Chapitre 5 : Algorithmes « Diviser pour régner »

Rappel : tri fusion

```
let rec fission q=match q with
  | [] | [_] -> q, []
  | x::y::p -> let a,b=fission p in x::a, y::b
;;

let rec fusion p1 p2=match p1, p2 with
  | [],_ -> p2
  | _, [] -> p1
  | x::q1, y::_ when x<=y -> x::(fusion q1 p2)
  | _,x::q2 -> x::(fusion p1 q2)
;;

let rec tri_fusion p=match p with
  | [] | [_] -> p
  | _ -> let a,b=fission p in fusion (tri_fusion a)
            (tri_fusion b)
;;
```

Rappel : tri fusion

Complexité

Pour une liste de taille $n \geq 2$: $C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n)$.

Complexité

Pour une liste de taille $n \geq 2$: $C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n)$.

Résolution

Lorsque n est une puissance de 2 ($n = 2^k$) :

- $C(2^i) = 2C(2^{i-1}) + O(2^i)$ pour tout $i \geq 1$;

Rappel : tri fusion

Complexité

Pour une liste de taille $n \geq 2$: $C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n)$.

Résolution

Lorsque n est une puissance de 2 ($n = 2^k$) :

- $C(2^i) = 2C(2^{i-1}) + O(2^i)$ pour tout $i \geq 1$;
- $\frac{C(2^i)}{2^i} = \frac{C(2^{i-1})}{2^{i-1}} + O(1)$ pour tout $i \geq 1$;

Rappel : tri fusion

Complexité

Pour une liste de taille $n \geq 2$: $C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n)$.

Résolution

Lorsque n est une puissance de 2 ($n = 2^k$) :

- $C(2^i) = 2C(2^{i-1}) + O(2^i)$ pour tout $i \geq 1$;
- $\frac{C(2^i)}{2^i} = \frac{C(2^{i-1})}{2^{i-1}} + O(1)$ pour tout $i \geq 1$;
- Ainsi, $\frac{C(2^k)}{2^k} = \sum_{i=1}^{k} \underbrace{\left[\frac{C(2^i)}{2^i} - \frac{C(2^{i-1})}{2^{i-1}}\right]}_{O(1)} + C(1) = O(k)$;

Rappel : tri fusion

Complexité

Pour une liste de taille $n \geq 2$: $C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n)$.

Résolution

Lorsque n est une puissance de 2 ($n = 2^k$) :

- $C(2^i) = 2C(2^{i-1}) + O(2^i)$ pour tout $i \geq 1$;
- $\frac{C(2^i)}{2^i} = \frac{C(2^{i-1})}{2^{i-1}} + O(1)$ pour tout $i \geq 1$;
- Ainsi, $\frac{C(2^k)}{2^k} = \sum_{i=1}^{k} \underbrace{\left[\frac{C(2^i)}{2^i} - \frac{C(2^{i-1})}{2^{i-1}}\right]}_{O(1)} + C(1) = O(k)$;
- D'où $C(2^k) = O(2^k k) \Rightarrow C(n) = O(n \log n)$.

Algorithmes « diviser pour régner » : principe

Pour résoudre un problème P « gros » :

- découper P en plus petits problèmes indépendants $P_1, \ldots, P_k$;
- résoudre séparément (et récursivement) $P_1, \ldots, P_k$;
- reconstruire une solution à P à partir des solutions aux P_i.

La récursion s'arrête lorsque le problème est « petit ». (exemple tri fusion : liste de taille ≤ 1).

Multiplication rapide de polynômes : problème

Énoncé

- $n > 0$;

Multiplication rapide de polynômes : problème

Énoncé

- $n > 0$;
- $P = \sum_{k=0}^{n-1} p_k X^k$ et $Q = \sum_{k=0}^{n-1} q_k X^k$ deux polynômes ;

Énoncé

- $n > 0$;
- $P = \sum_{k=0}^{n-1} p_k X^k$ et $Q = \sum_{k=0}^{n-1} q_k X^k$ deux polynômes ;
- représentés par des tableaux $[|\, p_0; p_1; \ldots; p_{n-1} \,|]$ et $[|\, q_0; q_1; \ldots; q_{n-1} \,|]$ de taille n ;

Multiplication rapide de polynômes : problème

Énoncé

- $n > 0$;
- $P = \sum_{k=0}^{n-1} p_k X^k$ et $Q = \sum_{k=0}^{n-1} q_k X^k$ deux polynômes ;
- représentés par des tableaux `[|` $p_0; p_1; \ldots; p_{n-1}$ `|]` et `[|` $q_0; q_1; \ldots; q_{n-1}$ `|]` de taille n ;
- PQ représentable par un tableau de taille $2n-1$: comment obtenir les coefficients ?

Multiplication rapide de polynômes : algorithme naïf

Solution naïve

- $PQ = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} p_i q_j X^{i+j}$;

Multiplication rapide de polynômes : algorithme naïf

Solution naïve

- $PQ = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} p_i q_j X^{i+j}$;
- $O(n^2)$.

Multiplication rapide de polynômes : algorithme naïf

Solution naïve

- $PQ = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} p_i q_j X^{i+j}$;
- $O(n^2)$.

Code Caml

```
let prod_naif p q =
  let n = Array.length p in
  let t = Array.make (2*n-1) 0 in
  for i=0 to n-1 do
    for j=0 to n-1 do
      t.(i+j) <- t.(i+j) + p.(i) * q.(j)
    done
  done ;
  t
;;
```

Multiplication rapide de polynômes : algorithme de Karatsuba ($n = 2$)

Coefficients à calculer

$P = p_0 + p_1 X$, $Q = q_0 + q_1 X$, donc
$PQ = p_0 q_0 + (p_1 q_0 + p_0 q_1)X + p_1 q_1 X^2$

Multiplication rapide de polynômes : algorithme de Karatsuba ($n = 2$)

Coefficients à calculer

$P = p_0 + p_1 X$, $Q = q_0 + q_1 X$, donc
$PQ = p_0 q_0 + (p_1 q_0 + p_0 q_1)X + p_1 q_1 X^2$

3 multiplications suffisent !

On calcule :

- $t_0 = p_0 q_0$;
- $t_1 = p_1 q_1$;
- $t_2 = (p_0 + p_1)(q_0 + q_1)$;

Multiplication rapide de polynômes : algorithme de Karatsuba ($n = 2$)

Coefficients à calculer

$P = p_0 + p_1 X$, $Q = q_0 + q_1 X$, donc
$PQ = p_0 q_0 + (p_1 q_0 + p_0 q_1)X + p_1 q_1 X^2$

3 multiplications suffisent !

On calcule :

- $t_0 = p_0 q_0$;
- $t_1 = p_1 q_1$;
- $t_2 = (p_0 + p_1)(q_0 + q_1)$;

Alors $PQ = t_0 + (t_2 - t_1 - t_0)X + t_1 X^2$

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;
- de tailles $m = \lfloor n/2 \rfloor$ et $n - m = \lceil n/2 \rceil$.

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;
- de tailles $m = \lfloor n/2 \rfloor$ et $n - m = \lceil n/2 \rceil$.
- de sorte que $P = P_0 + X^m P_1$

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;
- de tailles $m = \lfloor n/2 \rfloor$ et $n - m = \lceil n/2 \rceil$.
- de sorte que $P = P_0 + X^m P_1$
- de même $Q = Q_0 + X^m Q_1$.

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;
- de tailles $m = \lfloor n/2 \rfloor$ et $n - m = \lceil n/2 \rceil$.
- de sorte que $P = P_0 + X^m P_1$
- de même $Q = Q_0 + X^m Q_1$.

3 multiplications en taille $\simeq n/2$

$$\begin{aligned} PQ &= P_0 Q_0 + X^m(P_0 Q_1 + P_1 Q_0) + X^{2m} P_1 Q_1 = \\ &= T_0 + X^m(T_2 - T_1 - T_0) + X^{2m} T_1 \end{aligned}$$

Multiplication rapide de polynômes : algorithme de Karatsuba (cas général)

Découpage

Pour $n \geq 2$ on pose $m = \lfloor n/2 \rfloor$ et :

- $P_0 = \sum_{k=0}^{m-1} p_k X^k$, $P_1 = \sum_{k=m}^{n-1} p_k X^{k-m}$;
- de tailles $m = \lfloor n/2 \rfloor$ et $n - m = \lceil n/2 \rceil$.
- de sorte que $P = P_0 + X^m P_1$
- de même $Q = Q_0 + X^m Q_1$.

3 multiplications en taille $\simeq n/2$

$$\begin{aligned} PQ &= P_0 Q_0 + X^m (P_0 Q_1 + P_1 Q_0) + X^{2m} P_1 Q_1 = \\ &= T_0 + X^m (T_2 - T_1 - T_0) + X^{2m} T_1 \end{aligned}$$

avec :

- $T_0 = P_0 Q_0$;
- $T_1 = P_1 Q_1$;
- $T_2 = (P_0 + P_1)(Q_0 + Q_1)$;

Multiplication rapide de polynômes : algorithme de Karatsuba

Algorithme 1 : L'algorithme de Karatsuba

Entrée : Deux polynômes P et Q de même taille n, représentés par leurs tableaux de coefficients.

Sortie : Le tableau de taille $2n-1$ associé au produit $P \times Q$.

si $n = 1$ **alors**
  retourner $[\,|\, p_0 \times q_0 \,|\,]$
sinon
  $m = \lfloor n/2 \rfloor$;
  Décomposer $P = P_0 + X^m P_1, Q = Q_0 + X^m Q_1$;
  $T_0 =$Karatsuba(P_0, Q_0);
  $T_1 =$Karatsuba(P_1, Q_1);
  $T_2 =$Karatsuba$(P_0 + P_1, Q_0 + Q_1)$;
  retourner $T_0 + X^m(T_2 - T_1 - T_0) + X^{2m} T_1$

Algorithme de Karatsuba : terminaison / correction

Terminaison

- pour $n = 1$, ok !
- Sinon, 3 appels récursifs :
  - P_0 et Q_0 de tailles $1 \leq \lfloor n/2 \rfloor < n$;
  - P_1, Q_1, $P_0 + P_1$ et $Q_0 + Q_1$ de tailles $1 \leq \lceil n/2 \rceil < n$;

D'où la terminaison !

Algorithme de Karatsuba : terminaison / correction

Terminaison

- pour $n = 1$, ok !
- Sinon, 3 appels récursifs :
  - P_0 et Q_0 de tailles $1 \leq \lfloor n/2 \rfloor < n$;
  - P_1, Q_1, $P_0 + P_1$ et $Q_0 + Q_1$ de tailles $1 \leq \lceil n/2 \rceil < n$;

D'où la terminaison !

Correction

Immédiate par récurrence sur n.

Algorithme de Karatsuba : complexité

Coûts hors appel récursifs

- Calcul de $P_0, P_1, Q_0, Q_1, P_0 + P_1$ et $Q_0 + Q_1$: $O(n)$;

Algorithme de Karatsuba : complexité

Coûts hors appel récursifs

- Calcul de $P_0, P_1, Q_0, Q_1, P_0 + P_1$ et $Q_0 + Q_1$: $O(n)$;
- Une fois T_0, T_1 et T_2 obtenus, calcul de $PQ = T_0 + X^m(T_2 - T_1 - T_0) + X^{2m} T_1$: $O(n)$.

Algorithme de Karatsuba : complexité

Coûts hors appel récursifs

- Calcul de $P_0, P_1, Q_0, Q_1, P_0 + P_1$ et $Q_0 + Q_1$: $O(n)$;
- Une fois T_0, T_1 et T_2 obtenus, calcul de $PQ = T_0 + X^m(T_2 - T_1 - T_0) + X^{2m}T_1$: $O(n)$.

Appels récursifs

- Calcul de $T_0 = \text{Karatsuba}(P_0, Q_0)$: $C(\lfloor n/2 \rfloor)$;

Algorithme de Karatsuba : complexité

Coûts hors appel récursifs

- Calcul de $P_0, P_1, Q_0, Q_1, P_0 + P_1$ et $Q_0 + Q_1$: $O(n)$;
- Une fois T_0, T_1 et T_2 obtenus, calcul de $PQ = T_0 + X^m(T_2 - T_1 - T_0) + X^{2m}T_1$: $O(n)$.

Appels récursifs

- Calcul de $T_0 = \text{Karatsuba}(P_0, Q_0)$: $C(\lfloor n/2 \rfloor)$;
- Calcul de $T_1 = \text{Karatsuba}(P_1, Q_1)$ et $T_2 = \text{Karatsuba}(P_0 + P_1, Q_0 + Q_1)$: $2C(\lceil n/2 \rceil)$

Algorithme de Karatsuba : complexité

Coûts hors appel récursifs

- Calcul de $P_0, P_1, Q_0, Q_1, P_0 + P_1$ et $Q_0 + Q_1$: $O(n)$;
- Une fois T_0, T_1 et T_2 obtenus, calcul de $PQ = T_0 + X^m(T_2 - T_1 - T_0) + X^{2m}T_1$: $O(n)$.

Appels récursifs

- Calcul de $T_0 = \text{Karatsuba}(P_0, Q_0)$: $C(\lfloor n/2 \rfloor)$;
- Calcul de $T_1 = \text{Karatsuba}(P_1, Q_1)$ et $T_2 = \text{Karatsuba}(P_0 + P_1, Q_0 + Q_1)$: $2C(\lceil n/2 \rceil)$

Récurrence

$C(n) = C(\lfloor n/2 \rfloor) + 2C(\lceil n/2 \rceil) + O(n)$ pour $n \geq 2$

Algorithme de Karatsuba : complexité

Application du théorème

$C(n) = aC(\lfloor n/2 \rfloor) + bC(\lceil n/2 \rceil) + O(n^\alpha)$, avec $a = 1, b = 2$, $\alpha = 1$.
$\log_2(a+b) > \alpha$, donc $C(n) = O(n^{\log_2(a+b)}) = O(n^{\log_2(3)})$.
Rem : $\log_2(3) \simeq 1.58$.

Algorithme de Karatsuba : complexité

Application du théorème

$C(n) = aC(\lfloor n/2 \rfloor) + bC(\lceil n/2 \rceil) + O(n^\alpha)$, avec $a = 1, b = 2$, $\alpha = 1$.
$\log_2(a+b) > \alpha$, donc $C(n) = O(n^{\log_2(a+b)}) = O(n^{\log_2(3)})$.
Rem : $\log_2(3) \simeq 1.58$.

Démonstration dans le cas $n = 2^k$

$C(2^i) = 3C(2^{i-1}) + O(2^i)$ pour $i \geq 1$. On divise par 3^i et on télescope :

$$\begin{aligned}
\frac{C(2^k)}{3^k} &= \textstyle\sum_{i=1}^{k} \left(\frac{C(2^i)}{3^i} - \frac{C(2^{i-1})}{3^{i-1}} \right) + C(1) \\
&= \textstyle\sum_{i=1}^{k} O((2/3)^i) + O(1) \\
\frac{C(2^k)}{3^k} &= O(1)
\end{aligned}$$

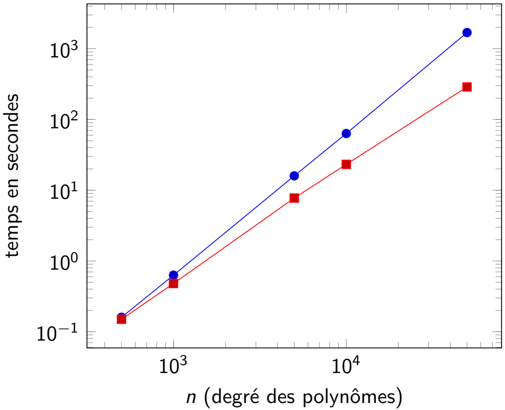
Donc $C(2^k) = O(3^k)$. Or $\log_2(3^k) = k \log_2(3) = \log_2(n) \log_2(3)$. D'où $3^k = 2^{\log_2(n) \log_2(3)} = n^{\log_2(3)}$.

Algorithme de Karatsuba : code Caml

```
let rec karatsuba p q=
  let n=Array.length p in
  if n=1 then [|p.(0)*q.(0)|] else begin
    let k=n/2 in
    let p0, p1=Array.sub p 0 k, Array.sub p k (n-k) and
         q0, q1=Array.sub q 0 k, Array.sub q k (n-k) in
    let t0, t1=karatsuba p0 q0, karatsuba p1 q1 in
    for i=0 to k-1 do
      p1.(i) <- p0.(i) + p1.(i) ;
      q1.(i) <- q0.(i) + q1.(i)
    done ;
    let t2=karatsuba p1 q1 in
    let res=Array.make (2*n-1) 0 in
    for i=0 to Array.length t0 - 1 do
      res.(i) <- res.(i)+t0.(i) ;
      res.(i+k) <- res.(i+k)-t0.(i)
    done ;
    for i=0 to Array.length t1 - 1 do
      res.(i+k) <- res.(i+k)-t1.(i)+t2.(i) ;
      res.(i+2*k) <- res.(i+2*k)+t1.(i)
    done ;
    res
  end
;;
```

Algorithme de Karatsuba : en pratique

$\deg(P) = \deg(Q) = n$	100	500	1000	5000	10000	50000
multiplication naïve	0.006	0.16	0.63	15.96	63.3	1692
Karatsuba	0.07	0.15	0.48	7.75	23.2	288



Multiplication matricielle : algorithme naïf

Problème

$A = (a_{i,j})_{0 \leq i,j < n}$ et $B = (b_{i,j})_{0 \leq i,j < n}$ carrées. $C = AB$ produit matriciel.
Calcul des coefficients de C ?

Multiplication matricielle : algorithme naïf

Problème

$A = (a_{i,j})_{0 \leq i,j < n}$ et $B = (b_{i,j})_{0 \leq i,j < n}$ carrées. $C = AB$ produit matriciel.
Calcul des coefficients de C ?

Algorithme naïf

$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$

Multiplication matricielle : algorithme naïf

Problème

$A = (a_{i,j})_{0 \leq i,j < n}$ et $B = (b_{i,j})_{0 \leq i,j < n}$ carrées. $C = AB$ produit matriciel.
Calcul des coefficients de C ?

Algorithme naïf

$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$
Complexité : $O(n^3)$.

Multiplication matricielle : algorithme de Strassen

Formules de Strassen

Pour n pair, on découpe en 4 chacune des matrices :

$$A = \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right), \quad B = \left(\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}\right) \text{ et } C = \left(\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}\right)$$

On considère alors les 7 produits suivants :

$$\left\{\begin{array}{lcl} P_1 & = & (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\ P_2 & = & (A_{2,1} + A_{2,2})B_{1,1} \\ P_3 & = & A_{1,1}(B_{1,2} - B_{2,2}) \\ P_4 & = & A_{2,2}(B_{2,1} - B_{1,1}) \\ P_5 & = & (A_{1,1} + A_{1,2})B_{2,2} \\ P_6 & = & (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\ P_7 & = & (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}) \end{array}\right.$$

Formules de Strassen

Alors :

$$\begin{cases} C_{1,1} & = & P_1 + P_4 - P_5 + P_7 \\ C_{1,2} & = & P_3 + P_5 \\ C_{2,1} & = & P_2 + P_4 \\ C_{2,2} & = & P_1 - P_2 + P_3 + P_6 \end{cases}$$

Multiplication matricielle : algorithme de Strassen

Formules de Strassen

Alors :

$$\begin{cases} C_{1,1} &= P_1 + P_4 - P_5 + P_7 \\ C_{1,2} &= P_3 + P_5 \\ C_{2,1} &= P_2 + P_4 \\ C_{2,2} &= P_1 - P_2 + P_3 + P_6 \end{cases}$$

Algorithme de Strassen

- Si n petit, effectuer le produit naïf ;
- sinon, utiliser récursivement les formules de Strassen, quite à rajouter une ligne et une colonne de zéros si n impair.

Analyse

$C(n) = 7C(n/2) + O(n^2) \Rightarrow C(n) = O(n^{\log_2(7)})$.
Rem : $\log_2(7) \simeq 2.80$

Points les plus proches d'un nuage

Problème

n points (distincts) du plan. Trouver les deux plus proches.

Points les plus proches d'un nuage

Problème

n points (distincts) du plan. Trouver les deux plus proches.

Solution naïve

Calculer toutes les distances entre deux points distincts : $O(n^2)$.

Code Caml

```
let distance2 p q =
  let a,b=p and c,d=q in (a-c)*(a-c) + (b-d)*(b-d)
;;

let plus_proches_naif a =
  let n=Array.length a in
  let dmin=distance2 a.(0) a.(1) and imin = ref (0,1) in
  for i=0 to n-1 do
    for j=i+1 to n-1 do
      if distance2 a.(i) a.(j) < !dmin then (imin := (i,j) ;
         dmin := distance2 a.(i) a.(j) )
    done
  done ;
  a.(fst !imin), a.(snd !imin)
;;
```

Algorithme « diviser pour régner »

- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)

Algorithme « diviser pour régner »

- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;

Algorithme « diviser pour régner »

- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;
- Sinon, scinder le nuage en deux parties de même taille autour d'une abscisse médiane c $\rightsquigarrow O(n)$

Algorithme « diviser pour régner »

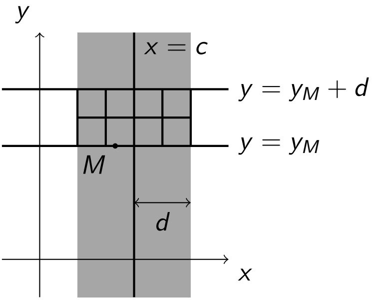
- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;
- Sinon, scinder le nuage en deux parties de même taille autour d'une abscisse médiane $c \rightsquigarrow O(n)$
- S'appeler récursivement sur les deux morceaux : distance minimale trouvée jusque là : d ;

Algorithme « diviser pour régner »

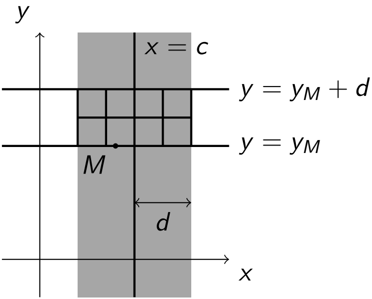
- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;
- Sinon, scinder le nuage en deux parties de même taille autour d'une abscisse médiane $c \rightsquigarrow O(n)$
- S'appeler récursivement sur les deux morceaux : distance minimale trouvée jusque là : d ;
- Calculer la distance minimale entre deux points du nuage situés dans la bande d'abscisse $[c-d, c+d] \rightsquigarrow O(n)!$

Algorithme « diviser pour régner »

- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;
- Sinon, scinder le nuage en deux parties de même taille autour d'une abscisse médiane $c \rightsquigarrow O(n)$
- S'appeler récursivement sur les deux morceaux : distance minimale trouvée jusque là : d ;
- Calculer la distance minimale entre deux points du nuage situés dans la bande d'abscisse $[c-d, c+d] \rightsquigarrow O(n)$!



- Précalcul : trier le nuage par abscisse croissante d'une part, par ordonnée croissante d'autre part ; $\rightsquigarrow O(n \log n)$ (fait une fois !)
- Si le nuage a peu de points, appliquer l'algorithme naïf ;
- Sinon, scinder le nuage en deux parties de même taille autour d'une abscisse médiane $c \rightsquigarrow O(n)$
- S'appeler récursivement sur les deux morceaux : distance minimale trouvée jusque là : d ;
- Calculer la distance minimale entre deux points du nuage situés dans la bande d'abscisse $[c-d, c+d] \rightsquigarrow O(n)$!



Complexité en plus du précalcul :

$$C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + O(n) = O(n \log n)$$

Conclusion

- Méthode efficace pour résoudre un problème ;
- si on peut le découper en petits problèmes indépendants ;
- si le découpage et la reconstruction d'une solution ne coûtent pas cher !