LI325 - Solutionnaire des TD123 - Algorithmes diviser pour régner

Simon Arame

Printemps 2010

1 Notations Asymptotiques

1.1 $f \in O(g)$ si $\exists c, n_0$ tel que $\forall n \geq n_0 \;\; f(n) \leq cg(n)$
$f \in \Omega(g)$ si $\exists c, n_0$ tel que $\forall n \geq n_0 \;\; f(n) \geq cg(n)$
$f \in \Theta(g)$ si $\exists c_1, c_2, n_0$ tel que $\forall n \geq n_0 \;\; c_1 f(n) \leq f(n) \leq c_2 g(n)$

1.2 1)oui 2) oui 3) oui 4) non

1.3 $f_14(n) = \sqrt{log(n)} \in O(f_3(n) = log(n)) \in O(f_11(n) = log(n^2)) \in O(f_6(n) = (log(n))^2) \in O(f_10(n) = \sqrt{n}) \in O(f_13(n) = n) \in O(f_9(n) = n + log(n)) \in O(f_1(n) = 2n) \in O(f_16(n) = nlog(n)) \in O(f_8(n) = n^2) \in O(f_4(n) = n^3) \in O(f_2(n) = 2^n) \in O(f_12(n) = e^n) \in O(f_5(n) = n!) \in O(f_7(n) = n^n)$

2 Fonctions définies par récurrence

2.1 $T(n) = 3(T(\frac{n}{2})) + n^2$
n^2 versus $n^{log_2(3)}$ $\quad //log_2(3) < 2$
(1) => $n^2 \in \Omega n^{log_2(3+\epsilon)}$?
cas 3 du Master theorem ==> $T(n) \in \Theta(n^2)$ si et seulement si
(i) $\exists \epsilon > 0$ vérifiant (1) => oui pour $0 < \epsilon < 2 - log_2(3)$
(ii) $\exists c < 1$ tel que $a(\frac{n}{b})^2 < cn^2$ i.e. $3(\frac{n}{2})^2 < cn^2$ => $3/4n^2 < cn^2$ => oui pour $c = 7/8$

2.2 1a)$T(n) = 2T(\frac{n}{2}) + n^2$
n^2 versus $n^{log_2(2)}$ $\quad //log_2(2) = 1$
(1)$n^2 \in \Omega n^{1+\epsilon}$?
cas 3 du Master theorem $T(n) \in \Theta(n^2)$ si et seulement si
$\exists \epsilon > 0, c < 1$ tel que
(i) (1) est vrai => oui pour $\epsilon = 1/2$ par exemple
(ii) $a(\frac{n}{b})^2 < cn^2$ i.e. $\frac{1}{4}n^2 < cn^2$ => oui pour $c = 3/4$

2.2 1b) $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
n versus $n^{log_2(2)}$ $\quad //log_2(2) = 1$
$n \in \Theta(n^1)$
cas 2 du Master theorem $T(n) \in \Theta(n^1 log(n))$

2.2 1c) $T(n) = 2(T(\frac{n}{2})) + \sqrt{n}$
$\sqrt{n}$ versus $n^{log_2(2)}$ $\quad //log_2(2) = 1$
$\sqrt{n} \in \Omega(n^{1-\epsilon})$ =>vrai pour $\epsilon = 1/4$
cas 1 du Master theorem $T(n) \in \Theta(n)$

2.2 1d) $T(n) = 2T(\frac{n}{2}) + \Theta(1)$
$\Theta(1)$ versus $n^{log_2(1)}$ $//log_2(1) = 0$
$\Theta(1) \in \Theta n^0 = 1$
cas 2 du Master theorem $T(n) \in \Theta n^0 lg(n) = log(n)$

2.1 2 a) ? b) Quicksort c) ? d) recherche binaire

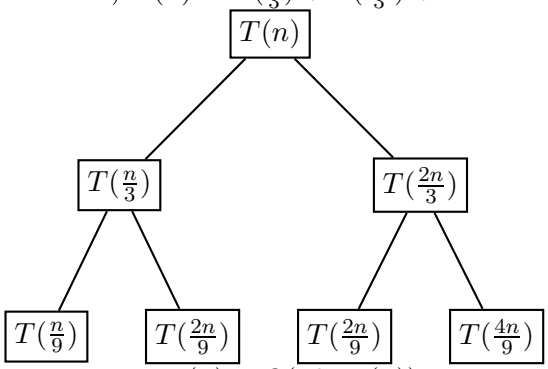
2.3 1a)$T(n) = 2T(\frac{n}{2}) + n^2$
n^2 versus $n^{log_2(2)}$ $//log_2(2) = 1$
$(1) n^2 \in \Omega n^{1+\epsilon}$?
cas 3 du Master theorem $T(n) \in \Theta(n^2)$ si et seulement si
$\exists \epsilon > 0, c < 1$ tel que
(i) (1) est vrai => oui pour $\epsilon = 1/2$ par exemple
(ii) $a(\frac{n}{b})^2 < cn^2$ i.e. $\frac{1}{4}n^2 < cn^2$ => oui pour $c = 3/4$

2.3 1b)$T(n) = 4T(\frac{n}{2}) + \sqrt{n}$
$\sqrt{n}$ versus $n^{log_2(4)}$ $//log_2(4) = 2$
$\sqrt{n} \in \Omega(n^{2-\epsilon})$ pour $\epsilon = 1$ cas 1 du Master theorem $T(n) \in \Theta(n^2)$

2.3 2a) $T(n) = T(\frac{n}{2} + \Omega(\sqrt{n})$
$f(n)$ versus $\Theta(1)$
$T(n) \in \Omega(\sqrt{n})$

2.3 2b) $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
$T(n) = nlog(n)$

2.4 1) $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$



Hypothèse : $T(n) \in O(nlog_{\frac{3}{2}}(n))$
2.4 2) Vérifions notre hypothèse par récurrence avec la méthode de substitution.
Supposons la propriété vrai pour tout $k < n$
$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$
=> $\exists n_0, c$ tel que $T(n) \leq c(\frac{n}{3}log_{\frac{3}{2}}\frac{n}{3}) + c(\frac{2n}{3}log_{\frac{3}{2}}\frac{2n}{3}) + n$
$= c\frac{n}{3}(log_{\frac{3}{2}}(n) - log_{\frac{3}{2}}(3) + 2c\frac{n}{3}(log_{\frac{3}{2}}(2n) - log_{\frac{3}{2}}(3)) + n$
$= c\frac{n}{3}(log_{\frac{3}{2}}(n) - log_{\frac{3}{2}}(3) + 2(log_{\frac{3}{2}}(2n) - log_{\frac{3}{2}}(3))) + n$
$= c\frac{n}{3}(log_{\frac{3}{2}}(n) + log_{\frac{3}{2}}(3) + 2log_{\frac{3}{2}}(n) + 2log_{\frac{3}{2}}(2))) + n$
$= c\frac{n}{3}(3log_{\frac{3}{2}}(n) + log_{\frac{3}{2}}(3) + 2(log_{\frac{3}{2}}(2)) + n$
$T(n) \leq c(nlog_{\frac{3}{2}}(n) + \frac{n}{3}log_{\frac{3}{2}}(3) + 2\frac{2n}{3}log_{\frac{3}{2}}(2) + 1)$
$nlog_{\frac{3}{2}}(n) \leq c(nlog_{\frac{3}{2}}(n) + \frac{n}{3}log_{\frac{3}{2}}(3) + 2\frac{2n}{3}log_{\frac{3}{2}}(2) + 1)$
Vrai pour $c \leq 1$, la propriété est donc vérifié.

3 Recherche d'éléments, de suite d'éléments

3.1 a)

```
RecherSequentiel(T,x)
  POUR i de 0 à Longueur(T)
    SI T[i]==x
      RETOURNER i
    FIN SI
    SI T[i] > x
      RETOURNER FAUX
    FIN SI
  FIN POUR
  RETOURNER FAUX
FIN
```

3.1 b)

```
Rec2(T,x,g,d)
  SI g = d
    SI T[g] = x
      RETOURNER g
    SINON
      RETOURNER FAUX
    FIN SI
  FIN SI
  SI T((g+d/2)) < x
    RETOURNER Rec2(T,x,g,(g+d)/2)
  SINON
    RETOURNER Rec2(T,x,(g+d+1)/2,d)
  FIN SI
FIN
```

3.2 a)

```
RechKieme(T,g,d,k)
  SI g = d
    RETOURNER T[g]
  FIN SI
  p <- T[g]
  j <- g
  POUR i de g+1 à d
    SI T[i] <= p
      j <- j+1
      echanger(j,g)
    FIN SI
    echanger j,g
  FIN POUR
  SI j = k
    RETOURNER T[g]
  FIN SI
  SI j < k
    RETOURNER RechKieme(T,j+1,d,k)
  SINON
    RETOURNER RechKieme(T,g,j-1,k)
```

```
  FIN SI
FIN
```

3.2 b)

2	8	4	1	7	1	3
2	1	4	8	7	10	3
1	2	4	8	7	10	3

3.3 c) $T(n) = T(n-1) + O(n)$

3.2 d) Dans le meilleur des cas, la taille du sous-problème est divisé en 2 à chaque étape, on a alors $T(n) = T(\frac{n}{2}) + O(n) \in \Theta(n)$

3.3 a)

```
AlgoElementaire(T)
  max <- 0
  POUR i de 1 à Longueur(T)
    POUR j de i+1 à Longueur(T)
      s <- 0
        POUR k de i à j
          s<- s+T[k]
          SI s > max
            max <- s
          FIN SI
        FIN POUR
      FIN POUR
    FIN POUR
  FIN POUR
  RETOURNER max
FIN
```

3.3 b)

```
Proc(T)
  RETOURNER REC(T,0,Longueur(T))
FIN
REC(T,a,b)
  SI a==b
    RETOURNER T[a]
  SINON
    m1 <- REC(T,a,(b-a)/2)
    m2 <- REC(T,(b-a+1)/2)
    max <- maximum(m1,m2)
    i <- (a+b)/2
    j <- (a+b+1)/2
    S <- T[j]
    m3 <- T[i]
    TANT QUE i <= a
      S += T[i]
      SI S > max
        max <- S
```

```
        m3 <- S
      FIN SI
      i <- i-1
    FIN TANT QUE
    S <- m3
    POUR k de j à b
      S += T[k]
      SI S > max
        max <- S
      FIN SI
    FIN POUR
    RETOURNER max
  FIN SI
FIN
```

On peut par contre trouver une procedure linéaire encore meilleure

```
PROC1(T)
  max <- 0
  S <- 0
  POUR i de 1 a L
    S <- S + T[i]
    SI S < 0
      S <- 0
    FIN SI
    SI S > max
      max <- S
    FIN SI
  FIN POUR
  RETOURNER max
FIN
```

3.4 1a)

```
Occ(T,i,j,x)
  compteur <- 0
  POUR k de 1 à j
    SI T[k] = x
      compteur += 1
    FIN SI
  FIN POUR
  RETOURNER compteur
FIN
 \par
```

3.4 1b)

```
MAJORITAIRE(T)
        max <- 0
        POUR i de 1 < à Longueur(T)/2
                v <- Occ(T,1,Longueur(T),T[k]
                SI v > Longueur(T)/2
```

```
                        RETOURNER T[k]
                FIN SI
        FIN POUR
        RETOURNER faux
FIN
```

1.3.4 2)

```
Algo(T)
  RETOURNER MAJ(T,0,Longueur(T))
FIN
MAJ(T,a,b)
  SI a = b
    RETOURNER (T[A],vrai)
  SINON
    (rx,x) <- MAJ(T,a,(a+b)/2)
    (ry,y) <- MAJ(T,(a+b+1)/2)
    SI (rx,ry) = (vrai,vrai) ET (x = y)
      RETOURNER (x,vrai)
    SINON
      SI (rx,ry) = (vrai,faux)
        SI Occ(T,(a+b)/2,b,x) + Occ(T,b,(a+b+1)/2,x) > (a+b)/2
          RETOURNER (x,vrai)
        SINON
          RETOURNER (0,faux)
        FIN SI
      SINON
        SI (rx,ry) = (faux,vrai)
          SI Occ(T,(a+b)/2,b,y) + Occ(T,b,(a+b+1)/2,y) > (a+b)/2
            RETOURNER (y,vrai)
          SINON
            RETOURNER (0,faux)
          FIN SI
        SINON //(rx,ry) = (faux,faux)
          RETOURNER (0,faux)
        FIN SI
      FIN SI
    FIN SI
  FIN SI
FIN
```

La complexité est $T(n) = 2T(\frac{n}{2}) + O(n) \in \Theta(nlg(n))$

3.4 3)

```
PSEUDOMAJ(T,g,d)
  SI g = d
    RETOURNER (vrai,x,1)
  FIN SI
  (bx,x,cx) <- PSEUDOMAJ(T,g,(d+g)/2)
  (by,y,cy) <- PSEUDOMAJ(T,(d+g+1)/2,d)
  SI (bx,by) = (faux,faux)
```

```
    RETOURNER (faux,0,0)
  FIN SI
  SI (bx,by) = (vrai,faux)
    RETOURNER (vrai,x,cx+(d-g+1)/4)
  FIN SI
  SI (bx,by) = (faux,vrai)
    RETOURNER (vrai,y,cy+(d-g+1)/4)
  FIN SI
  SI (bx,by) = (vrai,vrai)
    SI x = y
      RETOURNER (vrai,x,cx+cy)
    FIN SI
    SI cx > cy
      RETOURNER (vrai,x,cx+(d-g+1)/2-cy)
    FIN SI
    SI cy > cx
      RETOURNER (vrai,y,cy+(d-g+1)/2-cx)
    FIN SI
    SI cx = cy //x!=y
      retourner (faux,0,0)
    FIN SI
  FIN SI
FIN
```

La complexité de PSEUDOMAJ est $T(n) = 2T(\frac{n}{2}) + \Theta(1) \in \Theta(n)$

4 Tris

4.1 1)

```
FUSION(A,B)
  R <- CreerTablo(Longueur(A)+Longueur(B))
  Ia <- 0
  Ib <- 0
  Ir <- 0
  TANT QUE Ia < Longueur(A) ET Ib < Longueur(B)
    SI A[Ia] <= B[Ib]
      R[Ir] <- A[Ia]
      Ia += 1
      Ir += 1
    SINON
      R[Ir] <- B[Ib]
      Ib += 1
      Ir += 1
    FIN SI
  FIN TANT QUE
  TANT QUE Ia < Longueur(A)
      R[Ir] <- A[Ia]
      Ia += 1
      Ir += 1
  FIN TANT QUE
  TANT QUE Ib < Longueur(B)
      R[Ir] <- B[Ib]
```

```
      Ib += 1
      Ir += 1
  FIN TANT QUE
  RETOURNER R
FIN
```

La complexité de FUSION est pour n = Max(Longueur(A),Longueur(B)), $T(n) = n$

4.1 2)

```
TRIFUSION(T,g,d)
  si g = d
    RETOURNER T
  A <- TRIFUSION(T,g,(g+d)/2)
  B <- TRIFUSION(T,(g+d+1)/2,d)
  RETOURNER FUSION(A,B)
FIN
```

4.1 3)La procédure TRIFUSION est de l'ordre $T(n) = 2T(\frac{n}{2}) + n \in \Theta(nlgn)$

5 Algebre et arithmétique

5.1 1) On fait l'hypothèse que les additions et les multiplications d'entiers se font en un temps unitaire Vérifions que l'addition de deux polynomes de degré $n-1$ se fait en $\Theta(n)$

```
AddPolynome(p,q,n)
  r <- creerPolynome(n)
  POUR i de 1 a n
    n[i] <- p[i] + q[i]
  FIN POUR
  RETOURNER r
FIN
```

Oui, Cet algorithme est bien de l'ordre du degre des opérandes.

5.1 2) La méthode directe va comme suit :

```
MultPolynome(p,q,degMax)
  n <- degMax+1
  C <- creerPolynome(2n-1)
  POUR i de 1 à 2n-1
    C[i] <- 0
  FIN POUR
  POUR i de 0 à n-1
    POUR j de 0 à n-1
      C[i+j] <- C[i+j] + p[i]q[j]
    FIN POUR
  FIN POUR
  RETOURNER C
FIN
```

L'algorithme de la méthode directe est de l'ordre de n^2

5.1 3) Voici l'algorithme récursif pour le produit de 2 polynomes utilisant l'identité suivante :
$pq = p_1q_1 + x^{n/2}(p_1q_2 + p_2q_1) + x^{n/2}p_2q_2$
en prenant pour acquis la décomposition de p et de q suivante tel que dans l'énoncé :
$p(x) = p_1(x) + x^{n/2}p_2(x)$
$q(x) = q_1(x) + x^{n/2}q_2(x)$

```
MultPolyRec(p,q)
  n <- Max(deg(p),deg(q))
  SI n < 3
    RETOURNER MultPolynome(p,q,n)
  FIN SI
  m <- n/2
  //Les quatres prochaines lignes sont à clarifier,
  //Si vous connaissez le fin mot de l'histoire,
  //veuillez m'en informer à simonarame AT xsimo.com
  p1 <- p / x^m
  p2 <- p mod x^m
  q1 <- q / x^m
  q2 <- q mod x^m
  //La suite est sans ambiguité
  M1 <- MultPolyRec(p1,q1)
  M2 <- MultPolyRec(p1,q2)
  M3 <- MultPolyRec(p2,q1)
  M4 <- MultPolyRec(p2,q2)
  RETOURNER M1 + (M2+M3)*x^m + M4*x^m
FIN
```

On trouve l'ordre de cet algorithme récursif en calculant par la méthode du théorème maitre : $T(n) = 4T(\frac{n}{2}) + O(n)$
$T(n) \in \Theta(n^2)$

5.1 4) Toujours en gardant la même décomposition, on utilise maintenant la relation suivante : $p_1q_2 + p_2q_1 = (p_1 + p_2)(q_1 + q_2) - p_1q_1 - p_2q_2$ Alors l'algorithme récursif devient :

```
MultPolyRec2(p,q)
  n <- Max(deg(p),deg(q))
  SI n < 3
    RETOURNER MultPolynome(p,q,n)
  FIN SI
  m <- n/2
  //Les quatres prochaines lignes sont à clarifier,
  //Si vous connaissez le fin mot de l'histoire,
  //veuillez m'en informer à simonarame AT xsimo.com
  p1 <- p / x^m
  p2 <- p mod x^m
  q1 <- q / x^m
  q2 <- q mod x^m
  //La suite est sans ambiguité
  M1 <- MultPolyRec2(p1,q1)
  M4 <- MultPolyRec2(p2,q2)
```

```
  M5 <- MultPolyRec2(p1+p2,q1+q2)
  RETOURNER M1 + (M5-M1-M4)*x^m + M4*x^m
FIN
```

La complexité est alors d'ordre inférieur à ce que l'on avait auparavant. On sépare le problème en trois sous-problèmes de taille $n/2$ au lieu de quatre. On a donc par le théorème maitre,
$T(n) \in \Theta(n^{log_2(3)})$
et on a bien $n^2 > n^{log_2(3)} \forall n > 1$

5.1 5) À mon avis il s'agit d'une optimalité relative. Certains chercheurs ont du améliorer encore plus le modèle.

5.2 (Algorithme de strassen pour la multiplication de 2 matrices) Cet exercice ne fut pas traité par notre groupe par manque de temps. Il me ferait plaisir de recevoir une solution à l'adresse simonarame AT xsimo.com.

5.3 (Matrice de Toeplitz) IDEM

6 Géométrie algorithmique

6.1 1) Voici l'algorithme naïf qui calcule directement les deux points les plus rapprochés dans un nuage de points

```
AlgoNaif(P,n)
  dmin <- +infinity
  POUR i de 1 à n
    POUR j de i+1 à n
      SI distance(P[i],P[j])<dmin
        dmin <- distance(P[i],P[j])
      FIN SI
    FIN POUR
  FIN POUR
  RETOURNER dmin
FIN
```

Cet algorithme est de l'ordre de n^2.

6.1 2)Voici l'algorithme DiviserPoints qui calcule à partir de P les tableaux PgaucheX (resp. PgaucheY) correspondant aux points de l'ensemble PGauche triés selon les abcisses (resp. les ordonnées), de même pour les tableaux PdroitX et PdroitY.

```
DiviserPoints(PX,PY)
  PGX <- PX[1,n/2]
  PDX <- PX[n/2,n]
  PGY <- CreerTableau(n/2)
  PDY <- CreerTableau(n/2)
  l <- PX[n/2]
  i <- 1
  j <- 1
  POUR k de 1 à n
    SI PY[k].x < l.x OU ( PY[k].x = l.x ET PY[k].y < l.y )
      PGY[i] <- PY[k]
```

```
      i++
    SINON
      PDY[j] <- PY[k]
      j++
    FIN SI
  FIN POUR
  RETOURNER (l,PGX,PDX,PGY,PDY)
FIN
```

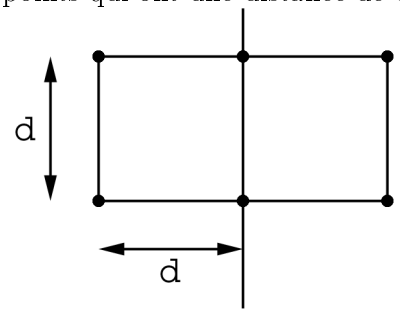
Cet algorithme est clairement de l'ordre de n

6.1 3a) L'algorithme BandeVerticale retourne le PY' cherché.

```
BandeVerticale(PY,l,d)
  i <- 1
  PY' <- creerTableauPoints()
  POUR j de 1 à n
    SI ( PY[j].x >= l.x-d ET PY[j].x <= l.x+d )
      PY'[i] <- PY[j]
      i++
    FIN SI
  FIN POUR
  RETOURNER PY'
FIN
```

6.1 3b) sur et dans un carré de côté a, si on place 4 points aux 4 coins du carré, la distance minimale entre 2 de ces points est a. Lorsque l'on rajoute un cinquième point, il ne peut faire que diminuer la distance minimale entre 2 points quelquonques de ces 5 points. Pour s'en convaincre, dessiner un carré et essayer de placer 5 points de telle sorte que la distance minimale soit a, c'est impossible.

6.1 3c) On démontre ici le fondement de l'efficacité de cet algorithme. Les points de PY' sont triés en ordre d'abcisse et ces derniers sont situés dans une bande verticale de "division" du nuage de points qui a une largeur 2d. Soit une largeur de d de part et d'autres de la droite verticale de division. En disposant 6 points qui ont une distance de d, le septième point est forcément à une distance inférieure à d.



6.1 3d) Voici l'algorithme en $O(n)$ qui vérifie s'il existe une paire de points dans PGauche et PDroit dont la distance est strictement inférieure à d et si oui la retourne ainsi que les points concernés.

```
RechVerticale(PY',d)
  dmin <- TRIPLET(d,null,null)
  POUR i de 1 a n
    POUR j de i+1 à min(i+8,n)
      SI distance(PY'[i],PY'[j])<dmin
```

```
        dmin <- TRIPLET(distance(PY'[i],PY'[j]), PY'[i],PY'[j])
      FIN SI
    FIN POUR
  FIN POUR
  RETOURNER dmin
FIN
```

6.1 4) L'algorithme prend entrée les tableaux triées PX et PY qui représente 2 fois le nuage de points trié respectivement en ordonnées et en abcisse. Ce prétraitement peut se faire en $O(nlog(n))$ par un tri-fusion par exemple.

```
PlusProches(PX,PY)
  SI (longueur(PX) < 4 )
    RETOURNER AlgoNaif(PX)
  SINON
    (l,PGX,PDX,PGY,PDY) <- DiviserPoints(PX,PY)
    TRIPLET(dg,PGX,PGX) <- PlusProches(PGX,PGY)
    TRIPLET(dd,PDX,PDX) <- PlusProches(PDX,PDY)
    dr <- min(dg,dd)
    PY' <- BandeVerticale(PY,l,dr)
    TIPLET(dm,P1,P2) <- RechVerticale(PY',dr)
    SI dr<dm
      RETOURNER TRIPLET(dr,P1,P2)
    SINON
      RETOURNER TRIPLET(dm,P1,P2)
    FIN SI
  FIN SI
FIN
```

6.1 5) La relation de récurrence va comme suit : $T(n) = 2T(n/2)+O(n)$. On obtient un temps d'exécution de l'ordre de $nlog(n)$