

## Exercice 1:

### Algo 1:

↳ Ligne 3 exécutée  $j$  fois avec  $j \leq n-1$

$$L3 + L4 \rightsquigarrow O(n)$$

$$L2 \text{ boucle } n \text{ fois, } L2 + L3 + L4 \rightsquigarrow O(n^2)$$

$$L1 \text{ boucle } n \text{ fois, } L1 + L2 + L3 + L4 \rightsquigarrow O(n^3)$$

$$L5 + L6 \rightsquigarrow O(n)$$

$$\text{Au final, } O(n^3) + \cancel{O(n)} = O(n^3)$$

La boucle de 0 à  $j$  fera :

1 itération, puis 2, puis ...  $n-1$  itérations

Formule de la somme, on fait au max  $\simeq \frac{n(n+1)}{2} \simeq \frac{n^2}{2}$

La constante  $\frac{1}{2}$  disparaît avec big  $O$

On a au final  $O(n)$  pour cette boucle

### Algo 2 :

Pour appliquer le **Master Theorem**  
On veut une récursion qui renvoie une **fraction** de la valeur

Ici on a pas de fraction en entrée, pas de **MT**

On note  $c$  la somme du temps des opérations élémentaires et  $t(n)$  la complexité de l'algorithme sur l'entrée  $n$

On a :

$$t(n) \leq 2t(n-1) + c$$

$$t(n-1) = 2t(n-2) + c$$

$$\text{remplace} \leq 2(2t(n-2)+c)+c$$

$$\begin{aligned} \text{simplifie} &\leq 4t(n-2)+3c \\ &\leq 4t(n-2)+3c \end{aligned}$$

$$\text{remplace} \leq 4(2t(n-3)+c)+3c$$

$$\text{simplifie} \leq 8t(n-3)+7c$$

$$\leq \dots$$

$$\leq 2^i t(n-i) + (2^i - 1)c$$

Montrons par récurrence qu'après  $i$  appels récurifs, on retrouve ça.

Au rang 0,  $i=0$  et on a:

$$\begin{aligned} t(n) &\leq 1 \times t(n-0) + (1-1)c \\ &\leq 1 \times t(n) \\ &\leq t(n) \end{aligned}$$

Vrai au rang 0

Si on suppose vrai au rang  $i$ , on a:

$$\begin{aligned} t(n) &\leq 2^i \times t(n-i) + (2^i - 1)c \\ &\leq 2^i \times (2t(n-i-1)+c) + (2^i - 1)c \\ &\leq 2^i \times 2 \times t(n-(i+1)) + (2 \times 2^i - 1)c \\ &\leq 2^{i+1} \times t(n-(i+1)) + (2^{i+1} - 1)c \end{aligned}$$

tout les  $i$  deviennent  $(i+1)$

Vrai au rang  $i+1$

Après  $n$  appels récursifs, on a:

$$t(n) \leq 2^n \times t(n-n) + (2^n - 1)c$$

$$\leq \underline{2^n \times t(0)} + \underline{2^n c - c}$$

$$\leq 2^n (\underbrace{t(0) + c}_{\text{Constante } C_1}) - c$$

Constante  $C_1$

$$\leq 2^n \times C_1$$

Donc  $t(n) = O(2^n)$

Algo 3: On note  $c$  les opérations élémentaires  
 $t(n)$  complexité proportionnelle aux  
nombres de **while**

L'algo fait  $k$  boucles avec  $\frac{n}{3^k} \leq 1$

En prenant  $\log$  base 2, qui est **croissant**  
(on cherche à faire tomber le  $k$  pour trouver  
une égalité). On a:

$$\log\left(\frac{n}{3^k}\right) \leq \log(1)$$

$$\log n - \log(3^k) \leq 0$$

$$\log n \leq \log(3^k)$$

$$\log n \leq k \log(3)$$

$$\frac{\log n}{\log(3)} \leq k$$

Donc on a:  $t(n) = O(\log n)$

Algo 4: On fait  $n$  fois  $t(n-1)$

$$t(n) \leq n \times t(n-1) \quad \text{le terme précédent}$$

$$\text{remplace} \leq n \left( (n-1) \times t(n-2) \right)$$

$$\text{remplace} \leq n \left( (n-1) \times ((n-2) \times t(n-3)) \right)$$

$$\leq \dots$$

$$\leq n(n-1)(n-2)(n-3) \dots \times 2 \times t(1)$$

$n!$  factorielle

Donc  $t(n) = O(n!)$

Algo 5: On fait  $n \rightarrow n-3$ ,  $k$  fois jusqu'à ce que  $n \leq 0$ , donc:

$$n - 3k < 0 \quad \leftarrow \text{modélisation de l'algorithme}$$

$$n < 3k$$

$$\frac{n}{3} < k$$

Etant donné que  $\frac{n}{3} = \frac{1}{3} \times n$ , avec big O, la constante disparaît. Au final,  $t(n) = O(n)$

Algo 6: On a une Fraction dans la récurrence: **Master Theorem**

Formule:

$$t(n) \leq a \cdot t\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

↑ nombre d'appels récurifs  
 ↑ nombre par lequel l'entrée est divisée  
 ↓ Temps en dehors des récurions ( $< op\_elem >$ )

$$a = 1, b = 2, d = 0$$

$$b^d = 2^0 = 1 \text{ donc } t(n) = O(\log n)$$

Algo 7:  $a = 2, b = 2, d = 1$

↑  
2 récurions

↑ Complexité  $O(n)$   
en dehors  
des récurions

$$b^d = 2 = a$$

$$T(n) = O(n \log n)$$

Donc on a:  $t(n) = (n \log n)$

Algo 8:  $a=2, b=3, d=1$

$b^d = 2 < a$  Donc on a:  $t(n) = O(n^1)$   
 $= O(n)$

Algo 9:  $a=3, b=2, d=1$  ← la dernière boucle se fait  $\frac{n}{15}$  fois.  
 $b^d = 2 < a$

Donc on a:  $t(n) = O\left(n^{\frac{\log 3}{\log 2}}\right) = O(n^{\log 3})$

## Exercice 2:

① a) On a 6 chances sur 36  $((1,1), (2,2), \dots, (6,6))$

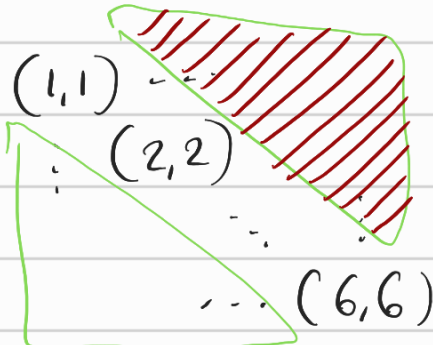
Autre méthode: On note  $x$  la valeur du 1<sup>er</sup> dé et  $y$  du second dé.

$$\begin{aligned} P[X=Y] &= \sum_{t=1}^6 P[X=t \wedge Y=t] \\ &= \sum_{t=1}^6 P[X=t] \times P[Y=t] \end{aligned}$$

Comme  $\forall t=1 \dots 6, P[X=t] = \frac{1}{6}$

et  $P[Y=t] = \frac{1}{6}$

Donc  $P[X=Y] = \sum_{t=1}^6 \frac{1}{6} \times \frac{1}{6} = \frac{1}{6} = \frac{1}{6}$

⑥  ← On cherche ces probabilités  
soit :  $36 - 6$  ←  $(1,1), (2,2), \dots, (6,6)$   
           $\uparrow 2$   
          la moitié

Méthode rigoureuse :

$$P_r[X < Y] = \sum_{t=1}^6 P_r[X=t \wedge Y>t]$$

$$= \sum_{t=1}^6 P_r[X=t] \times P_r[Y>t]$$

Comme  $\forall t=1 \dots 6, P_r[X=t] = \frac{1}{6}$

et  $P_r[Y>t] = \frac{6-t}{6}$

Donc  $P_r[X < Y] = \sum_{t=1}^6 \frac{1}{6} \times \left(\frac{6-t}{6}\right)$

$$= \frac{1}{36} \sum_{t=1}^6 (6-t)$$

On met sous le même dénominateur ici 36

$$= \frac{1}{36} \times (5+4+3+2+1)$$

$$= \frac{5}{12} = \frac{15}{36}$$

- ⑦
- ① "Le premier dé est pair"
  - ② "Le premier dé est impair"
  - ③ "La somme des deux dés est paire"



$(A, B)$  partitionne l'univers. Par la formule des probabilités totales:

$$\begin{aligned}
 P_r[S] &= P_r[S|A] \times P_r[A] + P_r[S|B] \times P_r[B] \\
 &= \underbrace{P_r[\text{"2<sup>nd</sup> dé pair"}]}_{\text{Proba somme pair avec 1<sup>er</sup> dé pair}} \times \underbrace{P_r[A]}_{\text{Proba somme pair avec 2<sup>nd</sup> dé impair}} + \underbrace{P_r[\text{"2<sup>nd</sup> dé impair"}]}_{\text{Proba somme pair avec 2<sup>nd</sup> dé impair}} \times \underbrace{P_r[B]}_{\text{Proba somme pair avec 2<sup>nd</sup> dé impair}} \\
 &= \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} \\
 &= \frac{1}{2}
 \end{aligned}$$

② @ Intuitivement, 10 tirages avec  $\frac{1}{2}$  chance,  $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \dots = \frac{1}{2^{10}}$  (car tirages indépendants)

Autre notation:  $P_r[(X_1=P) \wedge (X_2=P) \wedge \dots \wedge (X_{10}=P)]$

$$\begin{aligned}
 &= \left(\frac{1}{2}\right)^{10} \\
 &= \prod_{i=1}^{10} P_r[X_i=P]
 \end{aligned}$$

⑥ (A) "Obtenir au moins un Pile"  
 $\neg A$  "Obtenir que des faces"

Parfois l'opposé est plus facile à calculer

Par la question précédente:

$$P_r[\neg A] = \frac{1}{2^{10}} = \frac{1}{1024}$$

$$\text{Donc } P_r[A] = 1 - P_r[\neg A] = \frac{1023}{1024}$$



③  $C_{10}^5 \left(\frac{1}{2}\right)^5 \left(\frac{1}{2}\right)^5$  pas plus d'explications

### Exercice 3:

① Chaque valeur  $v$  a une chance  $\frac{1}{k}$  de sortir.

$$E[X] = \sum_{v=1}^k P_r[X=v] \times v$$

$$= \sum_{v=1}^k \frac{1}{k} \times v = \frac{1}{k} \sum_{v=1}^k v = \frac{1}{k} \times \left( \cancel{k} \times \frac{k+1}{2} \right)$$

$$= \frac{k+1}{2}$$

②  $E[X] = \sum_{p=0}^{10} P_r[X=p] \times p$

Probabilité d'avoir exactement  $p$  piles

On découpe en 10 tirages:

$$X_i = \begin{cases} 1 & \text{si le } i^{\text{ème}} \text{ tirage est pile} \\ 0 & \text{sinon} \end{cases}$$

$$X = X_1 + X_2 + \dots + X_{10}$$

Par linéarité de l'espérance

$$E[X] = E[X_1 + X_2 + \dots + X_{10}] = E[X_1] + \dots + E[X_{10}]$$

$$= \sum_{i=1}^{10} E[X_i] = \sum_{i=1}^{10} \left( 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} \right) = 10 \cdot \frac{1}{2} = 5$$

## Exercice 4:

① AlgoNaïf:

tant que  $i < n$ , faire:

    si  $T[i] = x$   
        retourner  $i$

↙ Pas 100% correct, il faut juste comprendre comment il fonctionne.

$$T(n) = O(n)$$

② ①  $\frac{1}{n}$  car  $X$  suit une loi géométrique de paramètre  $p$ ,  $P_r[X=i] = p(1-p)^{i-1}$ .  
On peut calculer directement son espérance

Formule de l'espérance totale :

$$\begin{aligned} E[X] &= E[X | \text{"x trouvé"}] \times P_r[\text{"x trouvé"}] + E[X | \text{"x pas trouvé"}] \times P_r[\text{"x pas trouvé"}] \\ &= 1 \times \frac{1}{n} + E[X | \text{"x pas trouvé"}] \times \frac{n-1}{n} \end{aligned}$$

Mais  $E[X | \text{"x pas trouvé"}] = 1 + E[X]$  ← même espérance que précédemment et +1 pour le premier tirage

$$\text{du coup, } E[X] = \frac{1}{n} + (1 + E[X]) \times \left(\frac{n-1}{n}\right)$$

$$\frac{1}{n} E[X] = \frac{1}{n} + \frac{n-1}{n} = 1 \rightarrow E[X] = n$$

