

# CAML

## 9 - Diviser pour régner

<http://tsi.tuxfamily.org/OCaml>



20 avril 2024

## 1 Recherche des zéros d'une fonction

- par balayage

## 1 Recherche des zéros d'une fonction

- par balayage
- par dichotomie

- 1 Recherche des zéros d'une fonction
  - par balayage
  - par dichotomie
  - conclusion
- 2 Exponentiation rapide

## 1 Recherche des zéros d'une fonction

- par balayage
- par dichotomie
- conclusion

## 2 Exponentiation rapide

- Algorithme naïf

## 1 Recherche des zéros d'une fonction

- par balayage
- par dichotomie
- conclusion

## 2 Exponentiation rapide

- Algorithme naïf
- Principe

## 1 Recherche des zéros d'une fonction

- par balayage
- par dichotomie
- conclusion

## 2 Exponentiation rapide

- Algorithme naïf
- Principe
- Complexité

- 1 Recherche des zéros d'une fonction
  - par balayage
  - par dichotomie
  - conclusion
- 2 Exponentiation rapide
  - Algorithme naïf
  - Principe
  - Complexité
  - Exemple d'utilisation : le RSA
- 3 Multiplication d'entiers
- 4 Tris par partition-fusion



- 1 Recherche des zéros d'une fonction
  - par balayage
  - par dichotomie
  - conclusion
- 2 Exponentiation rapide
  - Algorithme naïf
  - Principe
  - Complexité
  - Exemple d'utilisation : le RSA
- 3 Multiplication d'entiers
- 4 Tris par partition-fusion
  - Principe

- 1 Recherche des zéros d'une fonction
  - par balayage
  - par dichotomie
  - conclusion
- 2 Exponentiation rapide
  - Algorithme naïf
  - Principe
  - Complexité
  - Exemple d'utilisation : le RSA
- 3 Multiplication d'entiers
- 4 Tris par partition-fusion
  - Principe
  - Complexité
- 5 Deux points les plus proches

## DIVISER POUR REGNER

Divide ut imperes

Ce terme désigne une technique de programmation qui consiste à :

- **Diviser** : on découpe le problème initial en sous-problèmes
- **Régner** : on résout les problèmes simples.
- **Combiner** : on trouve une solution au problème initial à partir des solutions des sous-problèmes.

Remarque : on utilise aussi parfois ce terme lorsque l'on découpe une fonction complexe en plusieurs fonctions plus simples pour résoudre un problème.

On considère une fonction  $f$  continue et strictement monotone sur  $[a; b]$  et changeant de signe sur cet intervalle.

## Exercice

- 1 Combien d'itérations faut-il réaliser au pire pour obtenir un encadrement d'amplitude  $\varepsilon$  à partir d'un intervalle  $[a, b]$  ?
- 2 On note  $B_n$  le nombre d'itérations pour donner une précision de  $10^{-n}$ . Quel est le lien entre  $B_n$  et  $B_{n+1}$  ?

## Exercice

- 1 Combien d'itérations faut-il réaliser au pire pour obtenir un encadrement d'amplitude  $\varepsilon$  à partir d'un intervalle  $[a, b]$  ?
- 2 On note  $B_n$  le nombre d'itérations pour donner une précision de  $10^{-n}$ . Quel est le lien entre  $B_n$  et  $B_{n+1}$  ?

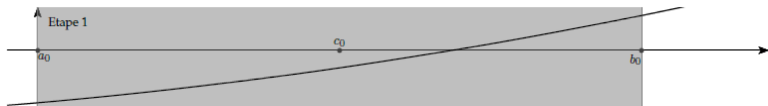
1.  $a + n\varepsilon > b \iff n > \frac{b - a}{\varepsilon}$

2.  $B_{n+1} \approx 10B_n$

## Recherche des zéros d'une fonction

Exponentiation rapide  
Multiplication d'entiers  
Tris par partition-fusion  
Deux points les plus proches

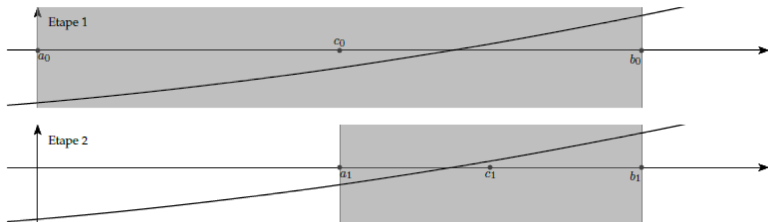
par balayage  
**par dichotomie**  
conclusion



## Recherche des zéros d'une fonction

Exponentiation rapide  
Multiplication d'entiers  
Tris par partition-fusion  
Deux points les plus proches

par balayage  
**par dichotomie**  
conclusion

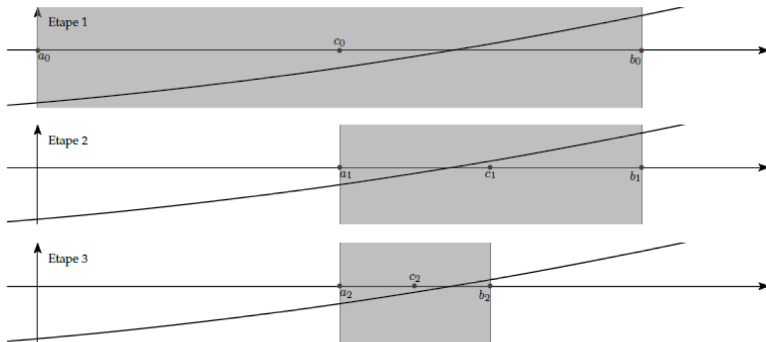


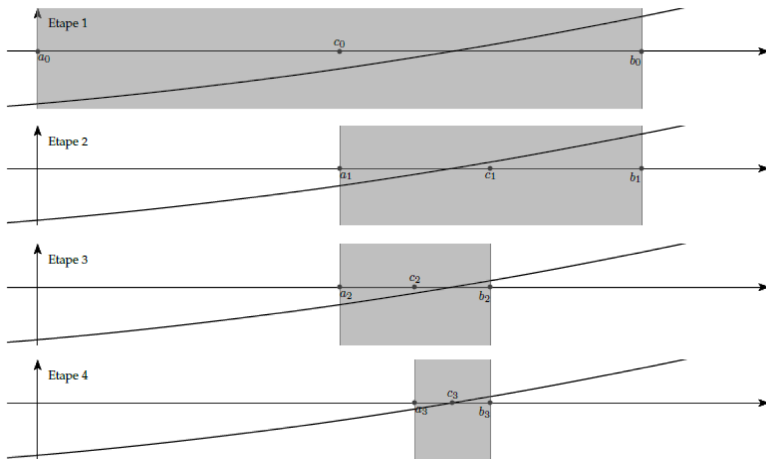


# Recherche des zéros d'une fonction

Exponentiation rapide  
Multiplication d'entiers  
Tris par partition-fusion  
Deux points les plus proches

par balayage  
**par dichotomie**  
conclusion





## Exercice

- 1 Combien d'itérations faut-il réaliser pour obtenir un encadrement d'amplitude  $\varepsilon$  à partir d'un intervalle  $[a, b]$  ?
- 2 On note  $T_n$  le nombre d'itérations pour donner une précision de  $10^{-n}$ . Quel est le lien entre  $T_n$  et  $T_{n+1}$  ?

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude

$$\text{de } A_n = \frac{b - a}{2^n}.$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$A_n < \varepsilon \iff \frac{b-a}{2^n} < \varepsilon$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned} A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\ &\iff \frac{b-a}{\varepsilon} < 2^n \end{aligned}$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned} A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\ &\iff \frac{b-a}{\varepsilon} < 2^n \\ &\iff 2^n > \frac{b-a}{\varepsilon} \end{aligned}$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned}
 A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\
 &\iff \frac{b-a}{\varepsilon} < 2^n \\
 &\iff 2^n > \frac{b-a}{\varepsilon} \\
 &\iff n \ln 2 > \ln(b-a) - \ln \varepsilon
 \end{aligned}$$



Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned}
 A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\
 &\iff \frac{b-a}{\varepsilon} < 2^n \\
 &\iff 2^n > \frac{b-a}{\varepsilon} \\
 &\iff n \ln 2 > \ln(b-a) - \ln \varepsilon \\
 &\iff n > \frac{\ln(b-a)}{\ln 2} - \frac{\ln \varepsilon}{\ln 2}
 \end{aligned}$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned}
 A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\
 &\iff \frac{b-a}{\varepsilon} < 2^n \\
 &\iff 2^n > \frac{b-a}{\varepsilon} \\
 &\iff n \ln 2 > \ln(b-a) - \ln \varepsilon \\
 &\iff n > \frac{\ln(b-a)}{\ln 2} - \frac{\ln \varepsilon}{\ln 2}
 \end{aligned}$$

Question 1 : à chaque itération, l'amplitude de l'encadrement est divisée par 2, donc à la fin de la  $n^{\text{ème}}$  itération, on a une amplitude de  $A_n = \frac{b-a}{2^n}$ . Ainsi :

$$\begin{aligned}
 A_n < \varepsilon &\iff \frac{b-a}{2^n} < \varepsilon \\
 &\iff \frac{b-a}{\varepsilon} < 2^n \\
 &\iff 2^n > \frac{b-a}{\varepsilon} \\
 &\iff n \ln 2 > \ln(b-a) - \ln \varepsilon \\
 &\iff n > \frac{\ln(b-a)}{\ln 2} - \frac{\ln \varepsilon}{\ln 2} \\
 &\iff n > \alpha \ln(1/\varepsilon) + \beta
 \end{aligned}$$

Question 2 : Ainsi la première valeur de  $n$  telle que  $A_{n+1} < \varepsilon$  est

$$T_{n+1}$$

Question 2 : Ainsi la première valeur de  $n$  telle que  $A_{n+1} < \varepsilon$  est

$$T_{n+1} = \lceil \alpha \ln(10^{n+1}) + \beta \rceil$$

Question 2 : Ainsi la première valeur de  $n$  telle que  $A_{n+1} < \varepsilon$  est

$$\begin{aligned}T_{n+1} &= \lceil \alpha \ln(10^{n+1}) + \beta \rceil \\ &= \lceil \alpha \ln(10^n) + \alpha \ln 10 + \beta \rceil\end{aligned}$$

Question 2 : Ainsi la première valeur de  $n$  telle que  $A_{n+1} < \varepsilon$  est

$$\begin{aligned}T_{n+1} &= \lceil \alpha \ln(10^{n+1}) + \beta \rceil \\ &= \lceil \alpha \ln(10^n) + \alpha \ln 10 + \beta \rceil \\ &\approx T_n + \underbrace{\alpha \ln 10 + \beta}_{\text{constante}}\end{aligned}$$

Exemple de code en OCaml :

```
let rec dichotomie f a b eps =  
  if b -. a < eps  
  then a  
  else let c = (b +.a) /. 2. in  
    if f(a) *. f(c) < 0.  
    then dichotomie f a c eps  
    else dichotomie f c b eps  
;;
```



- Par balayage :  $B_{n+1} = 10B_n$
- Par dichotomie :  $T_{n+1} = T_n + \textit{Constante}$

## Exercice

- 1 Ecrire une fonction récursive naïve qui donne  $a^n$  pour deux entiers naturels  $a$  et  $n$  non simultanément nuls.
- 2 Quelle est sa complexité ?

Version récursive :

```
let rec puiss a = function
  | 0 -> 1
  | n -> a * puiss a (n-1)
;;
```

Version récursive :

```
let rec puiss a = function
  | 0 -> 1
  | n -> a * puiss a (n-1)
;;
```

Version récursive terminale :

```
let puiss a n =
  let rec puissR r a = function
    | 0 -> r
    | n -> puissR (a*r) a (n-1)
  in puissR 1 a n
;;
```

Dans les 2 cas, la complexité est  $O(n)$

Quelques remarques sur la notation  $O$  :

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = O(v_n)$  si  $\left(\frac{u_n}{v_n}\right)$  est bornée.

Quelques remarques sur la notation  $O$  :

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = O(v_n)$  si  $\left(\frac{u_n}{v_n}\right)$  est bornée. Dit autrement
- On dit que  $u_n = O(v_n)$  si  $\exists M > 0, \forall n \in \mathbb{N}, u_n \leq Mv_n$

Quelques remarques sur la notation  $O$  :

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = O(v_n)$  si  $\left(\frac{u_n}{v_n}\right)$  est bornée. Dit autrement
- On dit que  $u_n = O(v_n)$  si  $\exists M > 0, \forall n \in \mathbb{N}, u_n \leq Mv_n$

Avec cette notation,  $u_n = O(n) \implies u_n = O(n^2)$



Quelques remarques sur la notation  $O$  :

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = O(v_n)$  si  $\left(\frac{u_n}{v_n}\right)$  est bornée. Dit autrement
- On dit que  $u_n = O(v_n)$  si  $\exists M > 0, \forall n \in \mathbb{N}, u_n \leq Mv_n$

Avec cette notation,  $u_n = O(n) \implies u_n = O(n^2) \implies u_n = O(2^n)$

Quelques remarques sur la notation  $O$  :

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = O(v_n)$  si  $\left(\frac{u_n}{v_n}\right)$  est bornée. Dit autrement
- On dit que  $u_n = O(v_n)$  si  $\exists M > 0, \forall n \in \mathbb{N}, u_n \leq Mv_n$

Avec cette notation,  $u_n = O(n) \implies u_n = O(n^2) \implies u_n = O(2^n)$   
ce qui est gênant, car ce n'est pas ce que l'on sous-entend en disant précédemment que la complexité est en  $O(n)$ .

Nouvelle notation  $\theta$  (« de l'ordre de »).

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = \theta(v_n)$  si  $u_n = O(v_n)$  et  $v_n = O(u_n)$ .

Nouvelle notation  $\theta$  (« de l'ordre de »).

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = \theta(v_n)$  si  $u_n = O(v_n)$  et  $v_n = O(u_n)$ .
- $u_n = \theta(v_n)$  si  $\exists M_1, M_2 > 0, \forall n \in \mathbb{N}, M_1 v_n \leq u_n \leq M_2 v_n$

Nouvelle notation  $\theta$  (« de l'ordre de »).

Soit  $(u_n)$  et  $(v_n)$  deux suites strictement positives,

### Vocabulaire

- On dit que  $u_n = \theta(v_n)$  si  $u_n = O(v_n)$  et  $v_n = O(u_n)$ .
- $u_n = \theta(v_n)$  si  $\exists M_1, M_2 > 0, \forall n \in \mathbb{N}, M_1 v_n \leq u_n \leq M_2 v_n$

Par exemple  $4n(n-3) \log_2(n+2) = \theta(n^2 \ln n)$ .

L'algorithme d'**exponentiation rapide** est basé sur le principe suivant :

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2 & \text{si } n \text{ pair} \\ a \times \left(a^{\frac{n-1}{2}}\right)^2 & \text{sinon} \end{cases}$$

L'algorithme d'**exponentiation rapide** est basé sur le principe suivant :

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2 & \text{si } n \text{ pair} \\ a \times \left(a^{\frac{n-1}{2}}\right)^2 & \text{sinon} \end{cases} = \begin{cases} \left(a^2\right)^{\frac{n}{2}} & \text{si } n \text{ pair} \\ a \times \left(a^2\right)^{\frac{n-1}{2}} & \text{sinon} \end{cases}$$

## Exercice

Écrire une fonction récursive appliquant cet algorithme.



## Exercice

Écrire une fonction récursive appliquant cet algorithme.

Une solution récursive :

```
let rec puiss a = function
  | 0 -> 1
  | n when n mod 2 == 0 ->
      let x = puiss a (n/2) in x*x
  | n -> let x = puiss a (n/2) in a*x*x
;;
```

## Exercice

Écrire une fonction récursive appliquant cet algorithme.

Une solution récursive plus simple :

```
let rec puiss a = function
  | 0 -> 1
  | n when n mod 2 == 0 -> puiss (a*a) (n/2)
  | n -> a * puiss (a*a) (n/2)
;;
```

## Exercice

Écrire une fonction récursive **terminale** appliquant cet algorithme.

## Exercice

Écrire une fonction récursive **terminale** appliquant cet algorithme.

Une solution récursive :

```
let puiss a n =  
  let rec puissR res a = function  
    | 0 -> res  
    | n when n mod 2 == 0 -> puissR res (a*a) (n/2)  
    | n -> puissR (res*a) (a*a) (n/2)  
  in puissR 1 a n  
;;
```

## Exercice

On note  $B_n$  le nombre de multiplications à effectuer pour calculer  $a^n$  avec cet algorithme et  $A_n$  pour l'algorithme naïf.

- 1 Montrer que  $\forall n \in \mathbb{N}, \log_2(n+1) \leq B_n \leq 2 \log_2(n+1)$ .
- 2 Conclure.

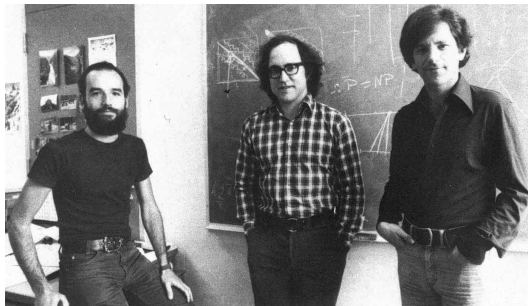
## Exercice

On note  $B_n$  le nombre de multiplications à effectuer pour calculer  $a^n$  avec cet algorithme et  $A_n$  pour l'algorithme naïf.



- 1 Montrer que  $\forall n \in \mathbb{N}, \log_2(n+1) \leq B_n \leq 2 \log_2(n+1)$ .
- 2 Conclure.

$B_n = \theta(\ln n)$  alors que  $A_n = \theta(n)$ , c'est beaucoup mieux !



- Le chiffrement RSA est un algorithme de cryptographie **asymétrique** encore très utilisé et décrit dans les années 70.





Adi **S**hamir, Ronald **R**ivest et Leonard **A**dleman

Par tradition, on appelle **Alice**  la personne qui désire  
recevoir un message chiffré, et **Bob**  la personne qui envoie  
le message.





Par tradition, on appelle **Alice**  la personne qui désire recevoir un message chiffré, et **Bob**  la personne qui envoie le message.



- Alice choisit deux grands entiers naturels premiers  $p$  et  $q$  et fait leur produit  $n = pq$ .

Par tradition, on appelle **Alice**  la personne qui désire recevoir un message chiffré, et **Bob**  la personne qui envoie le message.



- Alice choisit deux grands entiers naturels premiers  $p$  et  $q$  et fait leur produit  $n = pq$ .
- Elle choisit un entier  $c$  premier avec  $(p - 1)(q - 1)$ .

Par tradition, on appelle **Alice**  la personne qui désire recevoir un message chiffré, et **Bob**  la personne qui envoie le message.

- Alice choisit deux grands entiers naturels premiers  $p$  et  $q$  et fait leur produit  $n = pq$ .
- Elle choisit un entier  $c$  premier avec  $(p - 1)(q - 1)$ .
- Elle publie dans un annuaire sa clef publique :  $(n, c)$ .

Par tradition, on appelle **Alice**  la personne qui désire recevoir un message chiffré, et **Bob**  la personne qui envoie le message.

- Alice choisit deux grands entiers naturels premiers  $p$  et  $q$  et fait leur produit  $n = pq$ .
- Elle choisit un entier  $c$  premier avec  $(p - 1)(q - 1)$ .
- Elle publie dans un annuaire sa clef publique :  $(n, c)$ .
- Elle calcule à partir de  $p$  et  $q$ , qu'elle a gardés secrets, la clef  $d$  de déchiffrement (c'est sa clef privée). Celle-ci doit satisfaire l'équation  $cd \equiv 1 \pmod{(p - 1)(q - 1)}$

Par tradition, on appelle **Alice**  la personne qui désire recevoir un message chiffré, et **Bob**  la personne qui envoie le message.

- Alice choisit deux grands entiers naturels premiers  $p$  et  $q$  et fait leur produit  $n = pq$ .
- Elle choisit un entier  $c$  premier avec  $(p - 1)(q - 1)$ .
- Elle publie dans un annuaire sa clef publique :  $(n, c)$ .
- Elle calcule à partir de  $p$  et  $q$ , qu'elle a gardés secrets, la clef  $d$  de déchiffrement (c'est sa clef privée). Celle-ci doit satisfaire l'équation  $cd \equiv 1 \pmod{(p - 1)(q - 1)}$  (Euclide)

Ensuite  $p$  et  $q$  sont oubliés (c'est sur ce fait que repose la sécurité).

- Si Bob veut envoyer le message (nombre)  $m$  à Alice, il calcule

$$m' \equiv m^c [n]$$

Ensuite  $p$  et  $q$  sont oubliés (c'est sur ce fait que repose la sécurité).

- Si Bob veut envoyer le message (nombre)  $m$  à Alice, il calcule

$$m' \equiv m^c [n]$$

- Alice reçoit alors  $m'$  qu'elle convertit en

$$m'' \equiv m'^d [n]$$

Ensuite  $p$  et  $q$  sont oubliés (c'est sur ce fait que repose la sécurité).

- Si Bob veut envoyer le message (nombre)  $m$  à Alice, il calcule

$$m' \equiv m^c [n]$$

- Alice reçoit alors  $m'$  qu'elle convertit en

$$m'' \equiv m'^d [n]$$

**Propriété mathématique :**

On montre que  $m \equiv m'' [n]$  c'est à dire  $m \equiv m^{c \times d} [n]$



message  
initial

$m$



message  
codé

$m^c$



message  
décodé

$m^{cd} \equiv m$

message  
initial

$m$



message  
codé

$m^c$



message  
décodé

$m^{cd} \equiv m$

D'où le terme de **chiffrement asymétrique**  
la clé de chiffrement n'est pas celle de déchiffrement.

Un exemple pour comprendre :  
Création des clés :

- $p = 17$  et  $q = 13$

Un exemple pour comprendre :

Création des clés :

- $p = 17$  et  $q = 13$
- $N = 17 \times 13 = 221$  (on peut coder 221 nombres de 0 à 220).

Un exemple pour comprendre :

Création des clés :

- $p = 17$  et  $q = 13$
- $N = 17 \times 13 = 221$  (on peut coder 221 nombres de 0 à 220).
- $(p - 1)(q - 1) = 16 \times 12 = 192 = 2^6 \times 3$

Un exemple pour comprendre :

Création des clés :

- $p = 17$  et  $q = 13$
- $N = 17 \times 13 = 221$  (on peut coder 221 nombres de 0 à 220).
- $(p - 1)(q - 1) = 16 \times 12 = 192 = 2^6 \times 3$
- Si on choisit  $c = 5 \times 7 = 35$  qui est premier avec 192

Un exemple pour comprendre :

Création des clés :

- $p = 17$  et  $q = 13$
- $N = 17 \times 13 = 221$  (on peut coder 221 nombres de 0 à 220).
- $(p - 1)(q - 1) = 16 \times 12 = 192 = 2^6 \times 3$
- Si on choisit  $c = 5 \times 7 = 35$  qui est premier avec 192
- Cherchons  $d$  tel que  $cd \equiv 1[192]$ ... $d = 11$  convient.

Alice donne sa clé publique  $(n, c) = (221, 35)$  et conserve secrètement sa clé privée :  $d = 11$ . Bob veut envoyer le message  $m = 18$ .

- $18^2 = 324 \equiv 103[221]$ ,



Alice donne sa clé publique  $(n, c) = (221, 35)$  et conserve secrètement sa clé privée :  $d = 11$ . Bob veut envoyer le message  $m = 18$ .

- $18^2 = 324 \equiv 103[221]$ , et  $18^4 \equiv 103^2 \equiv 10609 \equiv 1[221]$ .

Alice donne sa clé publique  $(n, c) = (221, 35)$  et conserve secrètement sa clé privée :  $d = 11$ . Bob veut envoyer le message  $m = 18$ .

- $18^2 = 324 \equiv 103[221]$ , et  $18^4 \equiv 103^2 \equiv 10609 \equiv 1[221]$ .
- Ainsi  $m' = 18^{35} \equiv 18^{4 \times 8 + 3} \equiv 18^3 \equiv 86[221]$

Alice donne sa clé publique  $(n, c) = (221, 35)$  et conserve secrètement sa clé privée :  $d = 11$ . Bob veut envoyer le message  $m = 18$ .

- $18^2 = 324 \equiv 103[221]$ , et  $18^4 \equiv 103^2 \equiv 10609 \equiv 1[221]$ .
- Ainsi  $m' = 18^{35} \equiv 18^{4 \times 8 + 3} \equiv 18^3 \equiv 86[221]$

Alice reçoit  $m' = 86$  qu'elle décode :

- $86^2 \equiv 103[221]$ ;  $86^3 \equiv 18[221]$ ;  $86^4 \equiv 1[221]$ .

Alice donne sa clé publique  $(n, c) = (221, 35)$  et conserve secrètement sa clé privée :  $d = 11$ . Bob veut envoyer le message  $m = 18$ .

- $18^2 = 324 \equiv 103[221]$ , et  $18^4 \equiv 103^2 \equiv 10609 \equiv 1[221]$ .
- Ainsi  $m' = 18^{35} \equiv 18^{4 \times 8 + 3} \equiv 18^3 \equiv 86[221]$

Alice reçoit  $m' = 86$  qu'elle décode :

- $86^2 \equiv 103[221]$ ;  $86^3 \equiv 18[221]$ ;  $86^4 \equiv 1[221]$ .
- Ainsi  $m'' = 86^{11} = 86^{4 \times 2 + 3} \equiv 86^3 \equiv 18[221]$

Côté programmation, c'est très simple a priori :

- Codage du message :

```
let encode m c n = (puiss m c) mod n;;
```

Côté programmation, c'est très simple a priori :

- Codage du message :

```
let encode m c n = (puiss m c) mod n;;
```

- Décodage du message :

```
let decode = encode
```

Côté programmation, c'est très simple a priori :

- Codage du message :

```
let encode m c n = (puiss m c) mod n;;
```

- Décodage du message :

```
let decode = encode
```

Mais ça ne fonctionne pas....

```
let m1 = encode 18 35 221;;  
val m1 : int = -138
```

## Exercice

Écrire une fonction qui calcule  $a^n[m]$  en donnant pour réponse le représentant dans  $\{0, \dots, m - 1\}$ .





## Exercice



Écrire une fonction qui calcule  $a^n[m]$  en donnant pour réponse le représentant dans  $\{0, \dots, m - 1\}$ .

```
let puissM a n m =  
  let rec puissR res a = function  
    | 0 -> res  
    | n when n mod 2 == 0 ->  
      puissR res (a*a mod m) (n/2)  
    | n -> puissR (res*a mod m) (a*a mod m) (n/2)  
  in puissR 1 a n  
;;  
let encode = puissM;;  
let decode = encode
```

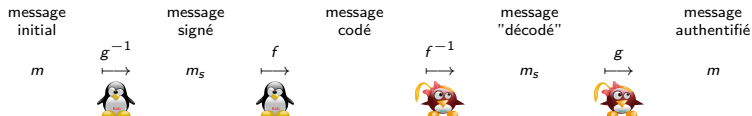
Encore plus fort, Alice et Bob peuvent **authentifier** leurs échanges

	clé publique	clé privée	fonction de codage
	$(n_A; c_A)$	$d_A$	$f$
	$(n_B; c_B)$	$d_B$	$g$

Encore plus fort, Alice et Bob peuvent **authentifier** leurs échanges

	clé publique	clé privée	fonction de codage
	$(n_A; c_A)$	$d_A$	$f$
	$(n_B; c_B)$	$d_B$	$g$

Si  $n_B < n_A$ , on choisit  $m \in \{0, \dots, n_B - 1\}$  :



On cherche à multiplier deux grands entiers naturels :

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 2 & 6 & 0 & 9 & 3 & 2 & 8 & 5 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 4 & 5 & 7 & 8 & 9 & 8 & 1 & 2 & 4 \\ \hline \end{array}$$
  
$$= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \\ \hline \end{array}$$

Où les nombres sont représentés par des tableaux  
ou des listes de chiffres

On cherche à multiplier deux grands entiers naturels :

- En base 10,

tout entier s'écrit sous la forme  $\sum_{k=0}^N a_k 10^k$  où  $a_k \in \{0, \dots, 9\}$

On cherche à multiplier deux grands entiers naturels :

- En base 10,

tout entier s'écrit sous la forme  $\sum_{k=0}^N a_k 10^k$  où  $a_k \in \{0, \dots, 9\}$

- En base 2,

tout entier s'écrit sous la forme  $\sum_{k=0}^N a_k 2^k$  où  $a_k \in \{0, 1\}$

Notre objectif se traduit par :

$$N_1 \times N_2 = \left( \sum_{k=0}^{n_1} a_k 10^k \right) \left( \sum_{k=0}^{n_2} b_k 10^k \right) = \sum_{k=0}^{n_1+n_2} c_k 10^k$$

Notre objectif se traduit par :

$$N_1 \times N_2 = \left( \sum_{k=0}^{n_1} a_k 10^k \right) \left( \sum_{k=0}^{n_2} b_k 10^k \right) = \sum_{k=0}^{n_1+n_2} c_k 10^k$$

On généralise en cherchant une manière efficace de multiplier deux polynômes de degré  $n$  :  $P(X) = \sum_{k=0}^n a_k X^k$  et  $Q(X) = \sum_{k=0}^n b_k X^k$ , modélisés par des listes  $[a_0, a_1, \dots, a_n]$  et  $[b_0, b_1, \dots, b_n]$



Calcul classique du produit (convention :  $a_j = b_j = 0$  si  $j > n$ ) :

$$P(X)Q(X) = \sum_{k=0}^{2n} c_k X^k \text{ avec } c_k = \sum_{j=0}^k a_j b_{k-j}$$

Calcul classique du produit (convention :  $a_j = b_j = 0$  si  $j > n$ ) :

$$P(X)Q(X) = \sum_{k=0}^{2n} c_k X^k \text{ avec } c_k = \sum_{j=0}^k a_j b_{k-j}$$

Nombre d'opérations élémentaires :

- calcul de  $c_k$  :  $k + 1$  multiplications et  $k$  additions

Calcul classique du produit (convention :  $a_j = b_j = 0$  si  $j > n$ ) :

$$P(X)Q(X) = \sum_{k=0}^{2n} c_k X^k \text{ avec } c_k = \sum_{j=0}^k a_j b_{k-j}$$

Nombre d'opérations élémentaires :

- calcul de  $c_k$  :  $k + 1$  multiplications et  $k$  additions

- Au total  $\sum_{k=0}^{2n} (2k + 1) = (2n + 1)^2 \sim 4n^2 = \theta(n^2)$

Calcul classique du produit (convention :  $a_j = b_j = 0$  si  $j > n$ ) :

$$P(X)Q(X) = \sum_{k=0}^{2n} c_k X^k \text{ avec } c_k = \sum_{j=0}^k a_j b_{k-j}$$

Nombre d'opérations élémentaires :

- calcul de  $c_k$  :  $k + 1$  multiplications et  $k$  additions

- Au total  $\sum_{k=0}^{2n} (2k + 1) = (2n + 1)^2 \sim 4n^2 = \theta(n^2)$

Si on ne compte que les  $a_k b_{n-k}$  non nuls, on améliore en  $2n^2 + 2n + 1$  opérations soit encore du  $\theta(n^2)$

On cherche à optimiser le nombre d'opérations selon le principe du diviser pour régner. En remarquant que si  $P$  et  $Q$  sont deux polynômes de degrés strictement inférieurs à  $2n$  alors

$$P = P_1 + X^n P_2 \text{ et } Q = Q_1 + X^n Q_2$$

où les degrés de  $P_1, P_2, Q_1$  et  $Q_2$  sont strictement inférieur à  $n$ .

On cherche à optimiser le nombre d'opérations selon le principe du diviser pour régner. En remarquant que si  $P$  et  $Q$  sont deux polynômes de degrés strictement inférieurs à  $2n$  alors

$$P = P_1 + X^n P_2 \text{ et } Q = Q_1 + X^n Q_2$$

où les degrés de  $P_1, P_2, Q_1$  et  $Q_2$  sont strictement inférieur à  $n$ .

$$\begin{aligned} \text{Exemple : } P(X) &= 5 - x^2 + 2X^3 - 3X^4 + 2X^6 + X^7 \\ &= \underbrace{5 - x^2 + 2X^3}_{P_1} + \underbrace{X^4(-3 + 2X^2 + X^3)}_{P_2} \end{aligned}$$

$$P = P_1 + X^n P_2 \text{ et } Q = Q_1 + X^n Q_2$$

## Exercice

- 1 Écrire une formule qui réduit la multiplication des polynômes  $P$  et  $Q$  de degrés strictement inférieurs à  $2n$  en multiplications de polynômes de degrés strictement inférieurs à  $n$ .
- 2 On note  $C(n)$  le coût en terme d'opérations élémentaires nécessaires à calculer le produit de deux polynômes de degrés  $n$ . Justifier que  $C(n) = 4C(n/2) + O(n)$ .
- 3 En posant  $\alpha_\ell = \frac{C(2^\ell)}{4^\ell}$ . Donner une relation en  $\alpha_{\ell+1}$  et  $\alpha_\ell$ .
- 4 En déduire l'ordre de  $C(n)$ .

Si  $P = P_1 + X^n P_2$  et  $Q = Q_1 + X^n Q_2$ , alors :

- $P \times Q = (P_1 + X^n P_2)(Q_1 + X^n Q_2)$   
 $= P_1 Q_1 X^{2n} + (P_2 Q_1 + P_1 Q_2) X^n + P_2 Q_2$
- Le produit par  $X^{2n}$  ne nécessite pas de calculs (il suffit de "décaler" les termes). Il y a 4 multiplications avec des polynômes de degrés inférieurs à  $n$  et  $2n + 3n + 4n = 9n$  additions ensuite.

$$C(2n) = 4C(n) + 9n$$

ou

$$C(n) = 4C(n/2) + \frac{9}{2}n$$



$$3. \alpha_{\ell+1} = \frac{C(2^{\ell+1})}{4^{\ell+1}} = \frac{4C(2^\ell) + 9 \cdot 2^\ell}{4^{\ell+1}} = \alpha_\ell + \frac{9}{4} \left(\frac{1}{2}\right)^\ell$$

4. En sommant l'égalité précédente :

$$\sum_{\ell=0}^{k-1} \alpha_{\ell+1} = \sum_{\ell=0}^{k-1} \alpha_\ell + \sum_{\ell=0}^{k-1} \frac{9}{4} \left(\frac{1}{2}\right)^\ell$$

$$\alpha_k = \alpha_0 + \frac{9}{2} (1 - (1/2)^k) \text{ avec } \alpha_0 = \frac{C(2^0)}{4^0} = C(1) = 1.$$

$$\text{Pour } n = 2^k, C(n) = \frac{11}{2}n^2 - \frac{9}{2}n \sim \frac{11}{2}n^2 = \theta(n^2).$$

Ce n'est donc pas mieux...

On peut raffiner cette méthode avec la remarque suivante de **Karatsuba** : le terme intermédiaire de  $PQ$  s'écrit

$$P_1Q_2 + P_2Q_1 = (P_1 + P_2)(Q_1 + Q_2) - P_1Q_1 - P_2Q_2.$$

## Exercice

- 1 En quoi est-ce mieux ?
- 2 En s'inspirant de la démarche précédente, trouver la formule de récurrence qui définit la complexité de la multiplication de Karatsuba. Quelle est sa solution ?

1.  $C(2n) = 3C(n) + 13n$  ce qui est mieux a priori (plus d'additions, mais moins de multiplications)
2. Cette fois  $C(n) = \theta\left(n^{\frac{\ln 3}{\ln 2}}\right) = \theta\left(n^\beta\right)$  avec  $\beta \approx 1,585$ .

C'est donc mieux !

Principe : Le tri par partition-fusion (merge sort en anglais) implémente une approche de type diviser pour régner très simple :

- la suite à trier est tout d'abord **scindée** en deux suites de longueurs égales (à un élément près).
- Ces deux suites sont ensuite **triées séparément** avant d'être **fusionnées**.

L'efficacité du tri par fusion vient de l'efficacité de la fusion : le principe consiste à parcourir simultanément les deux suites triées dans l'ordre croissant de leurs éléments en extrayant chaque fois l'élément le plus petit.

L'algorithme :

```
Fonction TriParFusion( $S$ )  
  SI  $|S| > 1$  ALORS  
     $(S_1, S_2) \leftarrow \text{Scinder}(S)$   
     $S_1 \leftarrow \text{TriParFusion}(S_1)$   
     $S_2 \leftarrow \text{TriParFusion}(S_2)$   
     $S \leftarrow \text{Fusion}(S_1, S_2)$   
  RENVOYER  $S$ 
```

5 ; 1 ; 6 ; 3 ; 2 ; 7 ; 8

Fonction TriParFusion( $S$ )

SI  $|S| > 1$  ALORS

$(S_1, S_2) \leftarrow \text{Scinder}(S)$

$S_1 \leftarrow \text{TriParFusion}(S_1)$

$S_2 \leftarrow \text{TriParFusion}(S_2)$

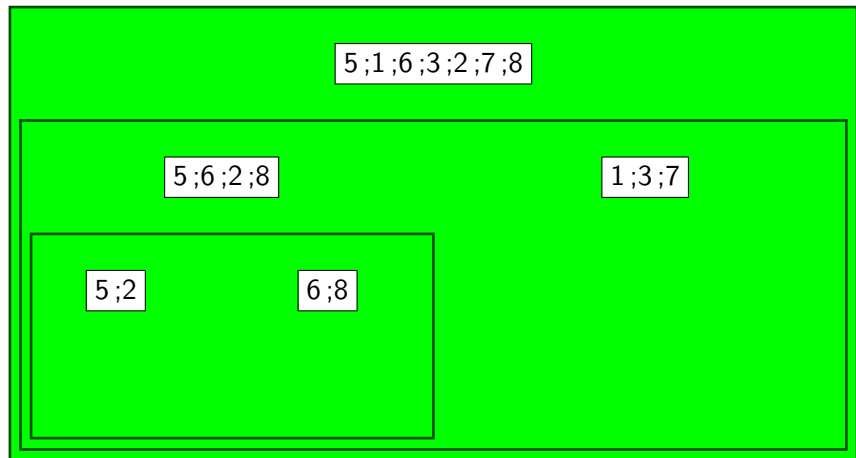
$S \leftarrow \text{Fusion}(S_1, S_2)$

RENOYER  $S$

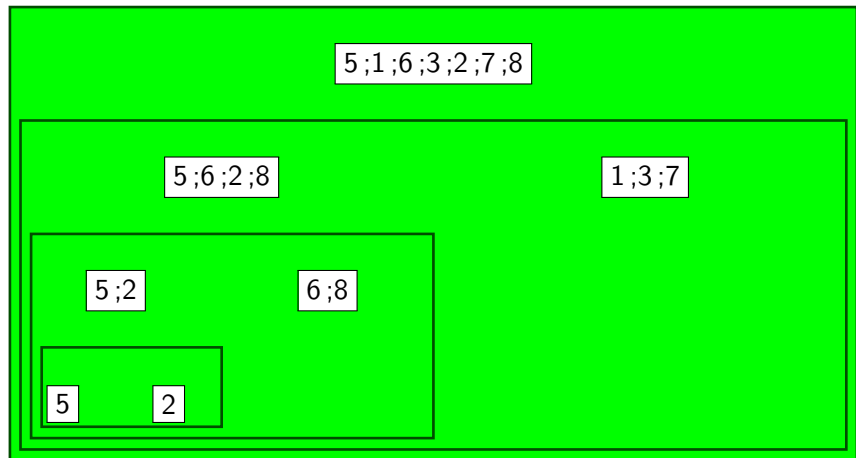
5;1;6;3;2;7;8

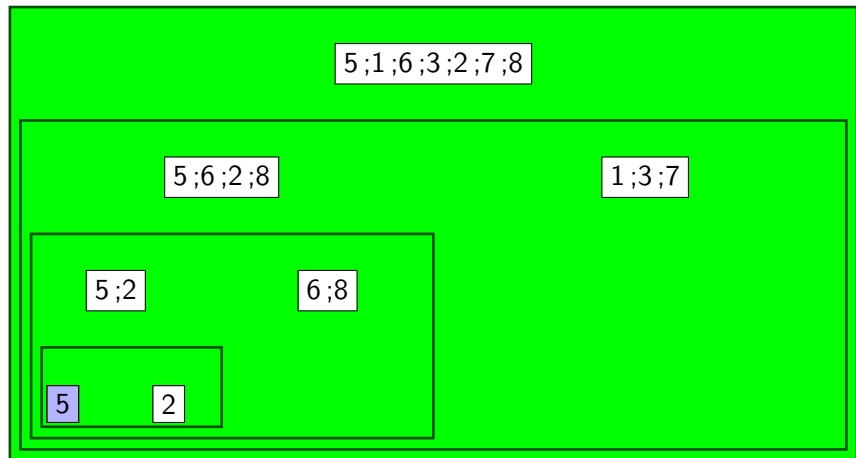
5;6;2;8

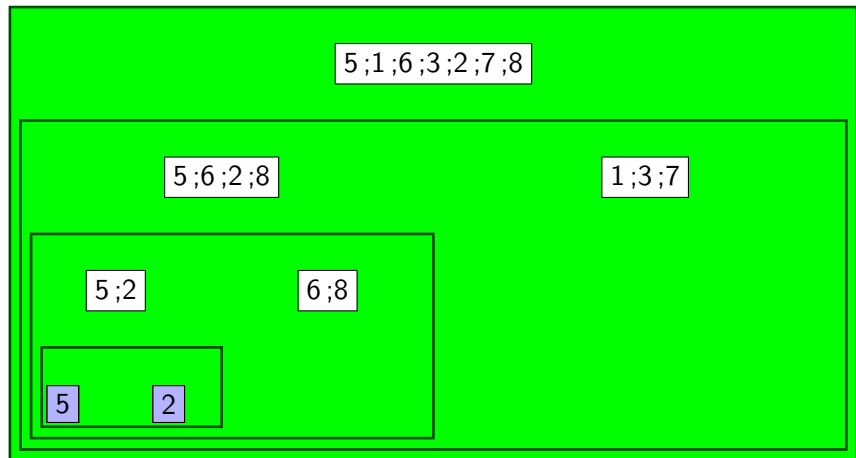
1;3;7

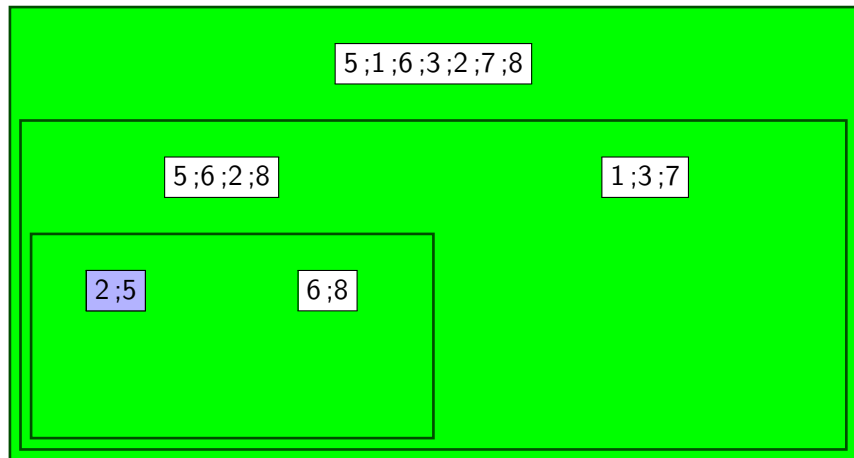


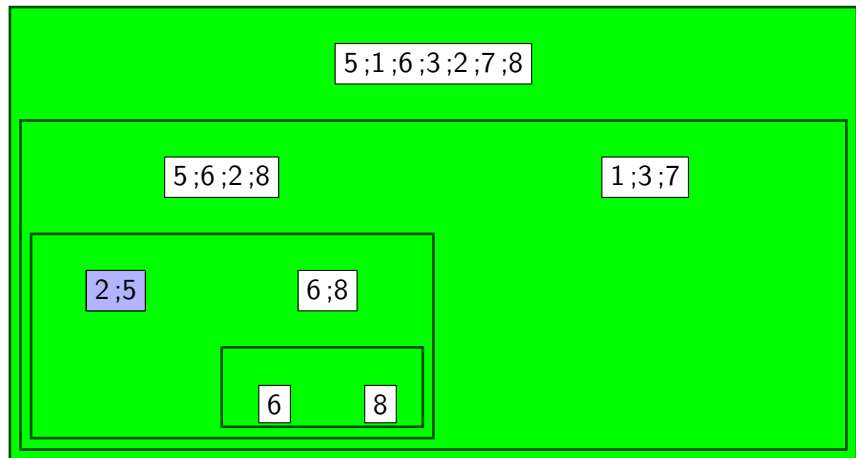


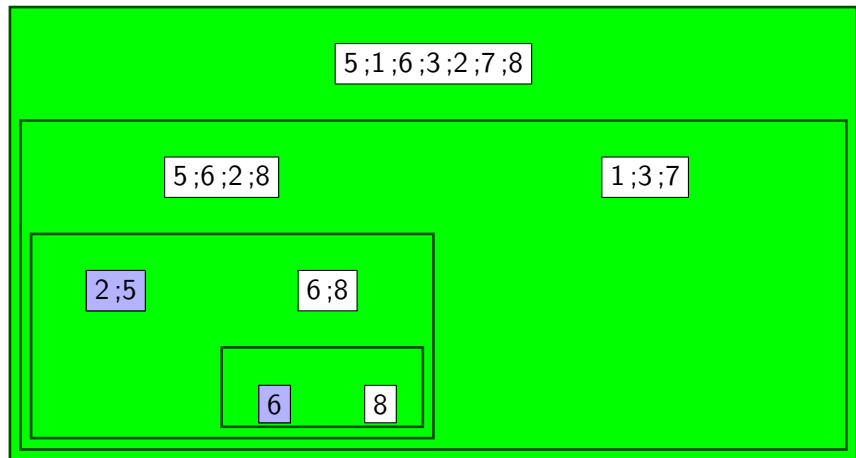


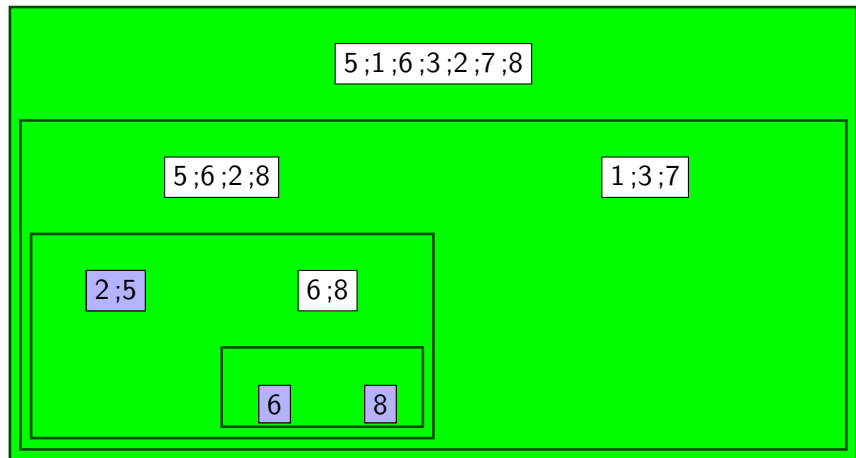


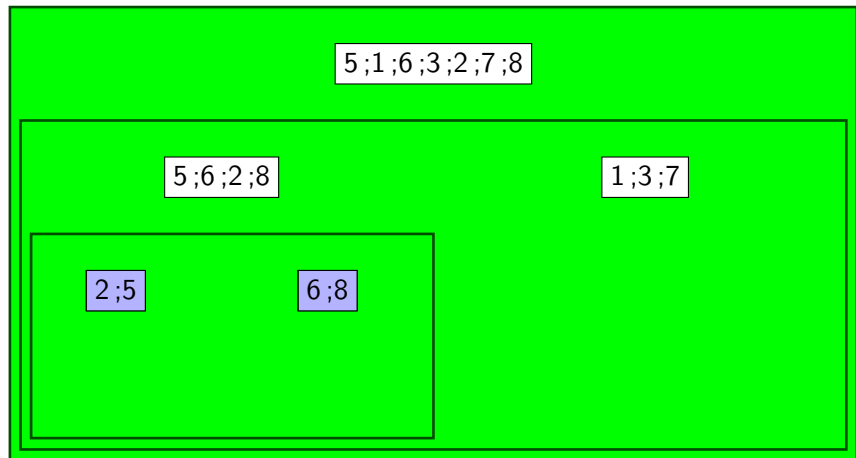










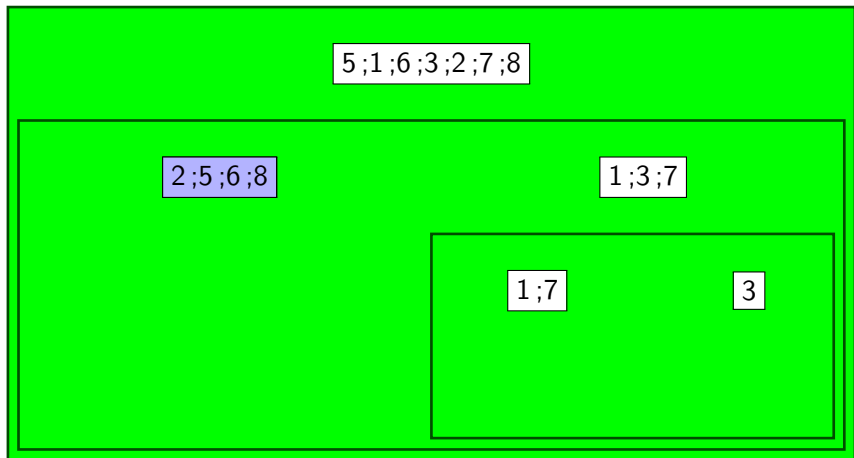


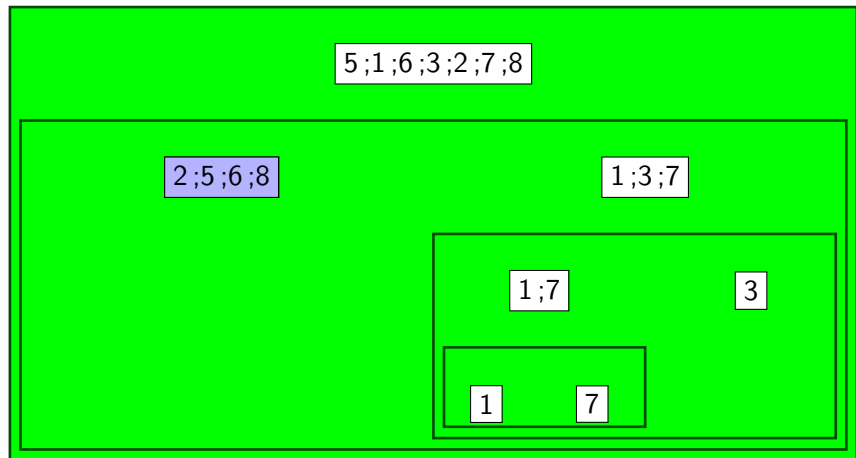


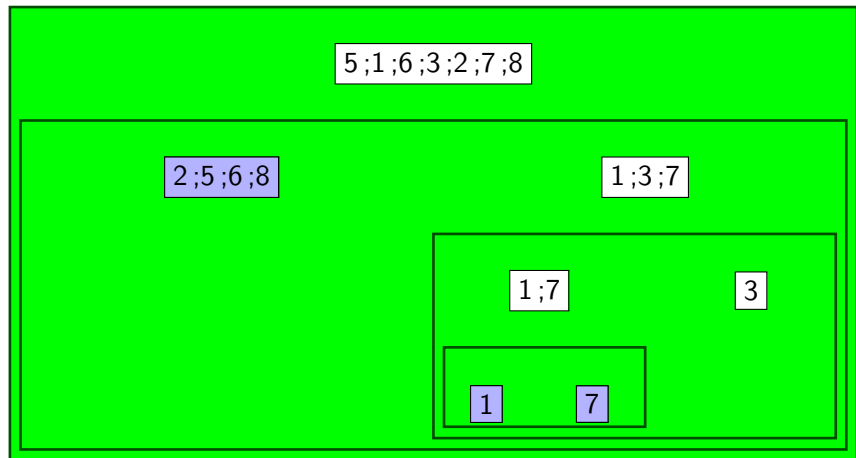
5;1;6;3;2;7;8

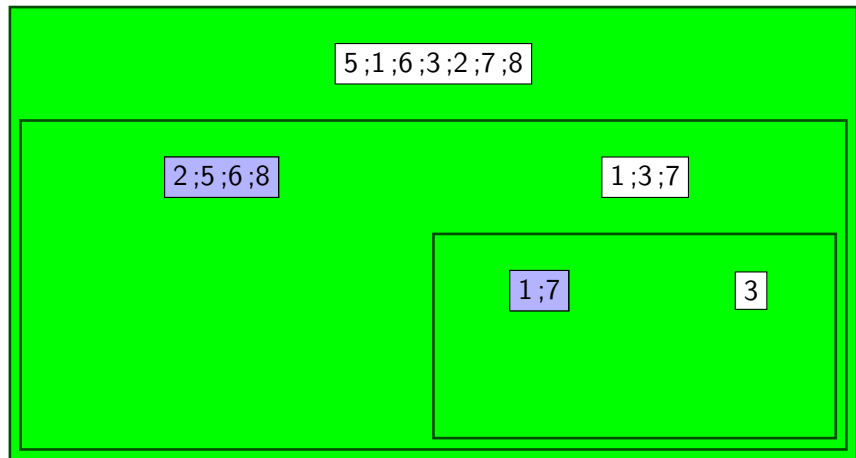
2;5;6;8

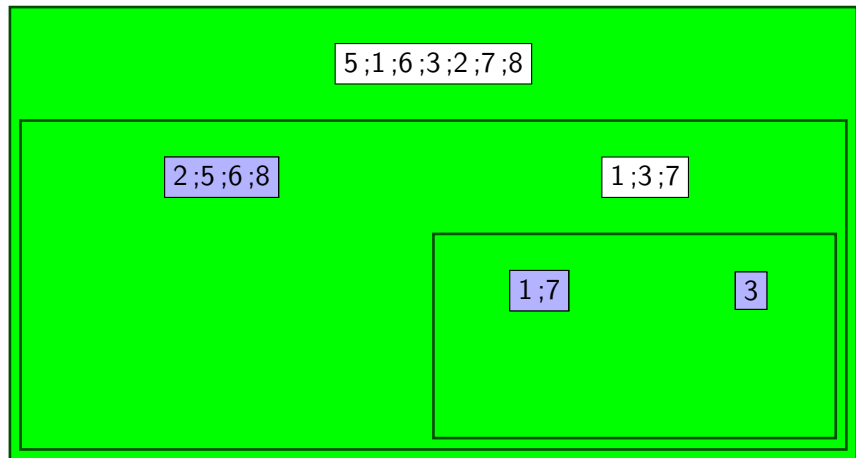
1;3;7











5;1;6;3;2;7;8

2;5;6;8

1;3;7

1;2;3;5;6;7;8

FIN



## Calcul de la complexité

### Exercice

Soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

- 1 Estimer le nombre de comparaisons pour :
  - 1 Scinder
  - 2 Trier
  - 3 Fusionner
- 2 En déduire une relation entre  $T(n)$  et  $T(n/2)$
- 3 Donner l'ordre de  $T(n)$

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments

9;1;8;4;7;3

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments  
9;1;8;4;7;3

Scinder

$n/2$  éléments  
9;8;7

$n/2$  éléments  
1;4;3

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments  
9;1;8;4;7;3

0 comparaison **Scinder**

$n/2$  éléments  
9;8;7

$n/2$  éléments  
1;4;3

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments  
9;1;8;4;7;3

0 comparaison **Scinder**

Trier

$n/2$  éléments  
7;8;9

$n/2$  éléments  
1;3;4

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments  
9;1;8;4;7;3

0 comparaison **Scinder**

$2 \times T\left(\frac{n}{2}\right)$  comparaisons **Trier**

$n/2$  éléments  
7;8;9

$n/2$  éléments  
1;3;4

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments

1;3;4;7;8;9

0 comparaison **Scinder**

$2 \times T\left(\frac{n}{2}\right)$  comparaisons **Trier**

$n/2$  éléments

7;8;9

$n/2$  éléments

1;3;4

**Fusionner**

Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments

1;3;4;7;8;9

0 comparaison **Scinder**

$2 \times T\left(\frac{n}{2}\right)$  comparaisons **Trier**

$n/2$  éléments

7;8;9

$n/2$  éléments

1;3;4

$n$  comparaisons **Fusionner**



Si  $n$  est pair, soit  $T(n)$  le nombre de comparaisons pour trier une liste à  $n$  éléments.

$n$  éléments

1;3;4;7;8;9

0 comparaison **Scinder**

$2 \times T\left(\frac{n}{2}\right)$  comparaisons **Trier**

$n$  comparaisons **Fusionner**

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Posons  $u_\ell = \frac{T(2^\ell)}{2^\ell}$ .

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1)$$

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}}$$

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}}$$

Posons  $u_\ell = \frac{T(2^\ell)}{2^\ell}$ .  $u_0 = \frac{T(2^0)}{2^0} = T(1) = 0$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1$$

Posons  $u_\ell = \frac{T(2^\ell)}{2^\ell}$ .  $u_0 = \frac{T(2^0)}{2^0} = T(1) = 0$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique,

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique, ainsi

$$\forall \ell \in \mathbb{N}, u_\ell = u_0 + \ell \times 1 = \ell.$$



$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique, ainsi

$$\forall \ell \in \mathbb{N}, u_\ell = u_0 + \ell \times 1 = \ell.$$

$$\forall \ell \in \mathbb{N}, T(2^\ell) = 2^\ell u_\ell$$

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique, ainsi

$$\forall \ell \in \mathbb{N}, u_\ell = u_0 + \ell \times 1 = \ell.$$

$$\forall \ell \in \mathbb{N}, T(2^\ell) = 2^\ell u_\ell = \ell 2^\ell.$$

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique, ainsi

$$\forall \ell \in \mathbb{N}, u_\ell = u_0 + \ell \times 1 = \ell.$$

$$\forall \ell \in \mathbb{N}, T(2^\ell) = 2^\ell u_\ell = \ell 2^\ell. \text{ Or } n = 2^\ell \iff \ell = \log_2 n.$$

$$\text{Posons } u_\ell = \frac{T(2^\ell)}{2^\ell}. \quad u_0 = \frac{T(2^0)}{2^0} = T(1) = 0.$$

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{T(2^{\ell+1})}{2^{\ell+1}} = \frac{2T(2^\ell) + 2^{\ell+1}}{2^{\ell+1}} = \frac{T(2^\ell)}{2^\ell} + 1 = u_\ell + 1$$

$(u_\ell)$  est donc une suite arithmétique, ainsi

$$\forall \ell \in \mathbb{N}, u_\ell = u_0 + \ell \times 1 = \ell.$$

$$\forall \ell \in \mathbb{N}, T(2^\ell) = 2^\ell u_\ell = \ell 2^\ell. \text{ Or } n = 2^\ell \iff \ell = \log_2 n.$$

Ainsi, si  $n$  est une puissance de 2, on a montré que

$$T(n) = n \log_2 n$$

$\forall n \in \mathbb{N}^*$ ,

- D'une part  $\exists m$ , puissance de 2 telle que  $m \leq n < 2m$ .

$\forall n \in \mathbb{N}^*$ ,

- D'une part  $\exists m$ , puissance de 2 telle que  $m \leq n < 2m$ .
- Assez intuitivement,  $T(m) \leq T(n) \leq T(2m)$

$\forall n \in \mathbb{N}^*$ ,

- D'une part  $\exists m$ , puissance de 2 telle que  $m \leq n < 2m$ .
- Assez intuitivement,  $T(m) \leq T(n) \leq T(2m)$
- $T(m) = m \log_2 m$  et  $T(2m) = 2m \log_2(2m)$ .

$\forall n \in \mathbb{N}^*$ ,

- D'une part  $\exists m$ , puissance de 2 telle que  $m \leq n < 2m$ .
- Assez intuitivement,  $T(m) \leq T(n) \leq T(2m)$
- $T(m) = m \log_2 m$  et  $T(2m) = 2m \log_2(2m)$ .

$$\frac{T(m)}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{T(2m)}{n \ln n}$$



$\forall n \in \mathbb{N}^*$ ,

- D'une part  $\exists m$ , puissance de 2 telle que  $m \leq n < 2m$ .
- Assez intuitivement,  $T(m) \leq T(n) \leq T(2m)$
- $T(m) = m \log_2 m$  et  $T(2m) = 2m \log_2(2m)$ .

$$\frac{T(m)}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{T(2m)}{n \ln n}$$

Donc,

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n}$$

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n} \text{ et } m \leq n < 2m$$

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n} \text{ et } m \leq n < 2m$$

Or,

$$\bullet \frac{m \log_2(m)}{n \ln n} > \frac{n/2 \log_2(n/2)}{n \ln n} = \underbrace{\frac{\log_2(n) - 1}{2 \ln n}}_{u_n}$$

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n} \text{ et } m \leq n < 2m$$

Or,

$$\bullet \frac{m \log_2(m)}{n \ln n} > \frac{n/2 \log_2(n/2)}{n \ln n} = \underbrace{\frac{\log_2(n) - 1}{2 \ln n}}_{u_n}$$

$$u_n = \frac{1}{2 \ln 2} - \frac{1}{2 \ln n} \rightarrow \frac{1}{2 \ln 2}$$

Cette suite converge et donc est minorée par un réel  $M_1$ .

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n} \text{ et } m \leq n < 2m$$

Or,

$$\bullet \frac{m \log_2(m)}{n \ln n} > \frac{n/2 \log_2(n/2)}{n \ln n} = \underbrace{\frac{\log_2(n) - 1}{2 \ln n}}_{u_n}$$

$$u_n = \frac{1}{2 \ln 2} - \frac{1}{2 \ln n} \rightarrow \frac{1}{2 \ln 2}$$

Cette suite converge et donc est minorée par un réel  $M_1$ .

$$\bullet \frac{2m \log_2(2m)}{n \ln n} = \frac{2m \log_2(m) + 2m}{n \ln n} \leq \underbrace{\frac{2n \log_2(n) + 2n}{n \ln n}}_{v_n}$$

$$\frac{m \log_2 m}{n \ln n} \leq \frac{T(n)}{n \ln n} \leq \frac{2m \log_2(2m)}{n \ln n} \text{ et } m \leq n < 2m$$

Or,

$$\bullet \frac{m \log_2(m)}{n \ln n} > \frac{n/2 \log_2(n/2)}{n \ln n} = \underbrace{\frac{\log_2(n) - 1}{2 \ln n}}_{u_n}$$

$$u_n = \frac{1}{2 \ln 2} - \frac{1}{2 \ln n} \rightarrow \frac{1}{2 \ln 2}$$

Cette suite converge et donc est minorée par un réel  $M_1$ .

$$\bullet \frac{2m \log_2(2m)}{n \ln n} = \frac{2m \log_2(m) + 2m}{n \ln n} \leq \underbrace{\frac{2n \log_2(n) + 2n}{n \ln n}}_{v_n}$$

$$v_n = \frac{2n \log_2(n)}{n \ln n} + \frac{2n}{n \ln n} \rightarrow \frac{2}{\ln 2}$$

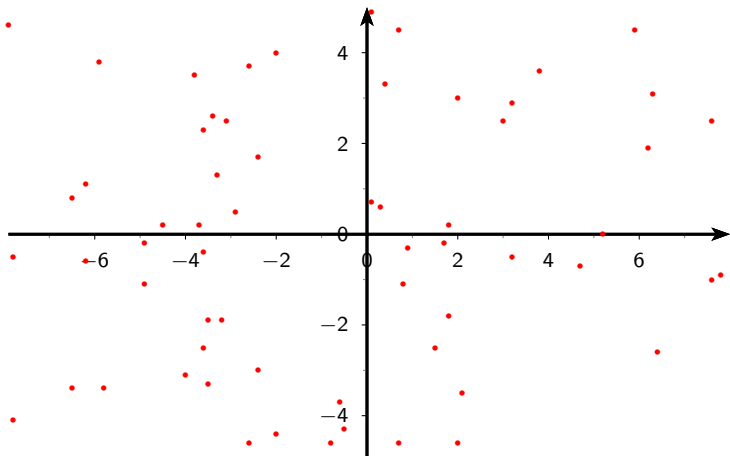
Cette suite converge et donc est majorée par un réel  $M_2$ .

Conclusion :

$$\exists M_1, M_2 > 0, \forall n \in \mathbb{N}^*, M_1 \leq \frac{T(n)}{n \ln n} \leq M_2$$

C'est-à-dire

$$T(n) = \theta(n \ln n)$$



Quels sont les 2 points les plus proches ? A quelle distance sont-ils ?



## Exercice

Si les points sont regroupés en tableau de coordonnées, proposer une solution naïve et calculer sa complexité.

## Exercice

Si les points sont regroupés en tableau de coordonnées, proposer une solution naïve et calculer sa complexité.

Il y a  $\binom{n}{2}$  paires de points, si on cherche la plus petite distance, il y aura donc  $\frac{n(n-1)}{2}$  calculs et comparaisons.

## Exercice

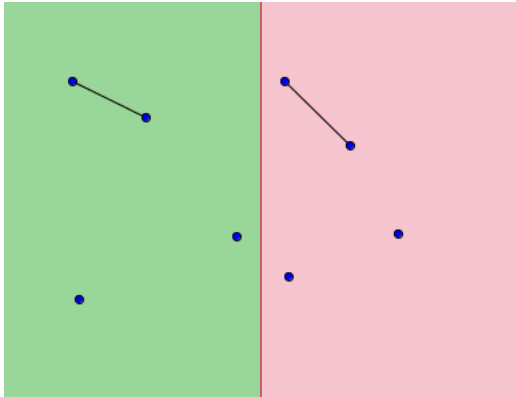
Si les points sont regroupés en tableau de coordonnées, proposer une solution naïve et calculer sa complexité.

Il y a  $\binom{n}{2}$  paires de points, si on cherche la plus petite distance, il y aura donc  $\frac{n(n-1)}{2}$  calculs et comparaisons.

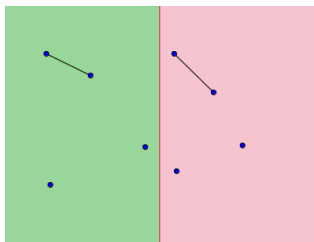
La complexité est en  $\theta(n^2)$ .

Peut-on améliorer l'algorithme  
avec une méthode diviser pour régner ?

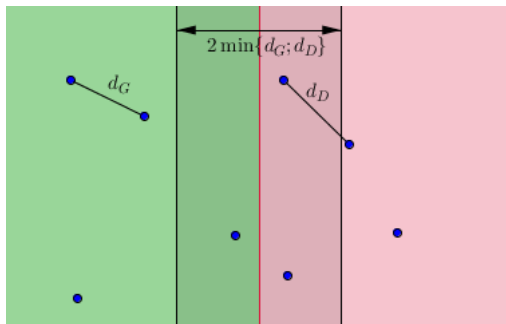
## Diviser



- Nécessité d'avoir un tableau  $TX$  de points triés par abscisses croissantes
- On continue les appels récursifs avec les tableaux  $TX_G$  et  $TX_D$
- Lorsque qu'il y a 2 ou 3 points, on résout le problème naïvement.



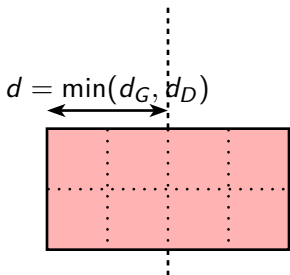
Comment fusionner les résultats obtenus ?



## Exercice

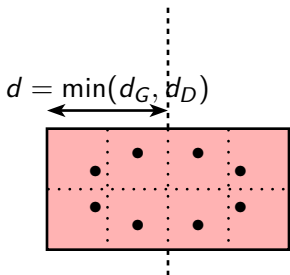
Pourquoi ne pas chercher en dehors de cette bande ?





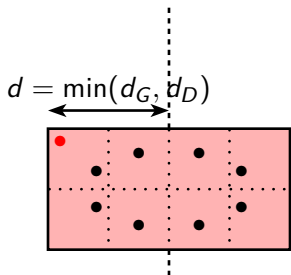
## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



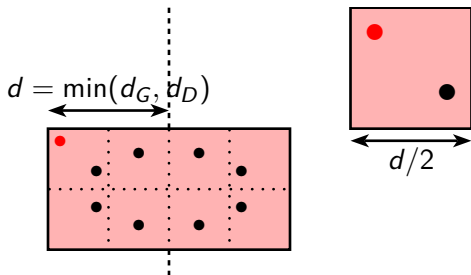
## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



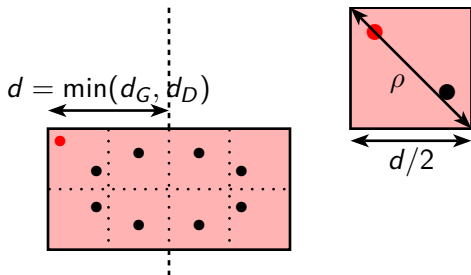
## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



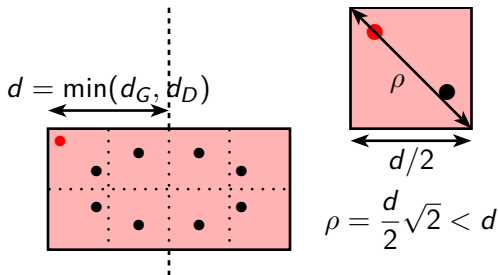
## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



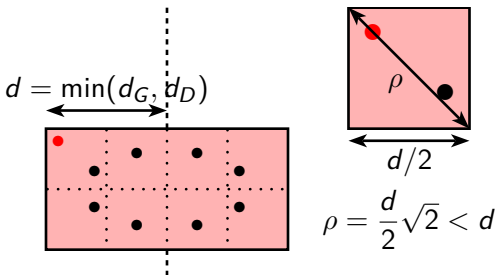
## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



## Exercice

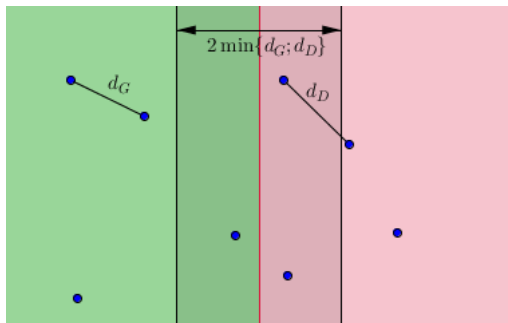
On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



I  
M  
P  
O  
S  
S  
I  
B  
L  
E

## Exercice

On pose  $d = \min(d_G; d_D)$ . Dans ce rectangle, centré sur la valeur médiane des abscisses, de longueur  $2d$  et de largeur  $d$ , pourquoi ne peut-il pas y avoir plus de 8 points ?



Nous avons donc besoin d'un autre tableau  $TY$  contenant les points classés par ordre d'ordonnées croissantes....



En résumé, voici l'algorithme :

- Entrées : les tableaux de points  $TX$ ,  $TY$  contenant les mêmes points, l'un avec les points classés dans l'ordre croissant des abscisses, l'autre par ordre croissant des ordonnées.
- Sorties : Deux points à distance minimum et la distance minimum.

## Fonction PlusProches(TX, TY)

SI  $|TX| < 4$  ALORS :

    | RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

    SI  $d_G < d_D$  ALORS :

        |  $(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

    SINON :

        |  $(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

    SI  $b$  ALORS :

        | RETOURNER  $(P'_0; P'_1, d')$

    SINON :

        | RETOURNER  $(P_0; P_1, d)$

## Exercice

Quelle complexité au final ?

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

```

SI |TX| < 4 ALORS :
    RETOURNER PlusProcheNaif(TX;TY)
SINON :
    (medx; TGX; TDX; TGY; TDY) ← DiviserPoints(TX; TY)
    (PG0; PG1; dG) ← PlusProches(TGX; TGY)
    (PD0; PD1; dD) ← PlusProches(TDX; TDY)
    SI dG < dD ALORS :
        | (P0; P1, d) ← (PG0; PG1; dG)
    SINON :
        | (P0; P1, d) ← (PD0; PD1; dD)
    TY' ← BandeVerticale(TY; d; medx)
    (b; P'0; P'1; d') ← DePartEtDautre(TY'; d)
    SI b ALORS :
        | RETOURNER (P'0; P'1, d')
    SINON :
        | RETOURNER (P0; P1, d)
    
```

$$C(n) =$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX, TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY) **DiviserPoints**

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

$$C(n) =$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX, TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

DiviserPoints  
 PlusProches

$$C(n) = n$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

DiviserPoints

PlusProches

$$C(n) =$$

$n$

$$+ 2 \times C(n/2)$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

DiviserPoints

PlusProches

Si  $d_G < d_D$

$$C(n) =$$

$$n$$

$$+ 2 \times C(n/2)$$



## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX, TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

$$C(n) =$$

$$n$$

$$\text{DiviserPoints} \\ \text{PlusProches} \quad +2 \times C(n/2)$$

$$\text{Si } d_G < d_D \quad +1$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY; d; med\_x)

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY'; d)

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

$$\begin{aligned}
 C(n) &= \\
 &n \\
 &+ 2 \times C(n/2) \\
 &+ 1 \\
 &\text{BandeVerticale}
 \end{aligned}$$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY; d; med\_x)

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY'; d)

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

$C(n) =$

$n$

$+2 \times C(n/2)$

$+1$

$+n$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX, TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX; TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

|  $(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

|  $(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d; med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

| RETOURNER  $(P'_0; P'_1, d')$

SINON :

| RETOURNER  $(P_0; P_1, d)$

	$C(n) =$
DiviserPoints	$n$
PlusProches	$+2 \times C(n/2)$
Si $d_G < d_D$	$+1$
BandeVerticale	$+n$
DePartEtDautre	

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY; d; med\_x)

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY'; d)

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

	$C(n) =$
DiviserPoints	$n$
PlusProches	$+2 \times C(n/2)$
Si $d_G < d_D$	$+1$
BandeVerticale	$+n$
DePartEtDautre	$+8n$

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY; d; med\_x)

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY'; d)

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

	$C(n) =$
DiviserPoints	$n$
PlusProches	$+2 \times C(n/2)$
Si $d_G < d_D$	$+1$
BandeVerticale	$+n$
DePartEtDautre	$+8n$
Si $b$	

## Exercice

### Quelle complexité au final ?

Si on note  $C(n)$  le coût pour trouver la distance minimale d'un nuage de  $n$  points ( $n$  pair) :

Fonction PlusProches(TX,TY)

SI  $|TX| < 4$  ALORS :

RETOURNER PlusProcheNaif(TX;TY)

SINON :

$(med_x; TG_X; TD_X; TG_Y; TD_Y) \leftarrow$  DiviserPoints(TX; TY)

$(P_G0; P_G1; d_G) \leftarrow$  PlusProches(TG\_X; TG\_Y)

$(P_D0; P_D1; d_D) \leftarrow$  PlusProches(TD\_X; TD\_Y)

SI  $d_G < d_D$  ALORS :

$(P_0; P_1, d) \leftarrow (P_G0; P_G1; d_G)$

SINON :

$(P_0; P_1, d) \leftarrow (P_D0; P_D1; d_D)$

$TY' \leftarrow$  BandeVerticale(TY;  $d$ ;  $med_x$ )

$(b; P'_0; P'_1; d') \leftarrow$  DePartEtDautre(TY';  $d$ )

SI  $b$  ALORS :

RETOURNER  $(P'_0; P'_1, d')$

SINON :

RETOURNER  $(P_0; P_1, d)$

	$C(n) =$
DiviserPoints	$n$
PlusProches	$+2 \times C(n/2)$
Si $d_G < d_D$	$+1$
BandeVerticale	$+n$
DePartEtDautre	$+8n$
Si $b$	$+1$

$$C(n) = 2C(n/2) + O(n)$$



$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

Sommons pour  $k$  allant de 1 à  $\ell - 1$  :

$$\sum_{k=1}^{\ell-1} u_{k+1} \leq \sum_{k=1}^{\ell-1} (u_k + \gamma)$$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

Sommons pour  $k$  allant de 1 à  $\ell - 1$  :

$$\sum_{k=1}^{\ell-1} u_{k+1} \leq \sum_{k=1}^{\ell-1} (u_k + \gamma)$$

$$\sum_{k=2}^{\ell} u_k \leq \sum_{k=1}^{\ell-1} u_k + \gamma(\ell - 1)$$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

Sommons pour  $k$  allant de 1 à  $\ell - 1$  :

$$\begin{aligned} \sum_{k=1}^{\ell-1} u_{k+1} &\leq \sum_{k=1}^{\ell-1} (u_k + \gamma) \\ \sum_{k=2}^{\ell} u_k &\leq \sum_{k=1}^{\ell-1} u_k + \gamma(\ell - 1) \\ u_\ell &\leq u_1 + \gamma\ell \end{aligned}$$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

Sommons pour  $k$  allant de 1 à  $\ell - 1$  :

$$\sum_{k=1}^{\ell-1} u_{k+1} \leq \sum_{k=1}^{\ell-1} (u_k + \gamma)$$

$$\sum_{k=2}^{\ell} u_k \leq \sum_{k=1}^{\ell-1} u_k + \gamma(\ell - 1)$$

$$u_\ell \leq u_1 + \gamma\ell$$

$$u_\ell \leq \gamma\ell$$

$$C(n) = 2C(n/2) + O(n)$$

C'est à dire que :  $\exists \gamma > 0$  tel que  $C(n) \leq 2C(n/2) + \gamma n$

Posons pour  $\ell \in \mathbb{N}^*$ ,  $u_\ell = \frac{C(2^\ell)}{2^\ell}$ .

$$\forall \ell \in \mathbb{N}, u_{\ell+1} = \frac{C(2^{\ell+1})}{2^{\ell+1}} \leq \frac{2C(2^\ell) + \gamma 2^{\ell+1}}{2^{\ell+1}} = u_\ell + \gamma$$

Sommons pour  $k$  allant de 1 à  $\ell - 1$  :

$$\sum_{k=1}^{\ell-1} u_{k+1} \leq \sum_{k=1}^{\ell-1} (u_k + \gamma)$$

$$\sum_{k=2}^{\ell} u_k \leq \sum_{k=1}^{\ell-1} u_k + \gamma(\ell - 1)$$

$$u_\ell \leq u_1 + \gamma\ell$$

$$u_\ell \leq \gamma\ell$$



On vient de montrer que  $u_\ell \leq \gamma \ell$ , c'est à dire  $C(2^\ell) \leq \gamma \ell 2^\ell$

On vient de montrer que  $u_\ell \leq \gamma \ell$ , c'est à dire  $C(2^\ell) \leq \gamma \ell 2^\ell$

Ou encore, si  $n$  est une puissance de 2,  $C(n) \leq \gamma n \log_2(n)$

On vient de montrer que  $u_\ell \leq \gamma \ell$ , c'est à dire  $C(2^\ell) \leq \gamma \ell 2^\ell$

Ou encore, si  $n$  est une puissance de 2,  $C(n) \leq \gamma n \log_2(n)$

Ici encore  $C(n)$  est croissante, donc on peut montrer que

$$C(n) = O(n \ln n).$$

On vient de montrer que  $u_\ell \leq \gamma \ell$ , c'est à dire  $C(2^\ell) \leq \gamma 2^\ell$

Ou encore, si  $n$  est une puissance de 2,  $C(n) \leq \gamma n \log_2(n)$

Ici encore  $C(n)$  est croissante, donc on peut montrer que

$$C(n) = O(n \ln n).$$

Il ne faut pas oublier le temps pour trier les tableaux  $TX$  et  $TY$  en début d'algorithme, ce qui peut être fait en  $n \ln n$ .

On vient de montrer que  $u_\ell \leq \gamma^\ell$ , c'est à dire  $C(2^\ell) \leq \gamma 2^{2^\ell}$

Ou encore, si  $n$  est une puissance de 2,  $C(n) \leq \gamma n \log_2(n)$

Ici encore  $C(n)$  est croissante, donc on peut montrer que

$$C(n) = O(n \ln n).$$

Il ne faut pas oublier le temps pour trier les tableaux  $TX$  et  $TY$  en début d'algorithme, ce qui peut être fait en  $n \ln n$ .

Finalement, la complexité est en  $O(n \ln n)$ .

MERCI