

# COMPLEX

## Cours 6 - Introduction aux algorithmes probabilistes

Damien Vergnaud

Sorbonne Université – CNRS



# Table des matières

## 1 Introduction

- Historique (très) incomplet
- Principe général
- Algorithmes déterministes
- Algorithmes probabilistes

## 2 Arbre binaire ET-OU

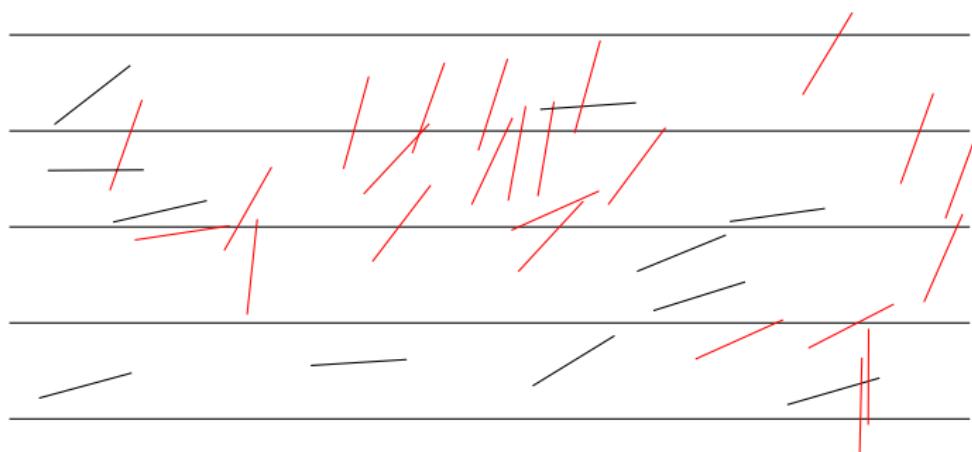
- Algorithmes déterministes
- Algorithmes probabiliistes

## 3 Tri et sélection rapides

- Tri rapide
- Sélection rapide
- Médian approché

# Aiguille de Buffon

- 1733 - Georges-Louis Leclerc de Buffon
- $\frac{2}{\pi} = 0.6366197724$



$$\frac{21}{36} = 0.685$$

# Aiguille de Buffon

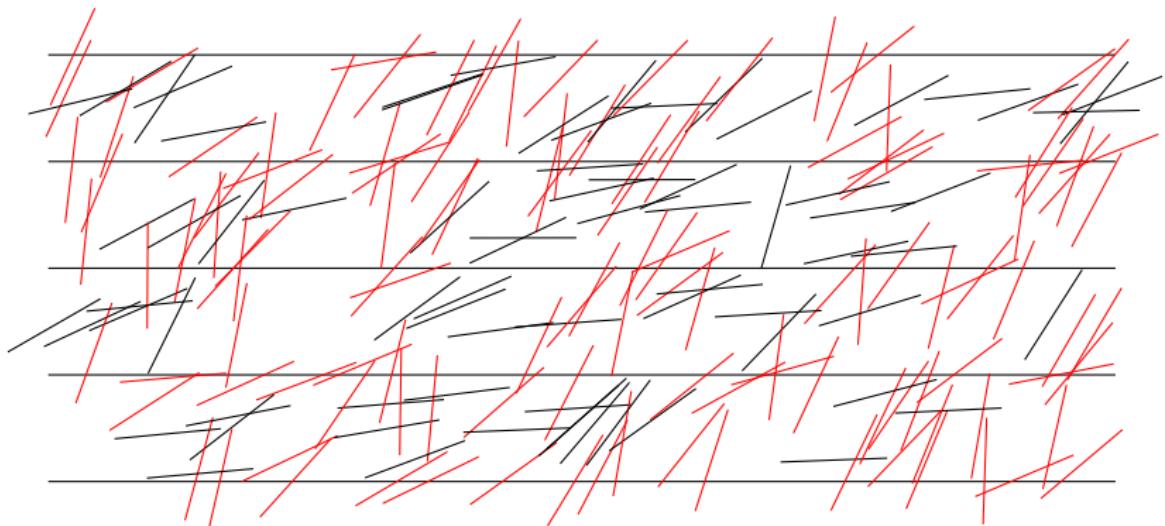
- 1733 - Georges-Louis Leclerc de Buffon
- $\frac{2}{\pi} = 0.6366197724$



$$\frac{39}{60} = 0.65$$

# Aiguille de Buffon

- 1733 - Georges-Louis Leclerc de Buffon
- $\frac{2}{\pi} = 0.6366197724$



$$\frac{127}{200} = 0.635$$

# Racine carrée modulaire

## RACINE CARRÉE MODULAIRE

- ENTRÉE :  $n \geq 2$  un entier et  $a$  un entier premier avec  $n$
- SORTIE :  $b$  entier tel que  $a \equiv b^2 \pmod{n}$  (ou NON\_CARRÉ)

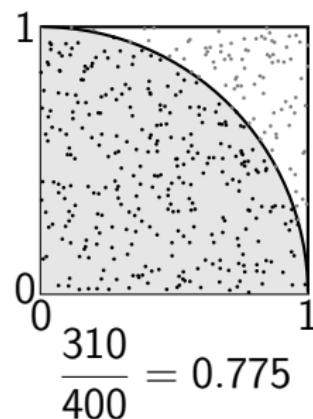
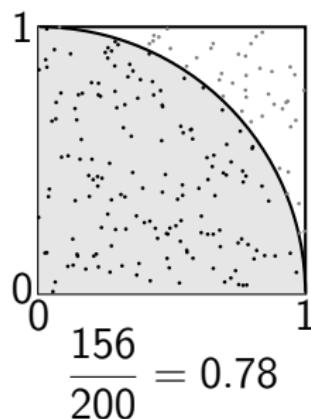
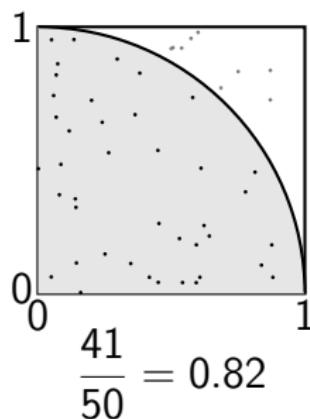
- 1917 – H. C. Pocklington
- Algorithme **probabiliste** efficace pour  $n = p$  un nombre premier
- **Problème ouvert** : Algorithme déterministe polynomial ?

Pour le Cours 8 ...

Faire des révisions d'arithmétique  
(cf. notes de cours indiquées sur le moodle du cours)

# Méthodes de Monte-Carlo

- Calculer une valeur numérique approchée en utilisant des procédés aléatoires
- 1947 – N. Metropolis
- 1949 – S. Ulam et N. Metropolis



# Voisins les plus proches

## VOISINS LES PLUS PROCHES

- ENTRÉE :  $n \geq 2$  un entier et  $(P_1, \dots, P_n)$   $n$  points de  $\mathbb{R}^2$
  - SORTIE :  $(i, j) \in \{1, \dots, n\}$  avec  $i \neq j$  tel que  $d(P_i, P_j)$  minimale
- 
- 1976 – M. O. Rabin
  - Algorithme probabiliste en temps  $O(n)$  !
  - Le dernier à avoir « découvert » les algorithmes probabilistes !

<https://rjlipton.wordpress.com/2009/03/01/rabin-flips-a-coin/>

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes déterministes et probabilistes

- **Algorithme déterministe :**

- termine toujours
- efficace dans **tous** les cas
- retourne **toujours** la bonne réponse

- **Algorithme probabiliste :**

- termine toujours
- utilise de l'aléa dans son exécution
- efficace dans **la plupart** des cas
- retourne
  - **souvent** la bonne réponse
  - **toujours** une réponse **souvent** proche de la bonne

# Algorithmes de type Las Vegas / Monte-Carlo

## ● Las Vegas

- termine toujours
- utilise de l'aléa dans son exécution
- temps d'exécution = **variable aléatoire**
- retourne **toujours** la bonne réponse

## ● Monte Carlo

- termine toujours
- utilise de l'aléa dans son exécution
- temps d'exécution connu *a priori*
- retourne **souvent** la bonne réponse

# Algorithmes de type Las Vegas / Monte-Carlo

## • Las Vegas

- termine toujours
- utilise de l'aléa dans son exécution
- temps d'exécution = variable aléatoire
- retourne toujours la bonne réponse

## • Monte Carlo

- termine toujours
- utilise de l'aléa dans son exécution
- temps d'exécution connu *a priori*
- retourne souvent la bonne réponse

# Élément majoritaire

## ÉLÉMENT MAJORITAIRE

- ENTRÉE :  $n \geq 2$  un entier et  $T$  un tableau de  $n$  valeurs
- SORTIE :  $i \in \{1, \dots, n\}$  tq  $T[i]$  apparaît  $> n/2$  fois dans  $T$   
(ou PAS D'ÉLÉMENT MAJORITY)

(comparaison des valeurs par égalité)



# Algorithme EST\_ÉLÉMENT\_MAJORITAIRE ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI si  $m$  est élément majoritaire de  $T$  et FAUX sinon

```
c ← 0
pour j de 1 à n faire
    si m = T[j] alors
        c ← c + 1
    fin si
fin pour
si c > n/2 alors
    retourner VRAI
sinon
    retourner FAUX
fin si
```

## Complexité

Nombres de comparaison :

# Algorithme EST\_ÉLÉMENT\_MAJORITAIRE ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI si  $m$  est élément majoritaire de  $T$  et FAUX sinon

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m = T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $c > n/2$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :

# Algorithme EST\_ÉLÉMENT\_MAJORITAIRE ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI si  $m$  est élément majoritaire de  $T$  et FAUX sinon

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m = T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $c > n/2$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :

# Algorithme EST\_ÉLÉMENT\_MAJORITAIRE ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI si  $m$  est élément majoritaire de  $T$  et FAUX sinon

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m = T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $c > n/2$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :  $O(n)$

# Algorithme ÉLÉMENT\_MAJORITAIRE\_NAÏF

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élém. maj. de  $T$  ou PAS D'ÉLÉMENT MAJORITAIRE

pour  $i$  de 1 à  $n$  faire

$m \leftarrow T[i]$

    si EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) alors

        retourner  $m$

    fin si

fin pour

retourner PAS D'ÉLÉMENT MAJORITAIRE

## Complexité

Nombres de comparaison :

# Algorithme ÉLÉMENT\_MAJORITAIRE\_NAÏF

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élém. maj. de  $T$  ou PAS D'ÉLÉMENT MAJORITAIRE

**pour**  $i$  de 1 à  $n$  **faire**

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin pour**

**retourner** PAS D'ÉLÉMENT MAJORITAIRE

## Complexité

Nombres de comparaison :

# Algorithme ÉLÉMENT\_MAJORITAIRE\_NAÏF

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élém. maj. de  $T$  ou PAS D'ÉLÉMENT MAJORITAIRE

**pour**  $i$  de 1 à  $n$  **faire**

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin pour**

**retourner** PAS D'ÉLÉMENT MAJORITAIRE

## Complexité

Nombres de comparaison :

# Algorithme ÉLÉMENT\_MAJORITAIRE\_NAÏF

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élém. maj. de  $T$  ou PAS D'ÉLÉMENT MAJORITAIRE

**pour**  $i$  de 1 à  $n$  **faire**

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

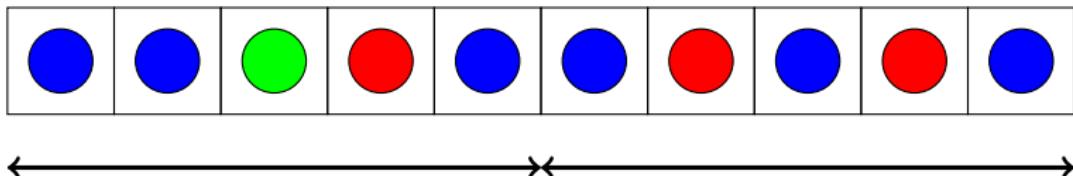
**fin pour**

**retourner** PAS D'ÉLÉMENT MAJORITAIRE

## Complexité

Nombres de comparaison :  $O(n^2)$

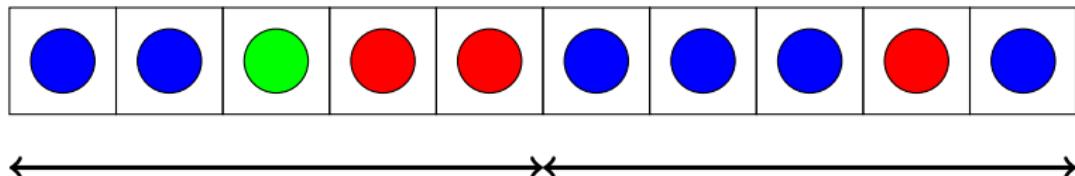
# Algorithme « diviser pour régner »



## Exemple 1

- $m$  élément maj. de  $T[1 \dots n]$   
~~~  $m$  élém. maj. de  $T[1 \dots n/2]$  ou  $m$  élém. maj. de  $T[n/2 \dots n]$
- Réciproque fausse !

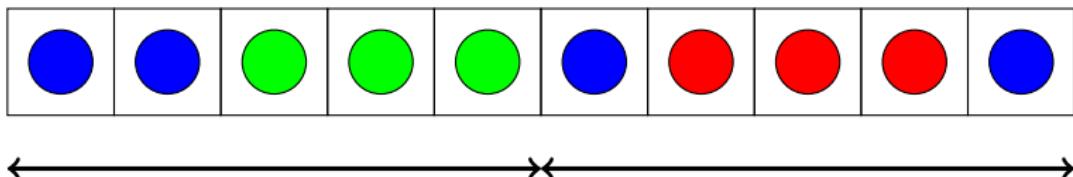
# Algorithme « diviser pour régner »



## Exemple 2

- $m$  élément maj. de  $T[1 \dots n]$   
~~~  $m$  élém. maj. de  $T[1 \dots n/2]$  ou  $m$  élém. maj. de  $T[n/2 \dots n]$
- **Réiproque fausse !**

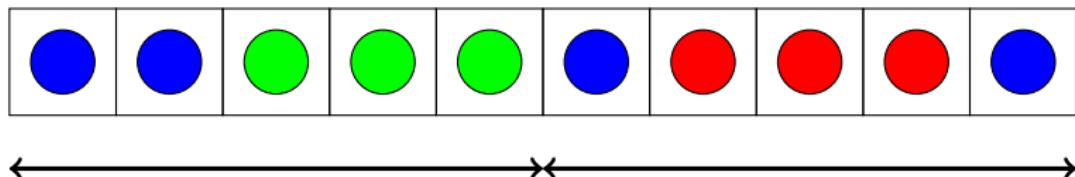
# Algorithme « diviser pour régner »



## Exemple 3

- $m$  élément maj. de  $T[1 \dots n]$   
~~~  $m$  élém. maj. de  $T[1 \dots n/2]$  ou  $m$  élém. maj. de  $T[n/2 \dots n]$
- **Réiproque fausse !**

# Algorithme « diviser pour régner »



## Exemple 3

- $m$  élément maj. de  $T[1 \dots n]$   
~~~  $m$  élém. maj. de  $T[1 \dots n/2]$  ou  $m$  élém. maj. de  $T[n/2 \dots n]$
- **Réiproque fausse !**

# Algorithme ÉLÉMENT\_MAJORITAIRE\_DPR

**Entrée:** Un tableau  $T$  de longueur  $n$ , deux entiers  $1 \leq d \leq f \leq n$

**Sortie:**  $m$  élém. maj. de  $T[d \dots f]$  ou PAS D'ÉLÉMENT MAJORITAIRE

si ( $f = d$ ) ou ( $f = d + 1$  et  $T[d] = T[f]$ ) alors

retourner  $T[d]$

fin si

$m_1 \leftarrow$  ÉLÉMENT\_MAJORITAIRE\_2( $T, 1, n/2$ )

$m_2 \leftarrow$  ÉLÉMENT\_MAJORITAIRE\_2( $T, n/2 + 1, n$ )

si EST\_ÉLÉMENT\_MAJORITAIRE?( $T, m_1$ ) alors

retourner  $m_1$

sinon si EST\_ÉLÉMENT\_MAJORITAIRE?( $T, m_2$ ) alors

retourner  $m_2$

sinon

retourner PAS D'ÉLÉMENT MAJORITAIRE

fin si

# Algorithme ÉLÉMENT\_MAJORITAIRE\_DPR

**Entrée:** Un tableau  $T$  de longueur  $n$ , deux entiers  $1 \leq d \leq f \leq n$

**Sortie:**  $m$  élém. maj. de  $T[d \dots f]$  ou PAS D'ÉLÉMENT MAJORITAIRE

**si** ( $f = d$ ) ou ( $f = d + 1$  et  $T[d] = T[f]$ ) **alors**

**retourner**  $T[d]$

**fin si**

$m_1 \leftarrow$  ÉLÉMENT\_MAJORITAIRE\_2( $T, 1, n/2$ )

$m_2 \leftarrow$  ÉLÉMENT\_MAJORITAIRE\_2( $T, n/2 + 1, n$ )

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m_1$ ) **alors**

**retourner**  $m_1$

**sinon si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m_2$ ) **alors**

**retourner**  $m_2$

**sinon**

**retourner** PAS D'ÉLÉMENT MAJORITAIRE

**fin si**

# Algorithme ÉLÉMENT\_MAJORITAIRE\_DPR

## Complexité

Nombres de comparaison :  $C(n) = O(n \log n)$

$$C(n) = 2C(n/2) + 2n \quad (\rightsquigarrow \text{Récurrence du tri fusion !})$$

$$\begin{aligned} C(2^k) &= 2C(2^{k-1}) + 2^{k+1} = 2(2C(2^{k-2}) + 2^k) + 2^{k+1} \\ &= 2^2 C(2^{k-2}) + 2 \cdot 2^{k+1} \\ &= \dots \\ &= 2^t C(2^{k-t}) + t \cdot 2^{k+1} \\ &= \dots \\ &= k \cdot 2^{k+1} = 2 \log(n) \cdot n \end{aligned}$$

# Algorithme ÉLÉMENT\_MAJORITAIRE\_DPR

## Complexité

Nombres de comparaison :  $C(n) = O(n \log n)$

$$C(n) = 2C(n/2) + 2n \quad (\rightsquigarrow \text{Récurrence du } \mathbf{\text{tri fusion}} !)$$

$$\begin{aligned} C(2^k) &= 2C(2^{k-1}) + 2^{k+1} = 2(2C(2^{k-2}) + 2^k) + 2^{k+1} \\ &= 2^2 C(2^{k-2}) + 2 \cdot 2^{k+1} \\ &= \dots \\ &= 2^t C(2^{k-t}) + t \cdot 2^{k+1} \\ &= \dots \\ &= k \cdot 2^{k+1} = 2 \log(n) \cdot n \end{aligned}$$

# Algorithme ÉLÉMENT\_MAJORITAIRE\_DPR

## Complexité

Nombres de comparaison :  $C(n) = O(n \log n)$

$$C(n) = 2C(n/2) + 2n \quad (\rightsquigarrow \text{Récurrence du } \mathbf{\text{tri fusion}} !)$$

$$\begin{aligned} C(2^k) &= 2C(2^{k-1}) + 2^{k+1} = 2(2C(2^{k-2}) + 2^k) + 2^{k+1} \\ &= 2^2 C(2^{k-2}) + 2 \cdot 2^{k+1} \\ &= \dots \\ &= 2^t C(2^{k-t}) + t \cdot 2^{k+1} \\ &= \dots \\ &= k \cdot 2^{k+1} = 2 \log(n) \cdot n \end{aligned}$$

# Élément majoritaire (avec promesse)

## ÉLÉMENT MAJORITAIRE (AVEC PROMESSE)

- ENTRÉE :  $n \geq 2$  un entier et  $T$  un tableau de  $n$  valeurs  
 $\rightsquigarrow T$  contient un élément majoritaire !
  - SORTIE :  $i \in \{1, \dots, n\}$  tq  $T[i]$  apparaît  $> n/2$  fois dans  $T$   
(comparaison des valeurs par égalité)
- 
- Les algorithmes déterministes précédents ne semblent pas améliorables
  - $\rightsquigarrow$  algorithmes probabilistes !

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

```
i      {1,...,n}  
retourner T[i]
```

## Complexité

Nombres de comparaison :

1) 0

2)  $\Theta(n)$

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\text{ }} \begin{array}{|c|c|} \hline \bullet & \square \\ \hline \square & \square \\ \hline \end{array} \quad \{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

```
i ←  {1, ..., n}  
retourner T[i]
```

## Complexité

Nombres de comparaison :

## Validité

Probabilité d'erreur :

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

```
i ←  {1, ..., n}  
retourner T[i]
```

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

```
i ←  {1, ..., n}  
retourner T[i]
```

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :

# Algorithme de type Monte-Carlo

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

```
i ←  {1, ..., n}  
retourner T[i]
```

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :  $\leq 1/2$

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

tant que VRAI faire

$i \in \{1, \dots, n\}$

$m \leftarrow T[i]$

    si EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) alors

        retourner  $m$

    fin si

fin tant que

## Complexité

Nombres de comparaison :

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ }} \{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

  $i \xleftarrow{\text{ }} \{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :

## Validité

Probabilité d'erreur :

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

  $i \xleftarrow{\text{ }} \{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur :

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

  $i \xleftarrow{\text{ }} \{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur :

# Algorithme de type Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ }} \boxed{\bullet} \boxed{\bullet}$   $\{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_ÉLÉMENT\_MAJORITAIRE ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur : 0

# Loi géométrique

- **Loi de Bernoulli**  $X$

$$\mathbb{P}(X = x) = \begin{cases} p & \text{si } x = 1, \\ 1 - p & \text{si } x = 0, \\ 0 & \text{sinon.} \end{cases}$$

- **Loi géométrique**  $Y$  = loi du nombre d'épreuves de Bernoulli indépendantes nécessaire pour obtenir le premier succès

$$\mathbb{P}(Y = k) = (1 - p)^{k-1} p$$

$$\mathbb{E}[Y] = \frac{1}{p}$$

# Loi géométrique

- **Loi de Bernoulli**  $X$

$$\mathbb{P}(X = x) = \begin{cases} p & \text{si } x = 1, \\ 1 - p & \text{si } x = 0, \\ 0 & \text{sinon.} \end{cases}$$

- **Loi géométrique**  $Y$  = loi du nombre d'épreuves de Bernoulli indépendantes nécessaire pour obtenir le premier succès

$$\mathbb{P}(Y = k) = (1 - p)^{k-1} p$$

$$\mathbb{E}[Y] = \frac{1}{p}$$

# Loi géométrique

- **Loi de Bernoulli**  $X$

$$\mathbb{P}(X = x) = \begin{cases} p & \text{si } x = 1, \\ 1 - p & \text{si } x = 0, \\ 0 & \text{sinon.} \end{cases}$$

- **Loi géométrique**  $Y$  = loi du nombre d'épreuves de Bernoulli indépendantes nécessaire pour obtenir le premier succès

$$\mathbb{P}(Y = k) = (1 - p)^{k-1} p$$

$$\mathbb{E}[Y] = \frac{1}{p}$$

# Loi géométrique (démonstration)

Pour tout  $x$  de  $[0, 1[$ ,

$$f(x) = \sum_{k=0}^{+\infty} x^k = \frac{1}{1-x}$$

$$f'(x) = \sum_{k=1}^{+\infty} kx^{k-1} = \frac{1}{(1-x)^2}$$

En particulier :

$$\begin{aligned}\mathbb{E}(Y) &= \sum_{k=1}^{+\infty} k \cdot \mathbb{P}(Y = k) \\ &= \sum_{k=1}^{+\infty} k(1-p)^{k-1} p = \frac{p}{(1-(1-p))^2} = \frac{1}{p}\end{aligned}$$

# Loi géométrique (démonstration)

Pour tout  $x$  de  $[0, 1[$ ,

$$f(x) = \sum_{k=0}^{+\infty} x^k = \frac{1}{1-x}$$

$$f'(x) = \sum_{k=1}^{+\infty} kx^{k-1} = \frac{1}{(1-x)^2}$$

En particulier :

$$\begin{aligned}\mathbb{E}(Y) &= \sum_{k=1}^{+\infty} k \cdot \mathbb{P}(Y = k) \\ &= \sum_{k=1}^{+\infty} k(1-p)^{k-1} p = \frac{p}{(1-(1-p))^2} = \frac{1}{p}\end{aligned}$$

# Algorithme de type Las Vegas

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

- Chaque **itération** de la boucle :  $n$  comparaisons
- **Nombre d'itérations** : en moyenne moins de 2 !

Notons  $p$  la probabilité de tirer l'élément majoritaire  $m$  de  $T$  :

$$p = \frac{\#\{i \in \{1, \dots, n\} \mid T[i] = m\}}{n} > \frac{n/2}{n} = \frac{1}{2}$$

Nombre d'itérations espéré :  $1/p \leq 2$   
(Espérance Loi géométrique)

## Remarque

Il existe des algorithmes déterministes simples de complexité  $O(n)$   
(1981, R. S. Boyer, J. S. Moore – cf. Compléments du TD)

# Algorithme de type Las Vegas

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

- Chaque **itération** de la boucle :  $n$  comparaisons
- **Nombre d'itérations** : en moyenne **moins de 2 !**

Notons  $p$  la probabilité de tirer l'élément majoritaire  $m$  de  $T$  :

$$p = \frac{\#\{i \in \{1, \dots, n\} \mid T[i] = m\}}{n} > \frac{n/2}{n} = \frac{1}{2}$$

Nombre d'itérations **espéré** :  $1/p \leq 2$   
(Espérance Loi géométrique)

## Remarque

Il existe des algorithmes déterministes simples de complexité  $O(n)$   
(1981, R. S. Boyer, J. S. Moore – cf. Compléments du TD)

# Table des matières

## 1 Introduction

- Historique (très) incomplet
- Principe général
- Algorithmes déterministes
- Algorithmes probabilistes

## 2 Arbre binaire ET-OU

- Algorithmes déterministes
- Algorithmes probabiliistes

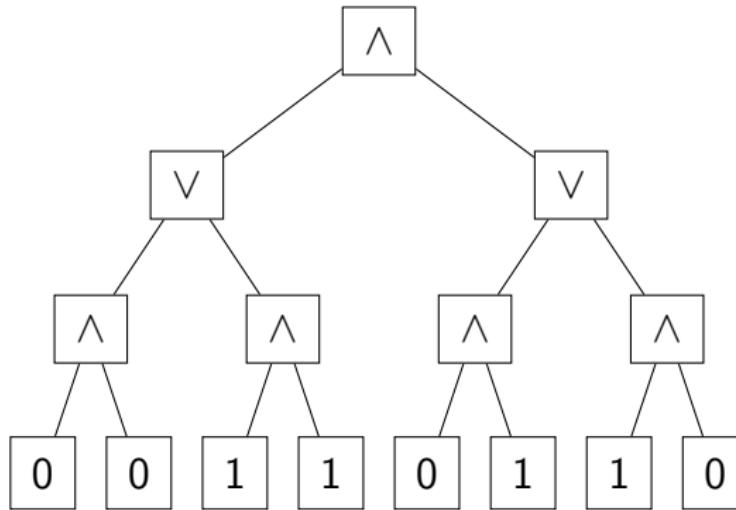
## 3 Tri et sélection rapides

- Tri rapide
- Sélection rapide
- Médian approché

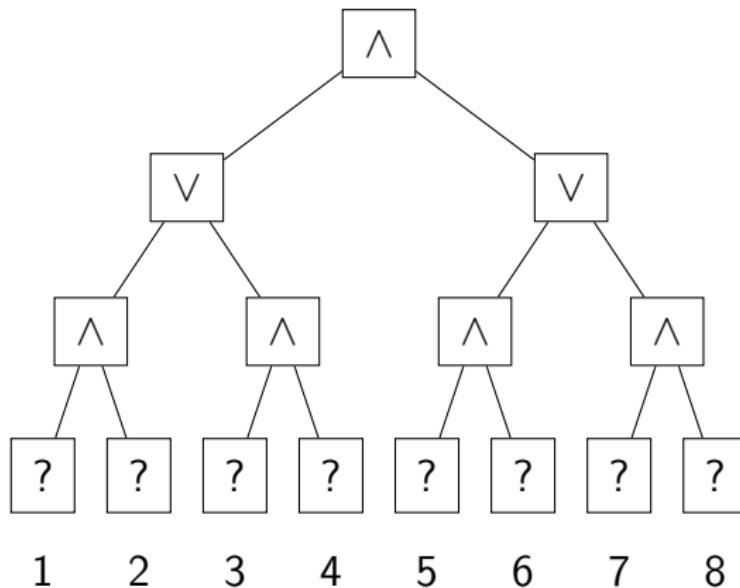
# Arbre binaire ET-OU

arbre binaire complet de profondeur  $n$  – feuille étiquetée 0 ou 1 :

- la valeur d'une feuille (profondeur  $n$ ) est son étiquette
- la valeur d'un nœud interne (profondeur  $i$ ) :
  - « ou logique » ( $\vee$ ) de la valeur de ces deux fils si  $i \equiv n \bmod 2$
  - « et logique » ( $\wedge$ ) de la valeur de ces deux fils si  $i \not\equiv n \bmod 2$

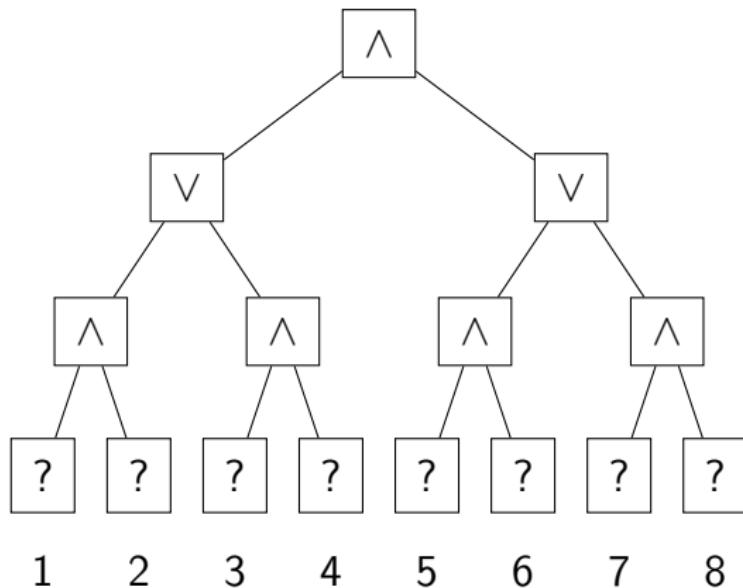


# Algorithmes déterministes



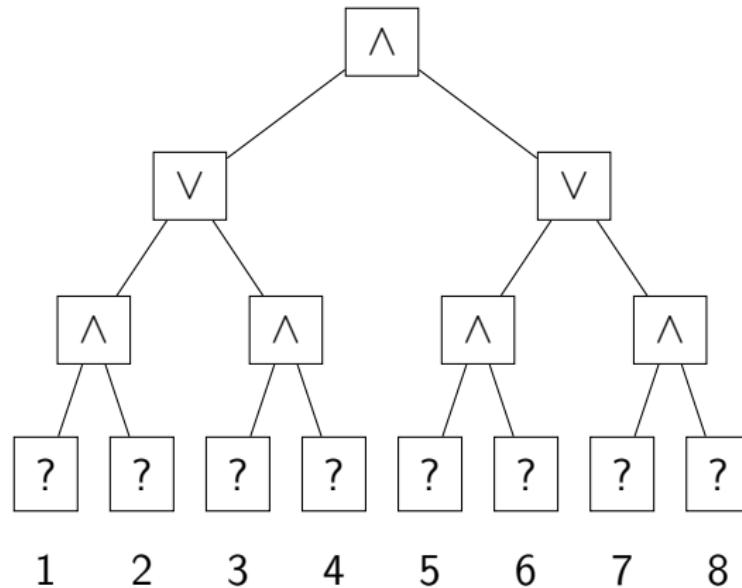
- Il existe un algorithme déterministe de complexité  $O(2^n)$
- Tout algorithme déterministe a une complexité en  $\Omega(2^n)$

# Algorithmes déterministes



- Il existe un algorithme déterministe de complexité  $O(2^n)$
- Tout algorithme déterministe a une complexité en  $\Omega(2^n)$

# Algorithme probabiliste



# Algorithme probabiliste ET-OU-PROBABILISTE

**si**  $n = 1$  **alors**

**retourner** valeur de la feuille

**sinon si**  $n \geq 2$  et le racine de l'arbre est  $\wedge$  **alors**

$b \xleftarrow{\text{ }} \{0, 1\}$

**si**  $b = 0$  **alors**

$v \leftarrow$  ET-OU-PROBABILISTE(sous-arbre gauche)

**sinon**

$v \leftarrow$  ET-OU-PROBABILISTE(sous-arbre droit)

**fin si**

**si**  $v = 0$  **alors**

**retourner** 0

**sinon si**  $b = 0$  **alors**

**retourner** ET-OU-PROBABILISTE(sous-arbre droit)

**sinon**

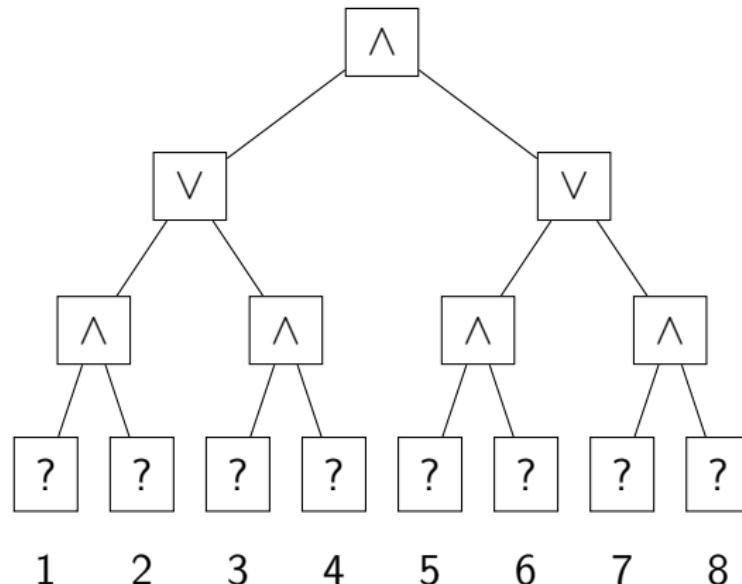
**retourner** ET-OU-PROBABILISTE(sous-arbre gauche)

**fin si**

**sinon si**  $n \geq 2$  et le racine de l'arbre est  $\vee$  **alors**

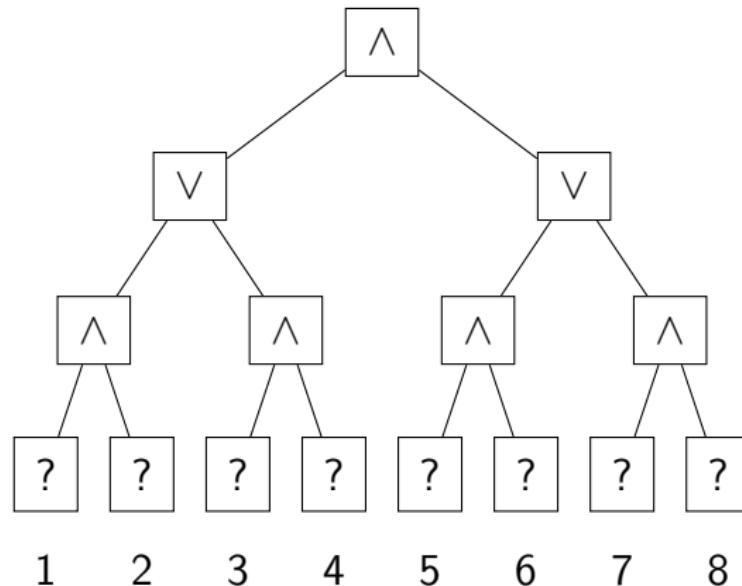
**fin si**

# Algorithme probabiliste



- Liste de choix aléatoires :

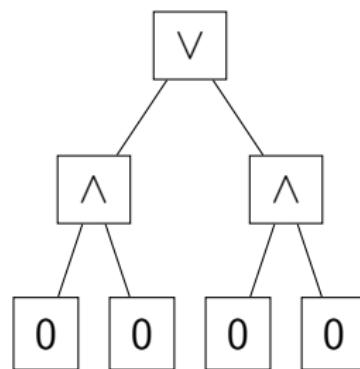
# Algorithme probabiliste



- Liste de choix aléatoires : ←←→←→→←

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

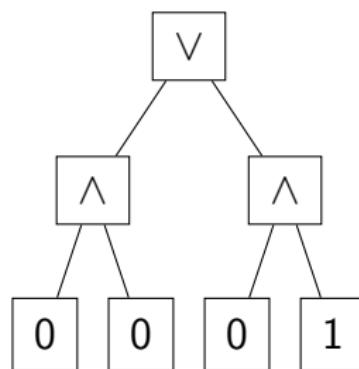


Nombre de feuilles examinées en moyenne

2

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

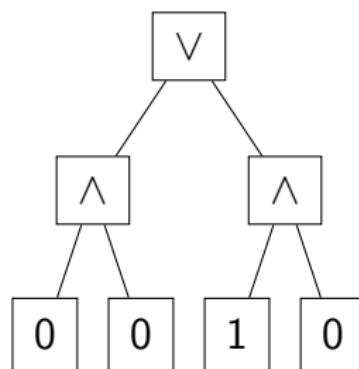


Nombre de feuilles examinées en moyenne

$$(2 + 3 + 2 + 3 + 2 + 2 + 3 + 3) / 8 = 2.5$$

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

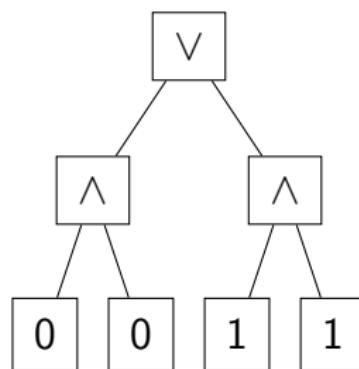


Nombre de feuilles examinées en moyenne

$$(2 + 3 + 2 + 3 + 3 + 3 + 2 + 2) / 8 = 2.5$$

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

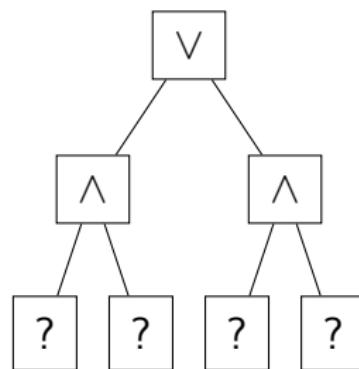


Nombre de feuilles examinées en moyenne

$$(3 + 3 + 3 + 3 + 2 + 2 + 2 + 2) / 8 = 2.5$$

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

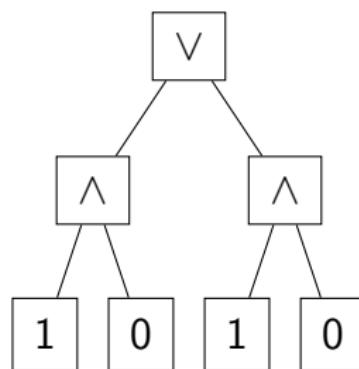


Nombre de feuilles examinées en moyenne

...

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

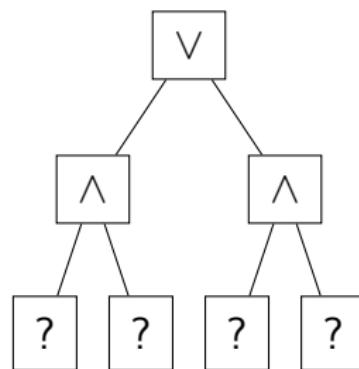


Nombre de feuilles examinées en moyenne

$$(4 + 3 + 3 + 2 + 4 + 3 + 3 + 2) / 8 = 3$$

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

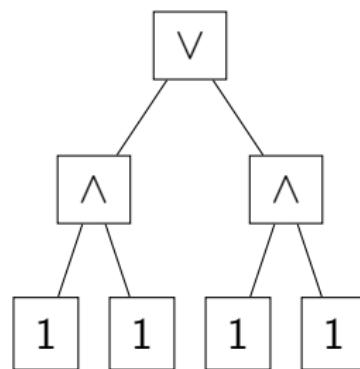


Nombre de feuilles examinées en moyenne

...

# Algorithme probabiliste - Analyse

Nombre de feuilles examinées en moyenne (pour toutes les instances)

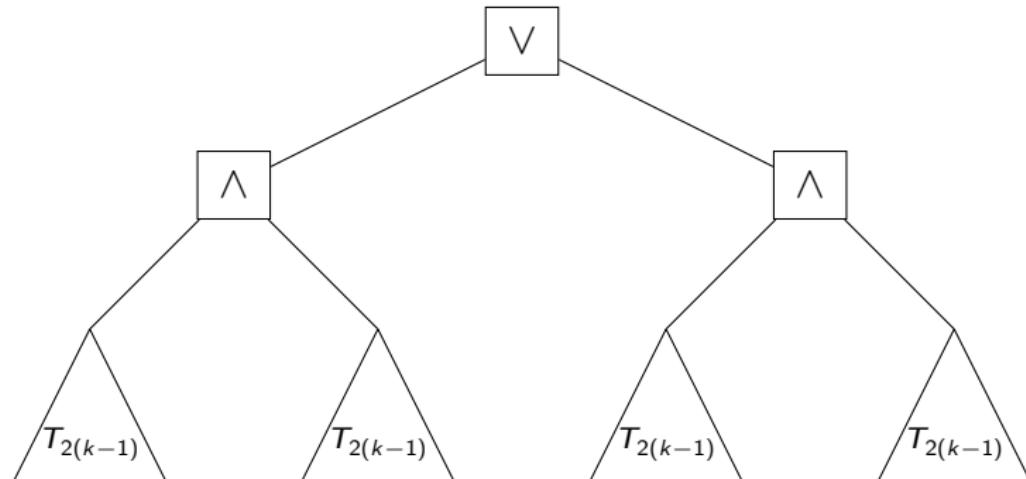


Nombre de feuilles examinées en moyenne

2

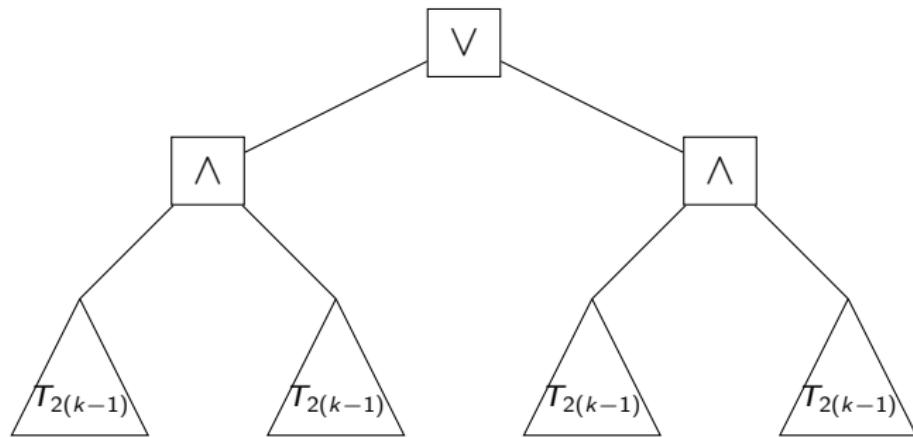
# Algorithme probabiliste - Analyse

- Pour un arbre de profondeur 2, en moyenne, on regarde moins de 3 feuilles
- Par récurrence, nous allons montrer que pour un arbre de profondeur  $2k \geq 2$ , on regarde en moyenne moins de  $3^k$  feuilles



$T_{2(k-1)}$  désigne des arbres de profondeur  $2(k - 1)$   
qui peuvent être différents

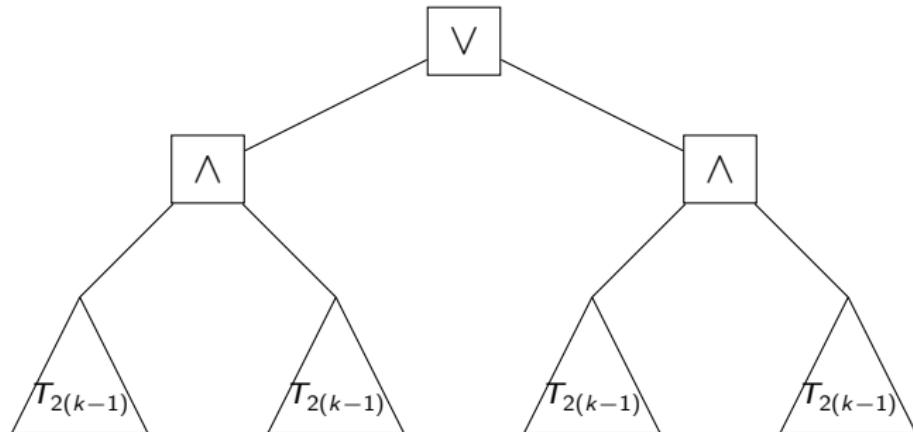
# Algorithme probabiliste - Analyse



- $\wedge \rightsquigarrow 0$ , un des sous-arbres  $\rightsquigarrow 0$ .

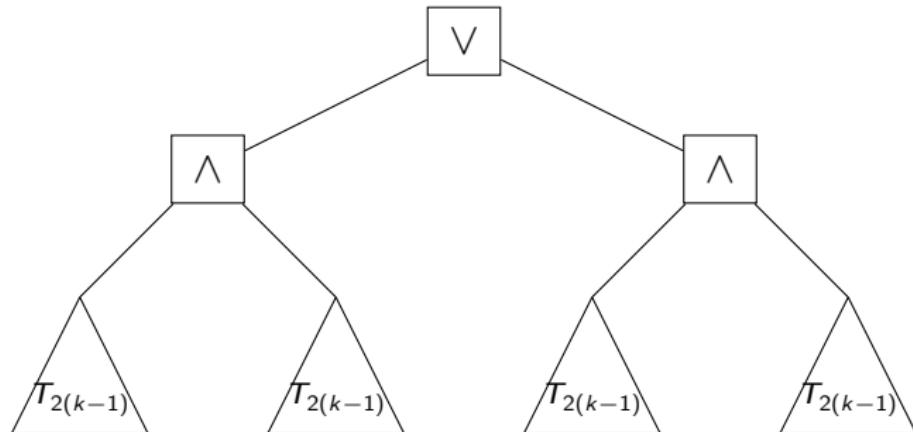
$$\leq \frac{1}{2}3^{k-1} + \frac{1}{2} \cdot 2 \cdot 3^{k-1} = \frac{3}{2}3^{k-1} \text{ nœuds}$$

# Algorithme probabiliste - Analyse



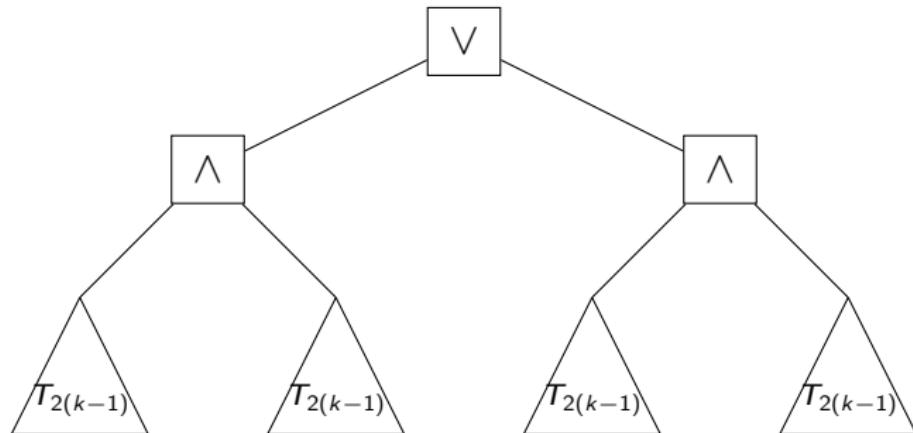
- $\wedge \rightsquigarrow 0 : \leq 1.5 \cdot 3^{k-1}$
- $\wedge \rightsquigarrow 1 : \leq 2 \cdot 3^{k-1}$
- $\vee \rightsquigarrow 0 : \leq 2 \cdot 1.5 \cdot 3^{k-1} = 3^k$
- $\vee \rightsquigarrow 1 :$ 
$$\leq \frac{1}{2}2 \cdot 3^{k-1} + \frac{1}{2}(1.5 \cdot 3^{k-1} + 2 \cdot 3^{k-1}) \leq 2.75 \cdot 3^{k-1} \leq 3^k$$

# Algorithme probabiliste - Analyse



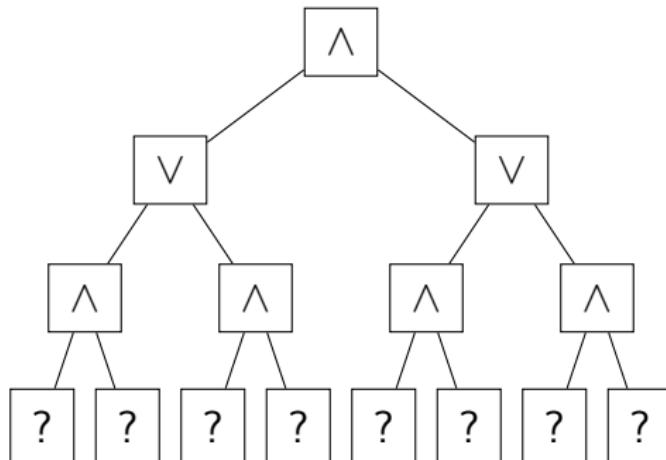
- $\wedge \rightsquigarrow 0 : \leq 1.5 \cdot 3^{k-1}$
- $\wedge \rightsquigarrow 1 : \leq 2 \cdot 3^{k-1}$
- $\vee \rightsquigarrow 0 : \leq 2 \cdot 1.5 \cdot 3^{k-1} = 3^k$
- $\vee \rightsquigarrow 1 :$ 
$$\leq \frac{1}{2}2 \cdot 3^{k-1} + \frac{1}{2}(1.5 \cdot 3^{k-1} + 2 \cdot 3^{k-1}) \leq 2.75 \cdot 3^{k-1} \leq 3^k$$

# Algorithme probabiliste - Analyse



- $\wedge \rightsquigarrow 0 : \leq 1.5 \cdot 3^{k-1}$
- $\wedge \rightsquigarrow 1 : \leq 2 \cdot 3^{k-1}$
- $\vee \rightsquigarrow 0 : \leq 2 \cdot 1.5 \cdot 3^{k-1} = 3^k$
- $\vee \rightsquigarrow 1 :$ 
$$\leq \frac{1}{2}2 \cdot 3^{k-1} + \frac{1}{2}(1.5 \cdot 3^{k-1} + 2 \cdot 3^{k-1}) \leq 2.75 \cdot 3^{k-1} \leq 3^k$$

# Algorithme probabiliste - Conclusion



## Conclusion

- Pour arbre binaire ET-OU de profondeur  $n$ , complexité en

$$O(3^{n/2}) = O(\sqrt{3}^n) = O(2^{0.793n})$$

- Gain exponentiel grâce à l'aléa !

# Table des matières

## 1 Introduction

- Historique (très) incomplet
- Principe général
- Algorithmes déterministes
- Algorithmes probabilistes

## 2 Arbre binaire ET-OU

- Algorithmes déterministes
- Algorithmes probabilistes

## 3 Tri et sélection rapides

- Tri rapide
- Sélection rapide
- Médian approché

# Tri rapide

## TRI

- ENTRÉE :  $n \geq 2$  et  $T$  un tableau de  $n$  valeurs distinctes
- SORTIE :  $T$  tableau trié (c.à.d.  $T[i] < T[i + 1]$  pour  $i \in \{1, \dots, n - 1\}$ )
- 1961 - C. A. R. Hoare
- « diviser pour régner »
- Principe :
  - sélectionner un pivot
  - le placer à sa place définitive avec
    - éléments inférieurs à gauche
    - éléments supérieurs à droite
  - répéter pour les sous-tableaux jusqu'à ce que le tableau soit trié

# Tri rapide

## TRI

- ENTRÉE :  $n \geq 2$  et  $T$  un tableau de  $n$  valeurs distinctes
- SORTIE :  $T$  tableau trié (c.à.d.  $T[i] < T[i + 1]$  pour  $i \in \{1, \dots, n - 1\}$ )
- 1961 - C. A. R. Hoare
- « diviser pour régner »
- **Principe :**
  - sélectionner un **pivot**
  - le placer à sa place définitive avec
    - éléments inférieurs à gauche
    - éléments supérieurs à droite
  - répéter pour les sous-tableaux jusqu'à ce que le tableau soit trié

# Algorithme PARTITIONNER

**Entrée:** Un tableau  $T$  de longueur  $n$ , trois entiers  $d, f, p$  avec  
 $1 \leq d \leq p \leq f \leq n$

**Sortie:**  $j \in [d, f]$ , Tableau réordonné avec  $T[i] < T[j]$  pour  
 $d \leq i < j$ ;  $T[i] > T[j]$  pour  $f \geq i > j$

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$

# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 8 | 4 | 2 | 6 | 3 | 1 | 7 |
|---|---|---|---|---|---|---|---|

# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 8 | 4 | 2 | 6 | 7 | 1 | 3 |
|---|---|---|---|---|---|---|---|

# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

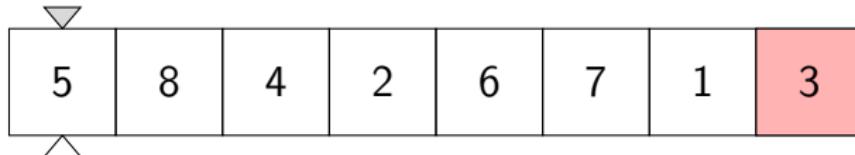
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

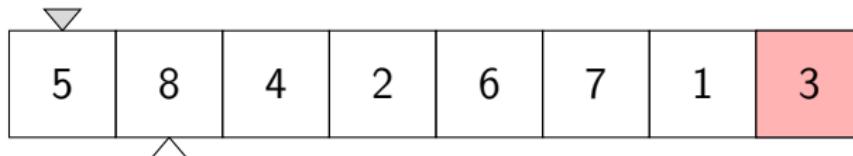
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

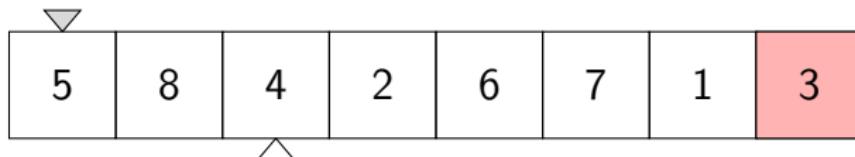
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

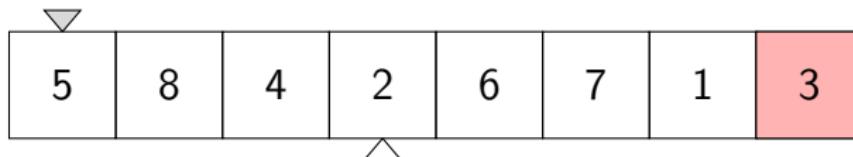
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

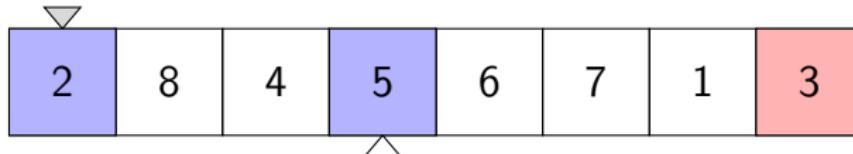
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

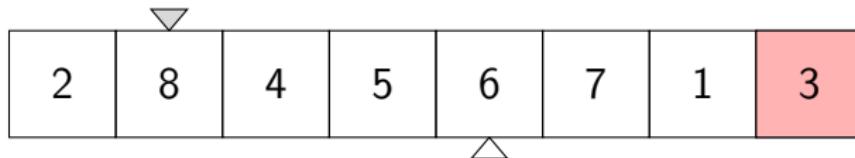
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

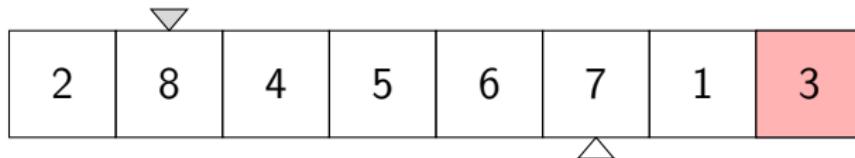
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

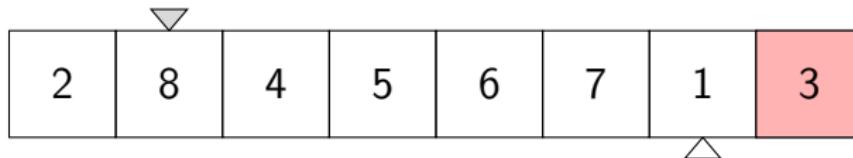
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

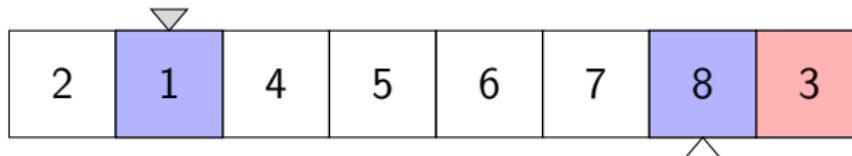
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

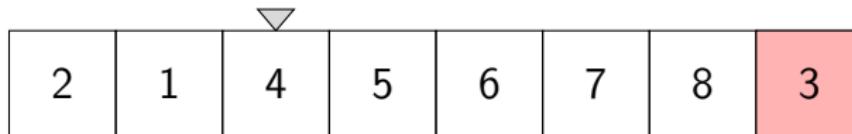
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme PARTITIONNER (Exemple)

Échanger  $T[p]$  et  $T[f]$

$j \leftarrow d$

**pour**  $i$  de  $d$  à  $f - 1$  **faire**

**si**  $T[i] \leq T[f]$  **alors**

        Échanger  $T[i]$  et  $T[j]$

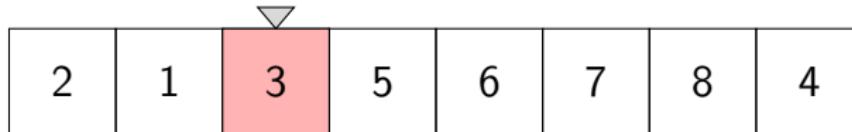
$j \leftarrow j + 1$

**fin si**

**fin pour**

Échanger  $T[j]$  et  $T[f]$

**retourner**  $j$



# Algorithme TRIRAPIDE

**Entrée:** Un tableau  $T$  de longueur  $n$ , deux entiers  $d, f$  avec

$$1 \leq d \leq f \leq n$$

**Sortie:** Tableau trié

$$p \xleftarrow{\text{partitionner}} \{d, \dots, f\}$$

$$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$$

$$\text{TRIRAPIDE}(T, d, p - 1)$$

$$\text{TRIRAPIDE}(T, p, f)$$

$$\triangleright \quad \text{TRIRAPIDE}(T, 1, n)$$

## Complexité

Nombres de comparaison :

- Dans le pire des cas :
- En moyenne :

# Algorithme TRIRAPIDE

**Entrée:** Un tableau  $T$  de longueur  $n$ , deux entiers  $d, f$  avec

$$1 \leq d \leq f \leq n$$

**Sortie:** Tableau trié

$$p \xleftarrow{\text{partitionner}} \{d, \dots, f\}$$

$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$

$\text{TRIRAPIDE}(T, d, p - 1)$

$\text{TRIRAPIDE}(T, p, f)$

▷  $\text{TRIRAPIDE}(T, 1, n)$

## Complexité

Nombres de comparaison :

- Dans le pire des cas :  $O(n^2)$
- En moyenne :

# Algorithme TRIRAPIDE

**Entrée:** Un tableau  $T$  de longueur  $n$ , deux entiers  $d, f$  avec

$$1 \leq d \leq f \leq n$$

**Sortie:** Tableau trié

$$p \xleftarrow{\text{tri}} \{d, \dots, f\}$$

$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$

$\text{TRIRAPIDE}(T, d, p - 1)$

$\text{TRIRAPIDE}(T, p, f)$

▷  $\text{TRIRAPIDE}(T, 1, n)$

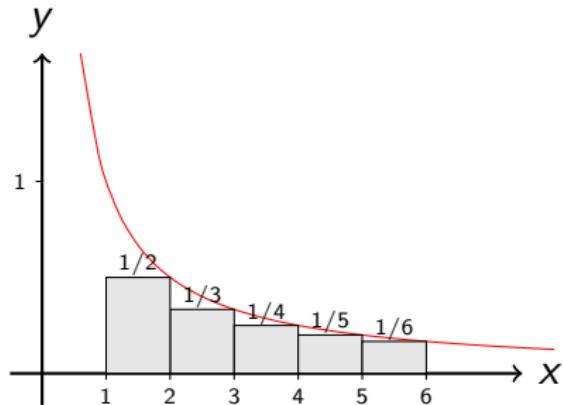
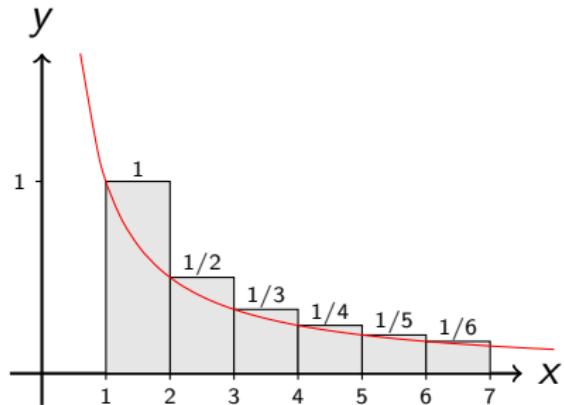
## Complexité

Nombres de comparaison :

- Dans le pire des cas :  $O(n^2)$
- En moyenne :  $O(n \log n)$

# Nombres harmoniques (par l'image)

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k}.$$



$$H_n \sim \ln n.$$

# Nombres harmoniques (plus formel)

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k}.$$

- Nous avons

$$\int_k^{k+1} \frac{1}{t} dt \leq \frac{1}{k} \leq \int_{k-1}^k \frac{1}{t} dt.$$

- En sommant de 1 à  $n$  :

$$\int_1^{n+1} \frac{1}{t} dt \leq H_n \leq 1 + \int_1^n \frac{1}{t} dt.$$

- Donc :

$$\ln(n+1)t \leq H_n \leq \ln(n) + 1 \text{ et } H_n \sim \ln n.$$

## Tri rapide - Analyse (1/2)

- $z_1, \dots, z_n$  = éléments du tableau par ordre croissant.
- $X$  = nombre total de comparaison dans une exécution (variable aléatoire)
- $X_{ij} =$  v.a. qui vaut 1 si  $z_i$  est comparé à  $z_j$  et 0 sinon  
Nous avons

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}$$

- On cherche à calculer  $\mathbb{E}[X]$

Par linéarité de l'espérance :

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \mathbb{E}[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \mathbb{P}(\{\text{L'algorithme compare } z_i \text{ et } z_j\})\end{aligned}$$

## Tri rapide - Analyse (2/2)

- $z_i$  est comparé à  $z_j$  sssi le premier élément de  $(z_i, \dots, z_j)$  choisi comme pivot est soit  $z_i$  soit  $z_j$ .  
(sinon  $z_i$  et  $z_j$  sont envoyés dans deux sous-tableaux différents de la partition)
- Donc

$$\mathbb{P}(\{\text{L'algorithme compare } z_i \text{ et } z_j\}) = \frac{2}{j - i + 1}$$

et

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1} \leq 2 \cdot n \cdot H_n \\ &= 2n \ln n + O(n) \\ &= O(n \log n)\end{aligned}$$

# Selection rapide

## SÉLECTION

- ENTRÉE :  $T$  un tableau de  $n$  valeurs distinctes,  $k \in \{1, \dots, n\}$
- SORTIE :  $m$  élément de  $T$  de rang  $k$ )
- 1961 - C. A. R. Hoare
- « diviser pour régner »
- Principe :
  - sélectionner un pivot
  - le placer à sa place définitive avec
    - éléments inférieurs à gauche
    - éléments supérieurs à droite
  - chercher l'élément dans le sous-tableau adéquat

# Sélection rapide

## SÉLECTION

- ENTRÉE :  $T$  un tableau de  $n$  valeurs distinctes,  $k \in \{1, \dots, n\}$
- SORTIE :  $m$  élément de  $T$  de rang  $k$ )
- 1961 - C. A. R. Hoare
- « diviser pour régner »
- **Principe :**
  - sélectionner un **pivot**
  - le placer à sa place définitive avec
    - éléments inférieurs à gauche
    - éléments supérieurs à droite
  - chercher l'élément dans le sous-tableau adéquat

# Algorithme SÉLECTIONRAPIDE

$p \leftarrow \boxed{\bullet\bullet} \{d, \dots, f\}$

$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$

**si**  $k = p$  **alors**

**retourner**  $T[p]$

**sinon si**  $k < p$  **alors**

SÉLECTIONRAPIDE( $T, d, p - 1, k$ )

**sinon**

SÉLECTIONRAPIDE( $T, p + 1, f, k - p + d - 1$ )

**fin si**

▷ SÉLECTIONRAPIDE( $T, 1, n, k$ )

## Complexité

Nombres de comparaison :

- Dans le pire des cas :
- En moyenne :

# Algorithme SÉLECTIONRAPIDE

$p \xleftarrow{\text{ }} \begin{array}{|c|c|} \hline \bullet & \bullet \\ \hline \end{array} \{d, \dots, f\}$

$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$

**si**  $k = p$  **alors**

**retourner**  $T[p]$

**sinon si**  $k < p$  **alors**

SÉLECTIONRAPIDE( $T, d, p - 1, k$ )

**sinon**

SÉLECTIONRAPIDE( $T, p + 1, f, k - p + d - 1$ )

**fin si**

▷ SÉLECTIONRAPIDE( $T, 1, n, k$ )

## Complexité

Nombres de comparaison :

- Dans le pire des cas :  $O(n^2)$
- En moyenne :

# Algorithme SÉLECTIONRAPIDE

$p \leftarrow \boxed{\bullet\bullet} \{d, \dots, f\}$

$p \leftarrow \text{PARTITIONNER}(T, d, f, p)$

**si**  $k = p$  **alors**

**retourner**  $T[p]$

**sinon si**  $k < p$  **alors**

SÉLECTIONRAPIDE( $T, d, p - 1, k$ )

**sinon**

SÉLECTIONRAPIDE( $T, p + 1, f, k - p + d - 1$ )

**fin si**

▷ SÉLECTIONRAPIDE( $T, 1, n, k$ )

## Complexité

Nombres de comparaison :

- Dans le pire des cas :  $O(n^2)$
- En moyenne :  $O(n)$

## Sélection rapide - Analyse

- Si PARTITIONNER découpe le tableau à chaque étape avec un tableau de taille maximale  $\alpha n$  (avec  $\alpha < 1$ ), nous avons :

$$C(n) \leq C(\alpha n) + n \quad \rightsquigarrow \quad C(n) = O(n)$$

- L'analyse exacte (plus difficile) donne  $C(n) \leq 2n + o(n)$

### Remarque

Nous verrons en TD une analyse formelle de  $C(n) \leq 4n + o(n)$

- Il existe un algorithme **déterministe** pour le problème de sélection de complexité  $O(n)$   
(1973 – M. Blum *et al.* – médian des médians)

### Remarque

Cet algorithme sera proposé dans les compléments du TD

## Sélection rapide - Analyse

- Si PARTITIONNER découpe le tableau à chaque étape avec un tableau de taille maximale  $\alpha n$  (avec  $\alpha < 1$ ), nous avons :

$$C(n) \leq C(\alpha n) + n \quad \rightsquigarrow \quad C(n) = O(n)$$

- L'analyse exacte (plus difficile) donne  $C(n) \leq 2n + o(n)$

### Remarque

Nous verrons en TD une analyse formelle de  $C(n) \leq 4n + o(n)$

- Il existe un algorithme **déterministe** pour le problème de sélection de complexité  $O(n)$   
(1973 – M. Blum *et al.* – médian des médians)

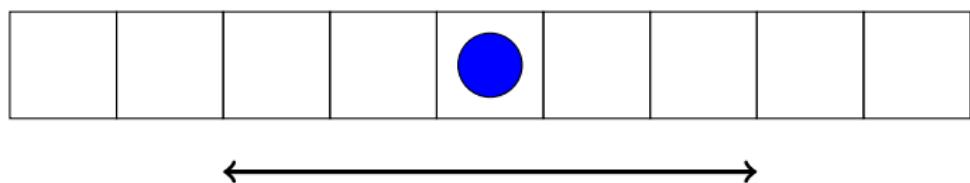
### Remarque

Cet algorithme sera proposé dans les compléments du TD

# Médian approché

## MÉDIAN APPROCHÉ

- ENTRÉE :  $T$  un tableau de  $n$  valeurs distinctes,  $k \in \{1, \dots, n\}$
- SORTIE :  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$
- $m = \text{presque médian}$

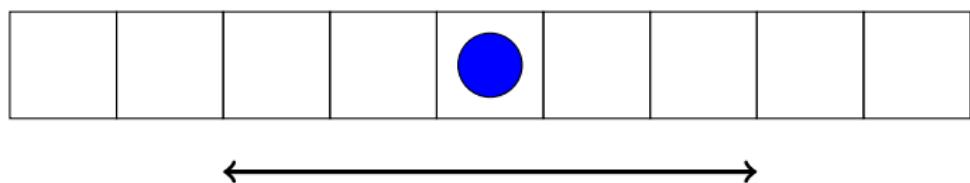


- plus facile que SÉLECTION  $\rightsquigarrow$  Complexité  $O(n)$
- Peut-on faire mieux ?
  - Las Vegas ?
  - Monte-Carlo ?

# Médian approché

## MÉDIAN APPROCHÉ

- ENTRÉE :  $T$  un tableau de  $n$  valeurs distinctes,  $k \in \{1, \dots, n\}$
- SORTIE :  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$
- $m = \text{presque médian}$



- plus facile que SÉLECTION  $\rightsquigarrow$  Complexité  $O(n)$
- Peut-on faire mieux ?
  - Las Vegas ?
  - Monte-Carlo ?

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \in \{1, \dots, n\}$   
retourner  $T[i]$

## Complexité

Nombres de comparaison :

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\text{ }} \boxed{\bullet} \boxed{\bullet}$   $\{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\text{ }} \begin{array}{|c|c|} \hline \bullet & \square \\ \hline \square & \square \\ \hline \end{array} \quad \{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :

## Validité

Probabilité d'erreur :

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\text{ }} \boxed{\bullet} \boxed{\bullet}$   $\{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\square \square} \{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :

# Médian approché - Monte-Carlo (I)

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

$i \xleftarrow{\text{ }} \begin{array}{|c|c|} \hline \bullet & \square \\ \hline \square & \square \\ \hline \end{array} \quad \{1, \dots, n\}$   
**retourner**  $T[i]$

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :  $\simeq 1/2$

# Algorithme EST\_PRESQUE\_MÉDIAN ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI sssi  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$

```
c ← 0
pour j de 1 à n faire
    si m < T[j] alors
        c ← c + 1
    fin si
fin pour
si 3n/4 ≥ c ≥ n/4 alors
    retourner VRAI
sinon
    retourner FAUX
fin si
```

## Complexité

Nombres de comparaison :

# Algorithme EST\_PRESQUE\_MÉDIAN ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI sssi  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m < T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $3n/4 \geq c \geq n/4$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :

# Algorithme EST\_PRESQUE\_MÉDIAN ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI sssi  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m < T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $3n/4 \geq c \geq n/4$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :

# Algorithme EST\_PRESQUE\_MÉDIAN ?

**Entrée:** Un tableau  $T$  de longueur  $n, m$

**Sortie:** VRAI sssi  $m$  est un élément de rang  $k \in [n/4, 3n/4]$  de  $T$

$c \leftarrow 0$

**pour**  $j$  de 1 à  $n$  **faire**

**si**  $m < T[j]$  **alors**

$c \leftarrow c + 1$

**fin si**

**fin pour**

**si**  $3n/4 \geq c \geq n/4$  **alors**

**retourner** VRAI

**sinon**

**retourner** FAUX

**fin si**

## Complexité

Nombres de comparaison :  $O(n)$

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

tant que VRAI faire

$i \in \{1, \dots, n\}$

$m \leftarrow T[i]$

    si EST\_PRESQUE\_MÉDIAN?( $T, m$ ) alors

        retourner  $m$

    fin si

fin tant que

## Complexité

Nombres de comparaison :

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

  $i \leftarrow \{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_PRESQUE\_MÉDIAN ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ }} \boxed{\bullet} \boxed{\bullet}$   $\{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_PRESQUE\_MÉDIAN ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :

## Validité

Probabilité d'erreur :

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ } \square \square \text{ }}$   $\{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_PRESQUE\_MÉDIAN ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur :

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ } \square \square \text{ }}$   $\{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_PRESQUE\_MÉDIAN ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur :

# Sélection approchée - Las Vegas

**Entrée:** Un tableau  $T$  de longueur  $n$

**Sortie:**  $m$  élément de  $T$

**tant que** VRAI faire

$i \xleftarrow{\text{ } \square \square \text{ }}$   $\{1, \dots, n\}$

$m \leftarrow T[i]$

**si** EST\_PRESQUE\_MÉDIAN ?( $T, m$ ) **alors**

**retourner**  $m$

**fin si**

**fin tant que**

## Complexité

Nombres de comparaison :  $O(n)$  en moyenne

## Validité

Probabilité d'erreur : 0

# Sélection approchée - Monte-Carlo (II)

**Entrée:** Un tableau  $T$  de longueur  $n$ ,  $k \in \mathbb{N}$  (paramètre)

**Sortie:**  $m$  élément de  $T$

$S \leftarrow \emptyset$

**pour**  $j$  de 1 à  $k$  **faire**

  $i \xleftarrow{\quad} \{1, \dots, n\}$

$S \leftarrow S \cup \{T[i]\}$

**fin pour**

Trier  $S$

▷ en temps  $O(k \log k)$

**retourner**  $S[k/2]$

▷ le médian de  $S$

## Complexité

Nombres de comparaison :

# Sélection approchée - Monte-Carlo (II)

**Entrée:** Un tableau  $T$  de longueur  $n$ ,  $k \in \mathbb{N}$  (paramètre)

**Sortie:**  $m$  élément de  $T$

$S \leftarrow \emptyset$

**pour**  $j$  de 1 à  $k$  **faire**

$i \xleftarrow{\text{ }} \{1, \dots, n\}$

$S \leftarrow S \cup \{T[i]\}$

**fin pour**

Trier  $S$

▷ en temps  $O(k \log k)$

**retourner**  $S[k/2]$

▷ le médian de  $S$

## Complexité

Nombres de comparaison :  $O(k \log k)$  ou  $O(k)$

## Validité

Probabilité d'erreur : ???

# Sélection approchée - Monte-Carlo (II)

**Entrée:** Un tableau  $T$  de longueur  $n$ ,  $k \in \mathbb{N}$  (paramètre)

**Sortie:**  $m$  élément de  $T$

$S \leftarrow \emptyset$

**pour**  $j$  de 1 à  $k$  **faire**

$i \xleftarrow{\text{ }} \{1, \dots, n\}$

$S \leftarrow S \cup \{T[i]\}$

**fin pour**

Trier  $S$

▷ en temps  $O(k \log k)$

**retourner**  $S[k/2]$

▷ le médian de  $S$

## Complexité

Nombres de comparaison :  $O(k \log k)$  ou  $O(k)$

## Validité

Probabilité d'erreur : ???

# Sélection approchée - Monte-Carlo (II)

**Entrée:** Un tableau  $T$  de longueur  $n$ ,  $k \in \mathbb{N}$  (paramètre)

**Sortie:**  $m$  élément de  $T$

$S \leftarrow \emptyset$

**pour**  $j$  de 1 à  $k$  **faire**

$i \xleftarrow{\text{ }} \begin{array}{|c|c|} \hline \bullet & \bullet \\ \hline \end{array} \{1, \dots, n\}$

$S \leftarrow S \cup \{T[i]\}$

**fin pour**

Trier  $S$

▷ en temps  $O(k \log k)$

**retourner**  $S[k/2]$

▷ le médian de  $S$

## Complexité

Nombres de comparaison :  $O(k \log k)$  ou  $O(k)$

## Validité

Probabilité d'erreur : ???

# Sélection approchée - Monte-Carlo (II)

**Entrée:** Un tableau  $T$  de longueur  $n$ ,  $k \in \mathbb{N}$  (paramètre)

**Sortie:**  $m$  élément de  $T$

$S \leftarrow \emptyset$

**pour**  $j$  de 1 à  $k$  **faire**

  $i \xleftarrow{\text{ }} \{1, \dots, n\}$

$S \leftarrow S \cup \{T[i]\}$

**fin pour**

Trier  $S$

▷ en temps  $O(k \log k)$

**retourner**  $S[k/2]$

▷ le médian de  $S$

## Complexité

Nombres de comparaison :  $O(k \log k)$  ou  $O(k)$

## Validité

Probabilité d'erreur : ???

# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ①  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ②  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) = \text{probabilité qu'une répétition de } k \text{ épreuves de Bernoulli de paramètre } p = 1/4 \text{ obtienne } \geq k/2 \text{ succès.}$

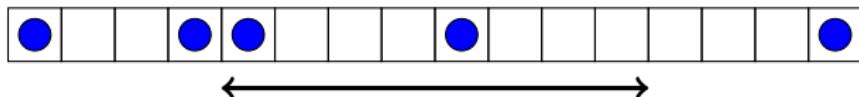
# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

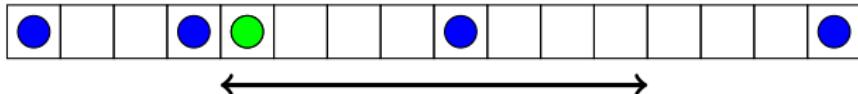
# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

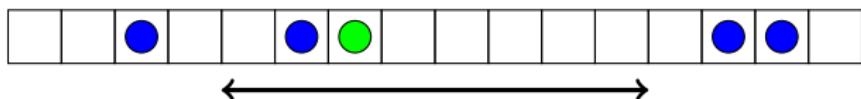
## Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

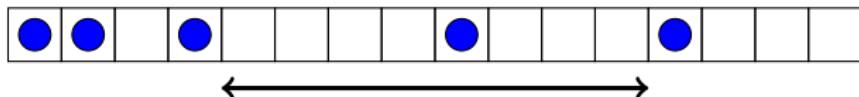
## Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

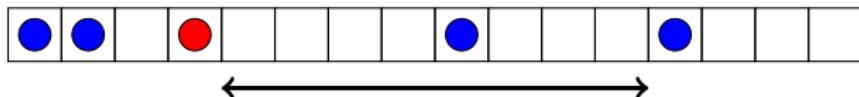
# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

# Sélection approchée - Monte-Carlo - Analyse (1/3)



+

- L'algorithme est correct sauf si  $S$  contient
  - ➊  $\geq k/2$  éléments dans le premier quart ( $E_1$ ), ou
  - ➋  $\geq k/2$  éléments dans le dernier quart ( $E_2$ )
- $\mathbb{P}(E_i) =$  probabilité qu'une répétition de  $k$  épreuves de Bernoulli de paramètre  $p = 1/4$  obtienne  $\geq k/2$  succès.

## Sélection approchée - Monte-Carlo - Analyse (2/3)

$(n \equiv 0 \pmod{4} \text{ et } k \equiv 0 \pmod{2})$

$$\begin{aligned}\mathbb{P}(E_i) &= \sum_{i=k/2}^k \binom{k}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \leq \binom{k}{k/2} \sum_{i=k/2}^k \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \\ &= \binom{k}{k/2} \left(\frac{3}{4}\right)^k \sum_{i=k/2}^k \left(\frac{1}{3}\right)^i \\ &\leq \binom{k}{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &\leq 4^{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &= \left(\frac{3}{4}\right)^{k/2} \frac{3}{2} \leq (1/2)^{k/5}\end{aligned}$$

## Sélection approchée - Monte-Carlo - Analyse (2/3)

$(n \equiv 0 \pmod{4} \text{ et } k \equiv 0 \pmod{2})$

$$\begin{aligned}\mathbb{P}(E_i) &= \sum_{i=k/2}^k \binom{k}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \leq \binom{k}{k/2} \sum_{i=k/2}^k \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \\ &= \binom{k}{k/2} \left(\frac{3}{4}\right)^k \sum_{i=k/2}^k \left(\frac{1}{3}\right)^i \\ &\leq \binom{k}{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &\leq 4^{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &= \left(\frac{3}{4}\right)^{k/2} \frac{3}{2} \leq (1/2)^{k/5}\end{aligned}$$

## Sélection approchée - Monte-Carlo - Analyse (2/3)

$(n \equiv 0 \pmod{4} \text{ et } k \equiv 0 \pmod{2})$

$$\begin{aligned}\mathbb{P}(E_i) &= \sum_{i=k/2}^k \binom{k}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \leq \binom{k}{k/2} \sum_{i=k/2}^k \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \\ &= \binom{k}{k/2} \left(\frac{3}{4}\right)^k \sum_{i=k/2}^k \left(\frac{1}{3}\right)^i \\ &\leq \binom{k}{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &\leq 4^{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &= \left(\frac{3}{4}\right)^{k/2} \frac{3}{2} \leq (1/2)^{k/5}\end{aligned}$$

## Sélection approchée - Monte-Carlo - Analyse (2/3)

$(n \equiv 0 \pmod{4} \text{ et } k \equiv 0 \pmod{2})$

$$\begin{aligned}\mathbb{P}(E_i) &= \sum_{i=k/2}^k \binom{k}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \leq \binom{k}{k/2} \sum_{i=k/2}^k \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \\ &= \binom{k}{k/2} \left(\frac{3}{4}\right)^k \sum_{i=k/2}^k \left(\frac{1}{3}\right)^i \\ &\leq \binom{k}{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &\leq 4^{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &= \left(\frac{3}{4}\right)^{k/2} \frac{3}{2} \leq (1/2)^{k/5}\end{aligned}$$

## Sélection approchée - Monte-Carlo - Analyse (2/3)

$(n \equiv 0 \pmod{4} \text{ et } k \equiv 0 \pmod{2})$

$$\begin{aligned}\mathbb{P}(E_i) &= \sum_{i=k/2}^k \binom{k}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \leq \binom{k}{k/2} \sum_{i=k/2}^k \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{k-i} \\ &= \binom{k}{k/2} \left(\frac{3}{4}\right)^k \sum_{i=k/2}^k \left(\frac{1}{3}\right)^i \\ &\leq \binom{k}{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &\leq 4^{k/2} \left(\frac{3}{4}\right)^k \left(\frac{1}{3}\right)^{k/2} \frac{3}{2} \\ &= \left(\frac{3}{4}\right)^{k/2} \frac{3}{2} \leq (1/2)^{k/5}\end{aligned}$$

# Sélection approchée - Monte-Carlo - Analyse (3/3)

- $k = 5(t + 1) \rightsquigarrow \mathbb{P}(E_i) \leq 2^{-(t+1)}$
- $k = 5(t + 1) \rightsquigarrow \mathbb{P}(\text{erreur}) \leq \mathbb{P}(E_1) + \mathbb{P}(E_2) \leq 2^{-t}$

$$k = 1500$$

## Complexité

Nombres de comparaison :  $O(1)$

## Validité

Probabilité d'erreur :  $\leq 10^{-80}$

# Sélection approchée - Monte-Carlo - Analyse (3/3)

- $k = 5(t + 1) \rightsquigarrow \mathbb{P}(E_i) \leq 2^{-(t+1)}$
- $k = 5(t + 1) \rightsquigarrow \mathbb{P}(\text{erreur}) \leq \mathbb{P}(E_1) + \mathbb{P}(E_2) \leq 2^{-t}$

$$k = 5(T \log(n) + 1) \quad (T \text{ constant})$$

## Complexité

Nombres de comparaison :  $O(\log n \log \log n)$

## Validité

Probabilité d'erreur :  $\leq n^{-T}$