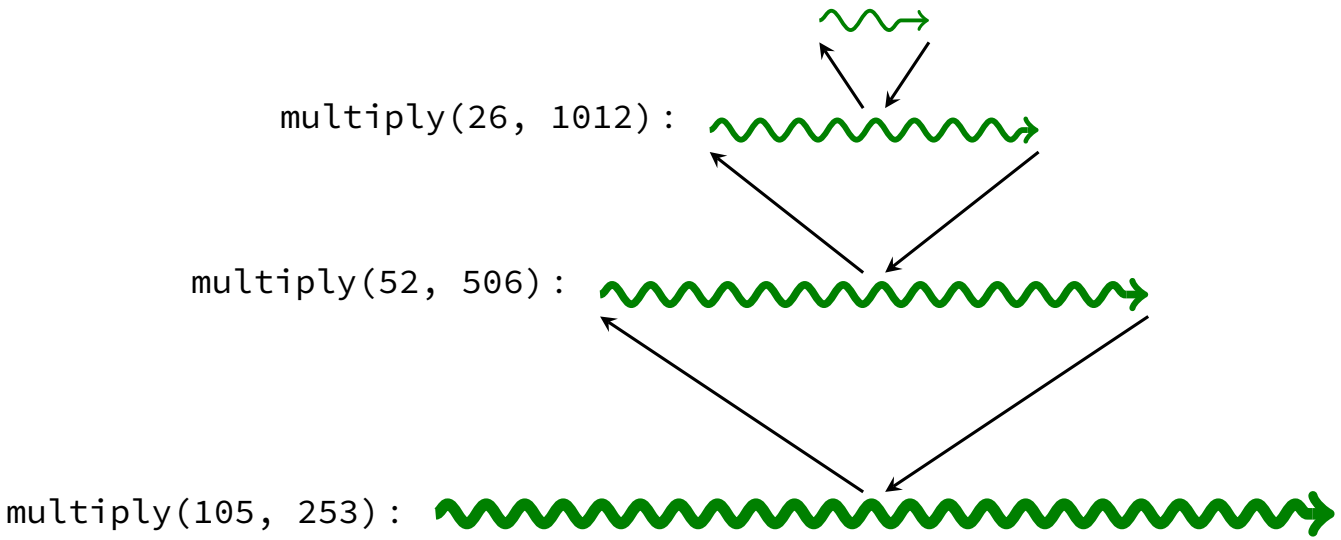
Pile d'exécution d'un programme récursif

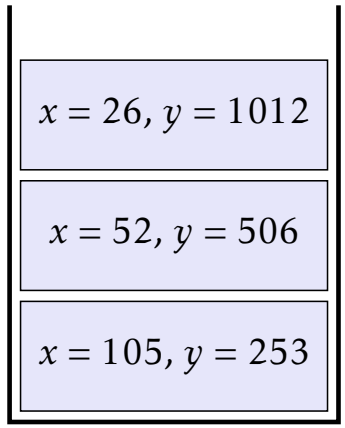




Pile d'exécution d'un programme récursif : exemple

```
def multiply(x, y):
    if x <= 0:
        return 0
    elif x % 2 == 0:
        return multiply(x//2, y+y)
    else:
        return multiply(x//2, y+y) + y
```





pile d'exécution

- Par défaut, pile de récursivité limitée à 1000 appels en Python !

Diviser pour régner

Des fonctions récursives particulières

Résolution d'un problème en trois étapes :

- **Diviser** : découper un problème initial en sous-problèmes
- **Régner** : résoudre les sous-problèmes (récursivement ou directement s'ils sont assez petits)
- **Combiner** : calculer une solution au problème initial à partir des solutions des sous-problèmes

Exemple 1 : Recherche dichotomique dans un tableau trié

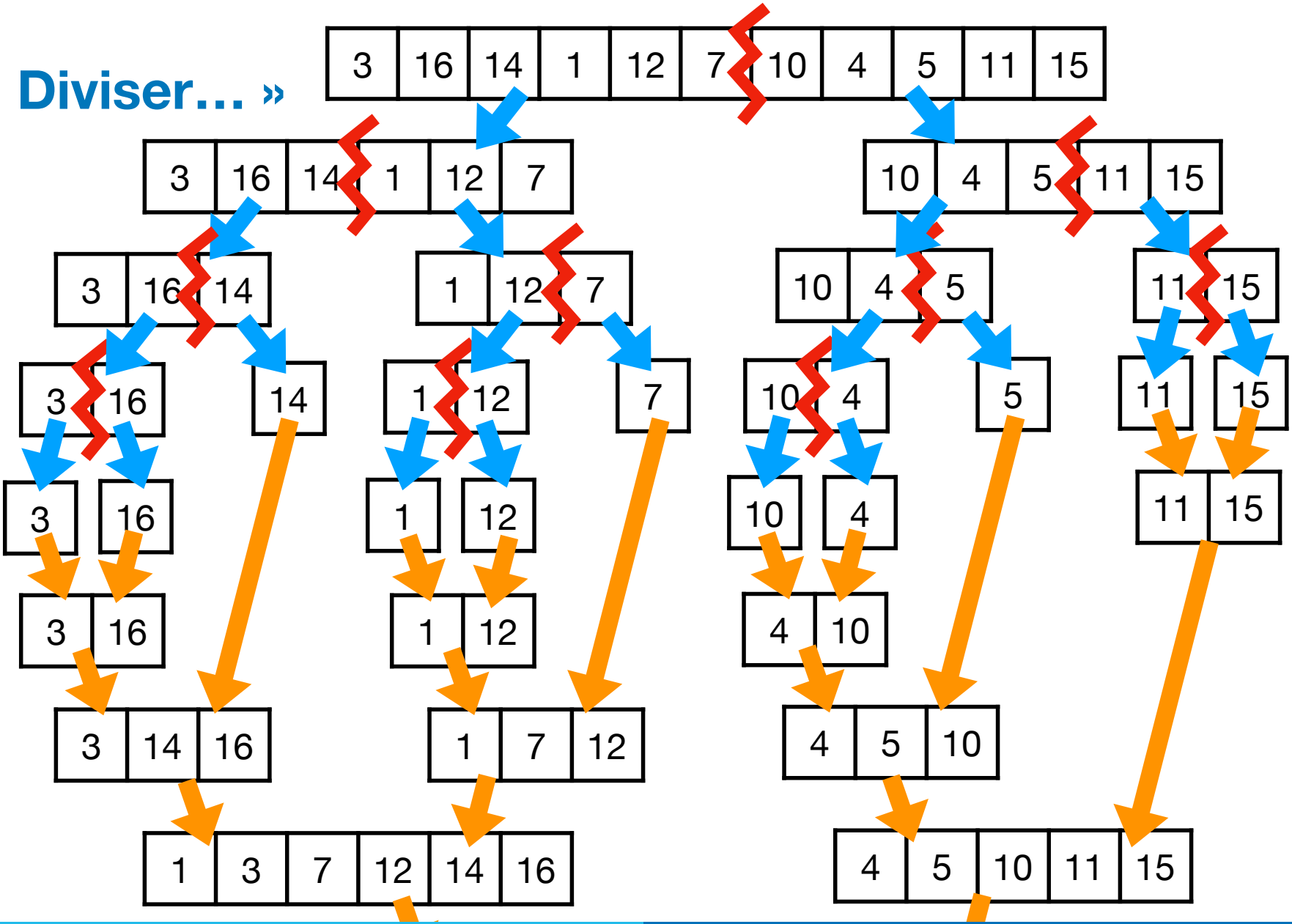
- **Diviser** : comparer l'élément recherché `e` à celui `x` au milieu du tableau pour savoir s'il faut chercher à gauche ou à droite
- **Régner** : conclure si `e`=`x`, ou rechercher récursivement dans l'un des deux sous-tableaux
- **Combiner** : rien à faire...

Exemple 2 : Tri par fusion

- **Diviser** : diviser le tableau en deux sous-tableaux de longueurs presque égales
- **Régner** : trier récursivement chacun des deux sous-tableaux
- **Combiner** : fusionner les deux sous-tableaux triés en temps linéaire

Exemple 2 : Tri par fusion

« Diviser... »



Complexité des algorithmes diviser pour régner

Si n est un paramètre de taille de l'entrée, la complexité d'un algorithme diviser pour régner est toujours de la forme

$$T(n) = \underbrace{T_d(n)}_{\text{division}} + \sum_{\text{sous-problèmes de taille } m<n} T(m) + \underbrace{T_c(n)}_{\text{combinaison}}$$

pour n suffisamment grand (sinon c'est le cas de base)

- Recherche dichotomique : $T(n) = \mathcal{O}(1) + T(\lceil n/2 \rceil) + 0$
- Tri par fusion : $T(n) = \mathcal{O}(n) + T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \mathcal{O}(n)$

Master theorem

Calculer $T(n)$ par substitutions... ou utiliser le *Master theorem*

Théorème

Soient $a \geq 1$ et $b > 1$, f une fonction et T définie par

$$T(n) = a \cdot T(n/b) + f(n)$$

- Si $f(n) = O(n^{\log_b a - \epsilon})$ pour un $\epsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$
- Si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

Exemple

- Recherche dichotomique : $a = 1$, $b = 2$, $f(n) = \Theta(1)$
  - 2ème cas, $\Theta(\log n)$
- Tri par fusion : $a = 2$, $b = 2$, $f(n) = \Theta(n)$
  - 2ème cas, $\Theta(n \log n)$

Tri rapide

Pour trier le tableau $T[g, \ldots, d]$:

- **Diviser** : choisir un pivot dans $T[g, \ldots, d]$ et partitionner
- **Régner** : trier (en place) récursivement chacun des deux sous-tableaux
- **Combiner** : rien à faire !

Complexité :

$$T(n) \leq \mathcal{O}(n) + \max_{0<k<n} T(k) + T(n-k-1) = \mathcal{O}(n) + T(n-1)$$

$$\Rightarrow \qquad T(n) = \mathcal{O}(n^2)$$

Un meilleur tri rapide

- Choix du pivot de sorte qu'on divise le tableau en deux portions de taille entre $n/4$ et $3n/4$...

Complexité : $T(n) \leq \mathcal{O}(n) + \max_{n/4 \leq k \leq 3n/4} [T(k) + T(n-k-1)]$

Calcul par intuition/récurrence... $T(n) = \mathcal{O}(n \log n)$

Comment faire ce choix de pivot ?

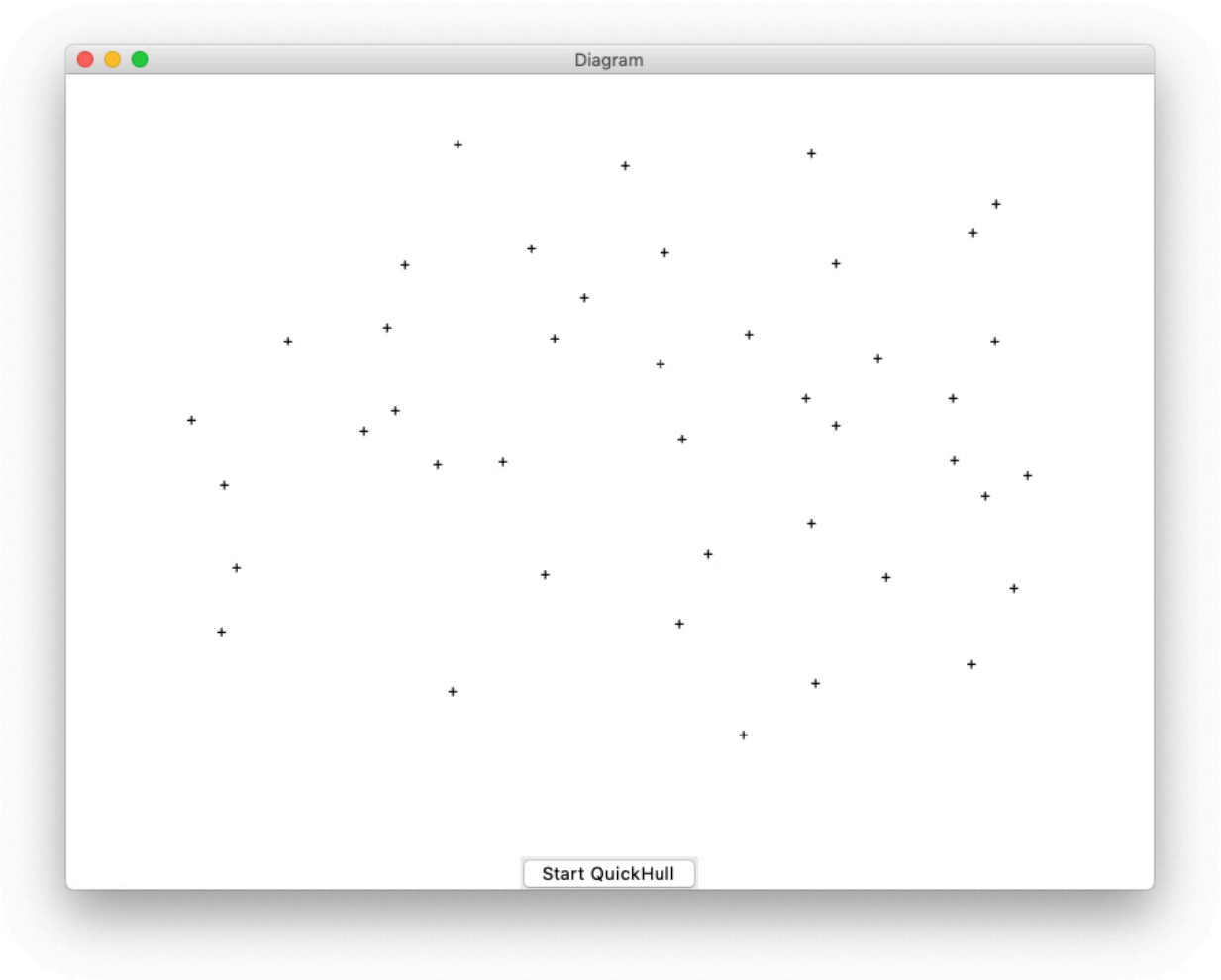
- Choix aléatoire parmi les n éléments du tableau
- Probabilité que le choix soit un *bon pivot* : $n/2$
- Si choix aléatoire non convenable... *on recommence !*
- Nombre moyen de tentatives nécessaires : 2

###Théorème

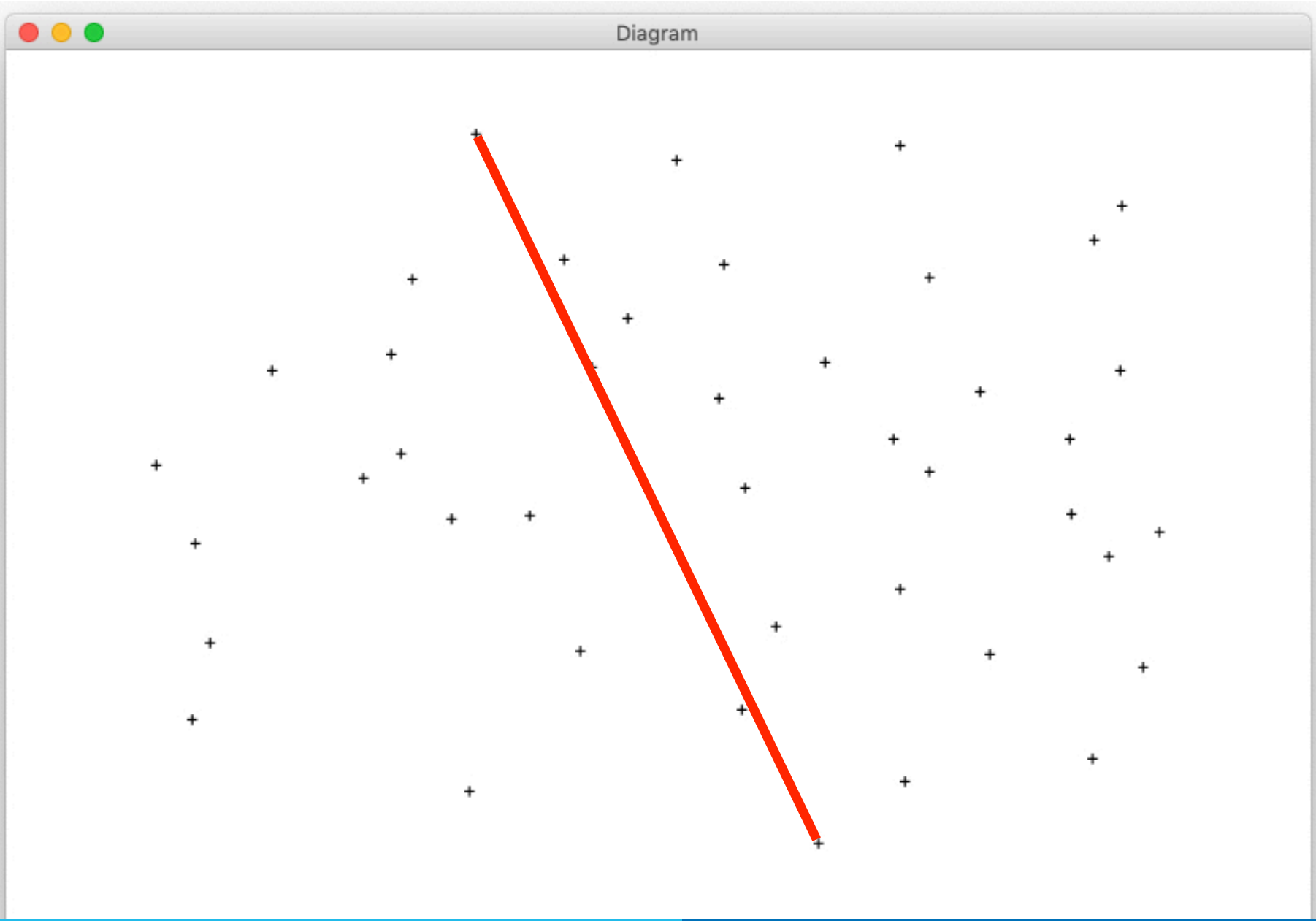
- Le tri rapide avec choix aléatoire répété du pivot pour obtenir des portions de taille entre $n/4$ et $3n/4$ est en *complexité moyenne* $\mathcal{O}(n \log n)$.
- En fait, on peut aussi montrer que si on commence par faire une permutation aléatoire du tableau en entrée, le tri rapide (non modifié) est de *complexité moyenne* $\mathcal{O}(n \log n)$.

Application au calcul de l'enveloppe convexe d'un ensemble de points du plan

Enveloppe convexe : plus petit polygone qui contient tous les points



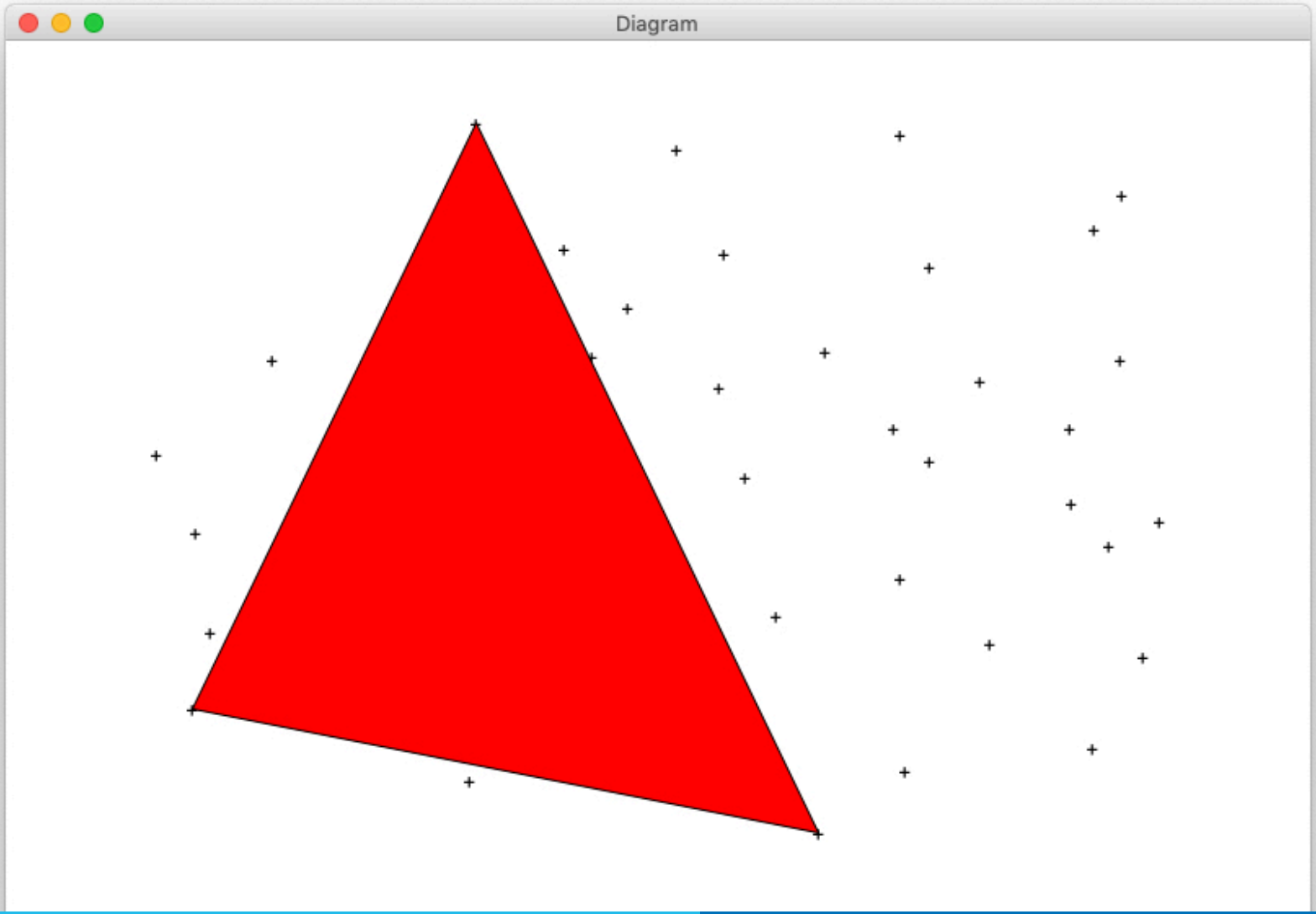
Trouver un “pivot” : les deux sommets extrêmes



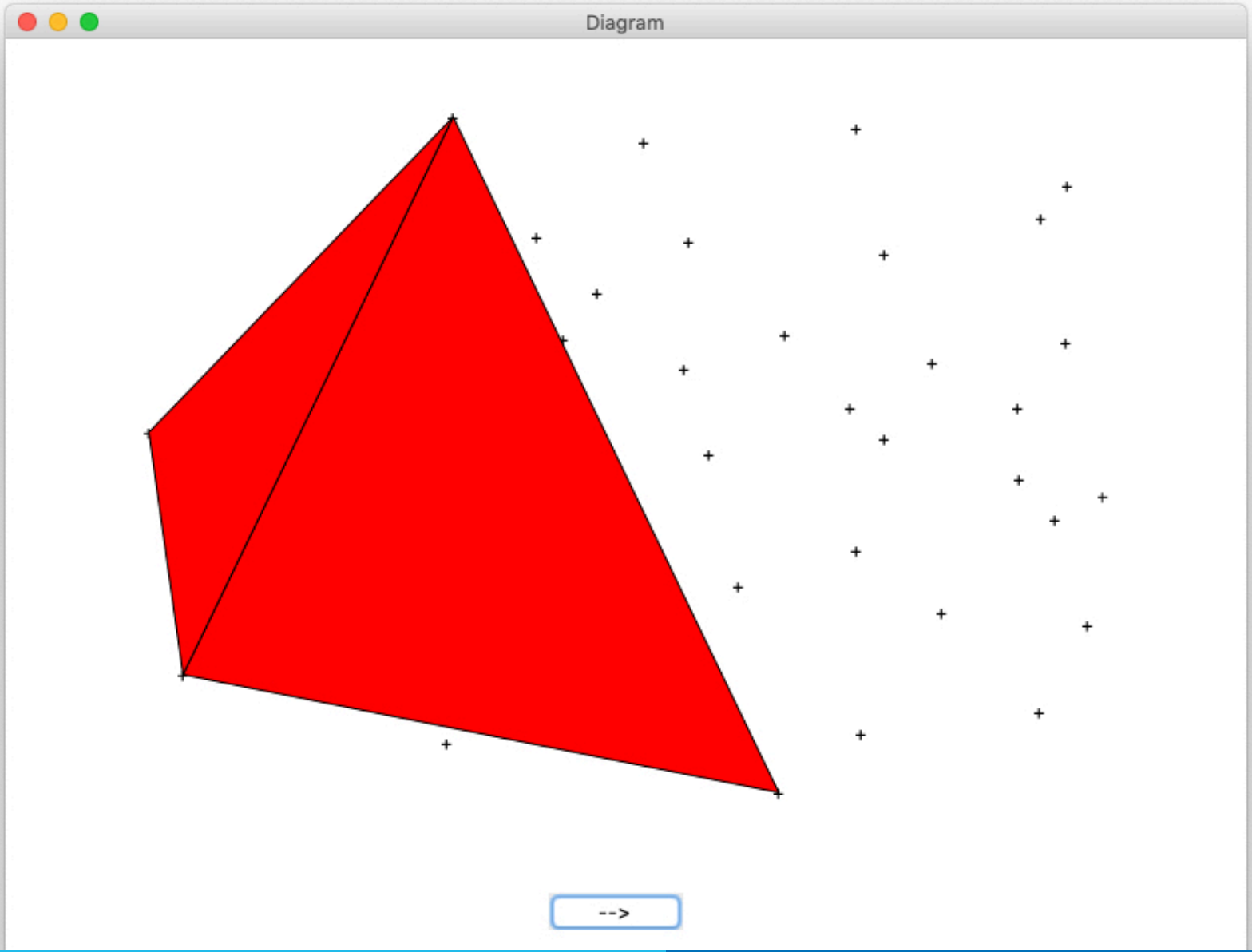
Trouver un “pivot” : les deux sommets extrêmes

- **Diviser** : on se retrouve à devoir trouver l'enveloppe convexe des points *à gauche* et *à droite*
- **Régner** : calcul en temps linéaire du point les plus éloignés pour trouver les prochains découpages
- **Combiner** : mettre bout à bout les différents segments du polygone convexe

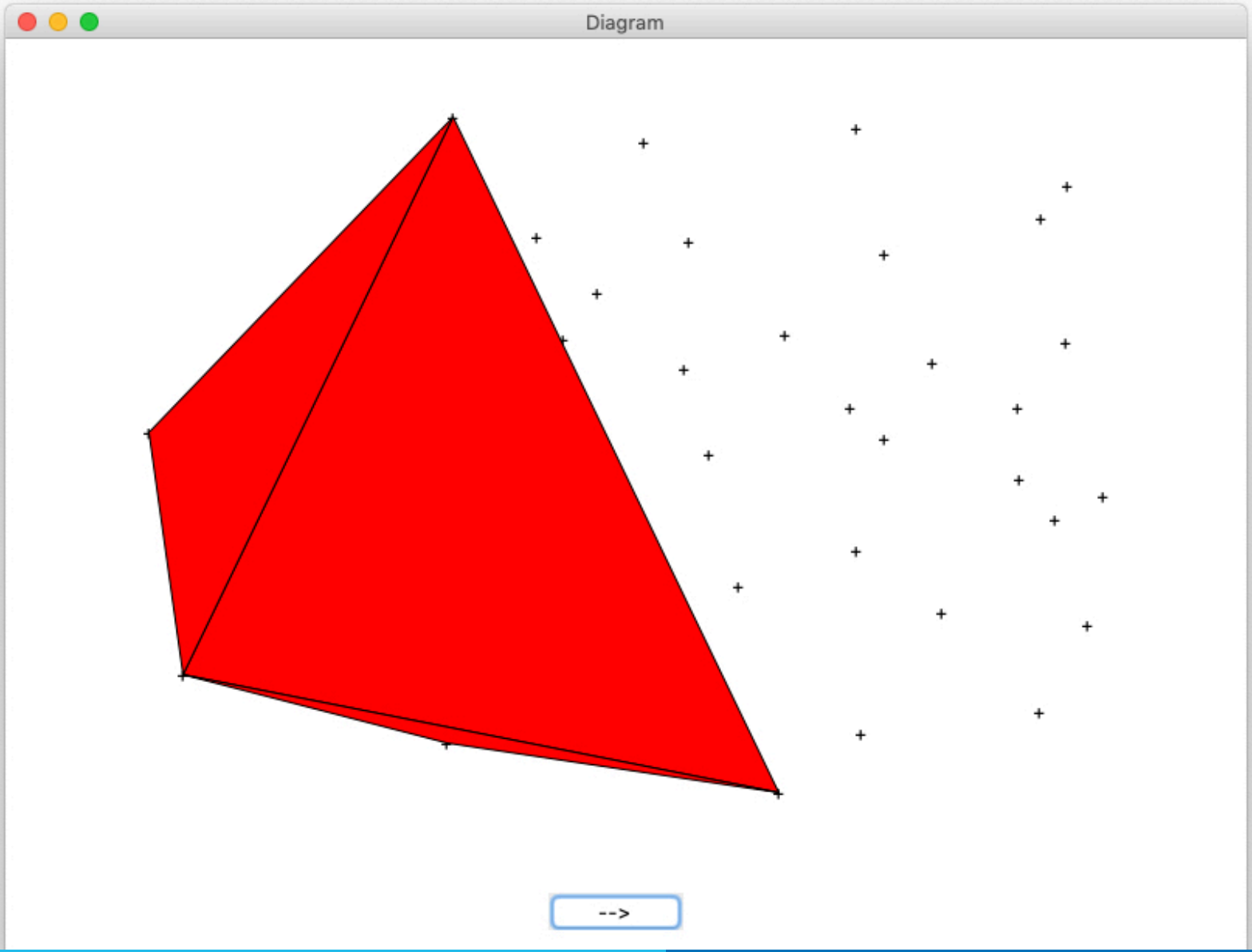
À gauche : Calcul en temps linéaire du point le plus éloigné pour trouver le prochain découpage...



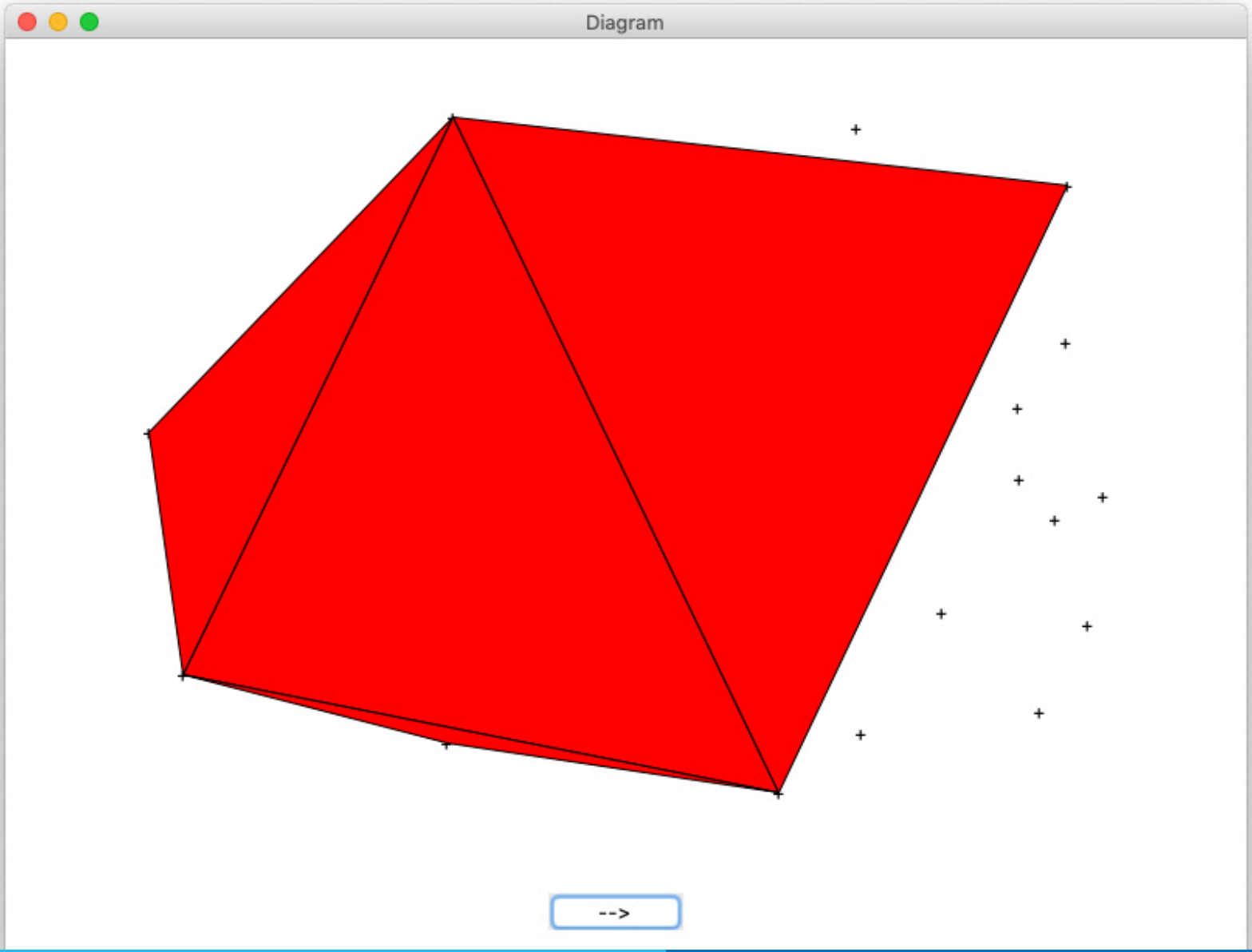
Et ainsi de suite. . .



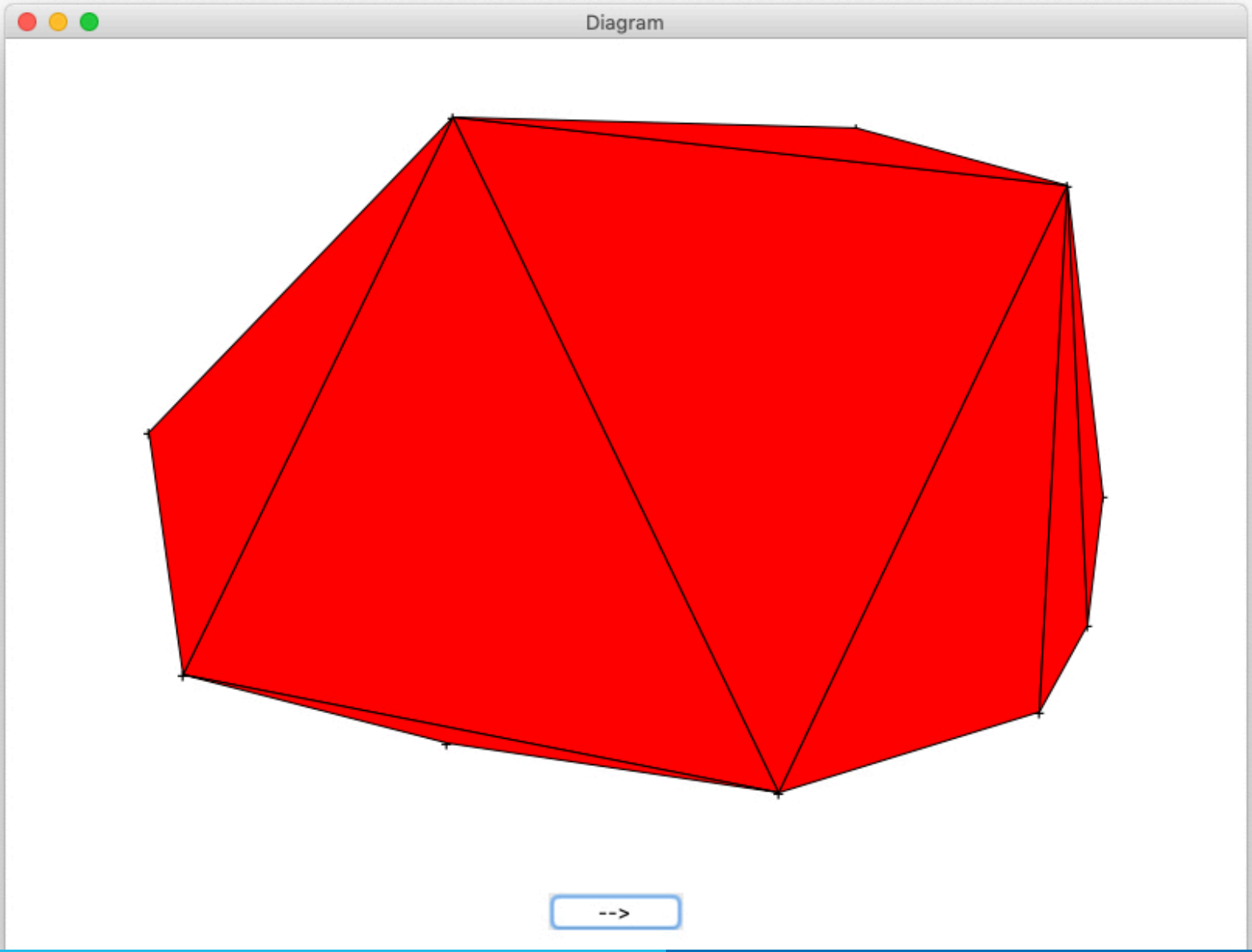
Et ainsi de suite...



Même chose à droite. . .



Jusqu'à avoir construit l'enveloppe convexe

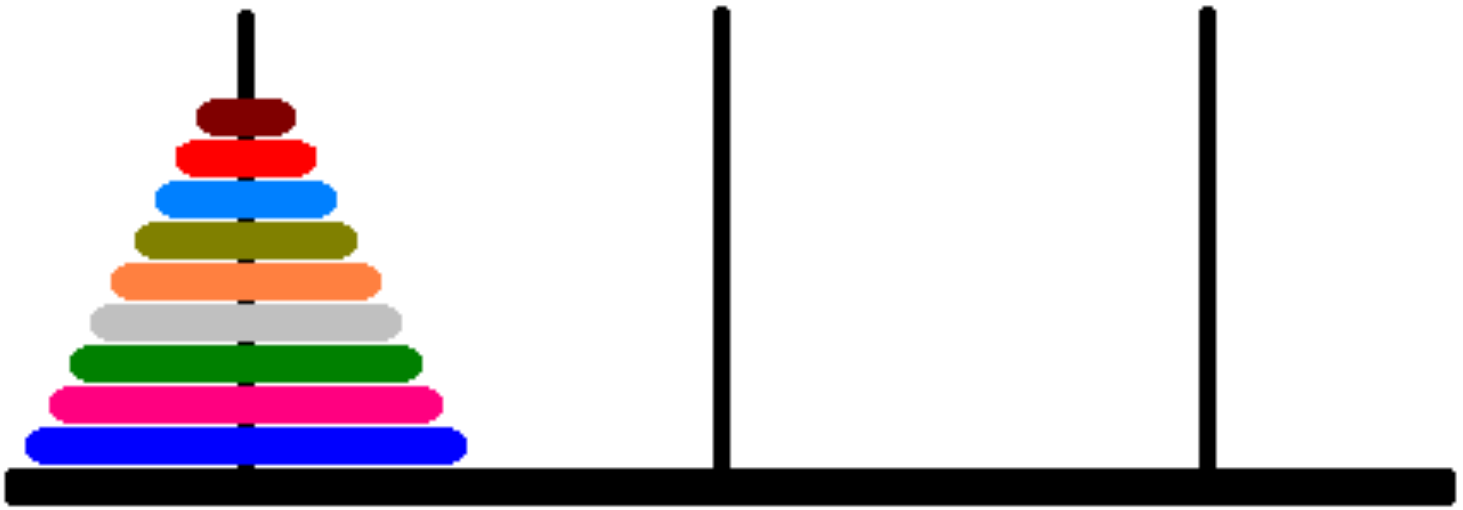


Complexité

- Même analyse que le **tri rapide** :
  - $\mathcal{O}(n^2)$ dans le pire des cas
  - $\mathcal{O}(n \log n)$ en moyenne

Récursivité pour la résolution de jeux

Jeu des tours de Hanoi



- Déplacer les disques un par un de la tour de gauche à celle de droite en ne plaçant jamais un disque sur un disque de diamètre inférieur
- But : trouver une stratégie !

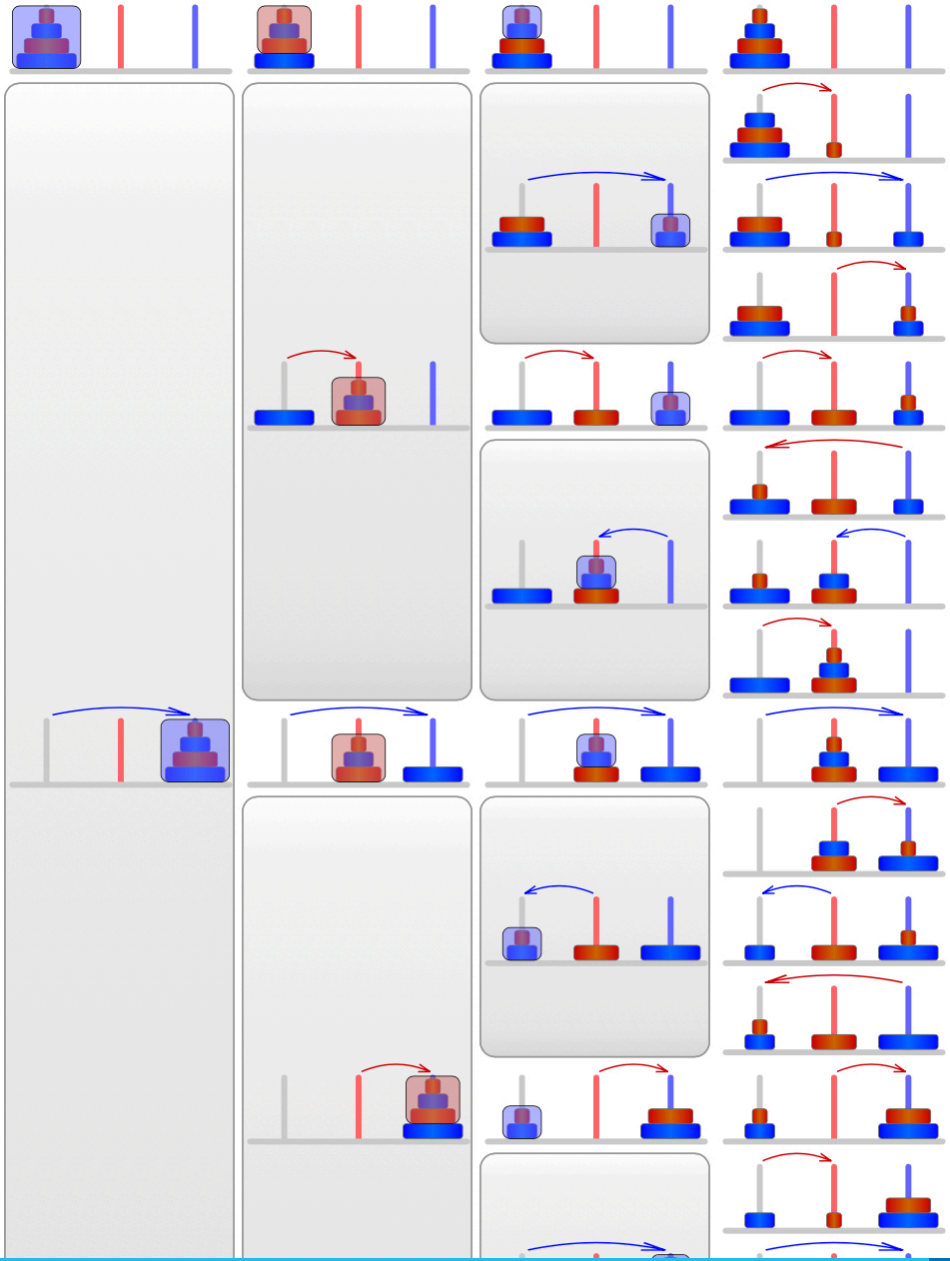
Solution récursive d'inspiration *diviser pour régner*

- Divisons la stratégie pour n disques en trois partie :
  - transporter les $n-1$ plus petits disques vers la tour du milieu
  - transporter le plus grand disque sur la tour de droite
  - transporter les $n-1$ plus petits disques vers la tour de droite

Le problème a donc une solution récursive, si on s'autorise d'échanger

- la tour de départ
- la tour d'arrivée
- la tour intermédiaire

Solution imagée (crédits: wikipédia)



Nombre d'étapes nécessaires pour résoudre le problème à n disques

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ T(n-1) + 1 + T(n-1) = 1 + 2T(n-1) & \text{sinon} \end{cases}$$

$$\Rightarrow T(n) = 2^n - 1$$

On peut démontrer que c'est le **nombre minimal** de mouvements nécessaires pour gagner... On a donc une **stratégie optimale**.

Solution en Python

```
def hanoi(n, depart, intermediaire, arrivee):
    assert n>0
    if n == 1:
        print(depart, " -> ", arrivee)
    else:
        hanoi(n-1, depart, arrivee, intermediaire)
        print(depart, " -> ", arrivee)
        hanoi(n-1, intermediaire, depart, arrivee)
```