Correction
Polytechnique 2004 (Filière MP – Option Informatique)
Composition d'Informatique

R. LOUBOUTIN
Lycée Chateaubriand
Trabujo.Louboutin@wanadoo.fr

31 janvier 2006

Premier problème : Sélection

1 : Il s'agit de calculer la taille d'une liste. C'est classique.

Prog. 1 Question 1 : card

```
let rec card X = match X with
  | [] -> 0
  | t::q -> 1 + card q
;;
```

La complexité temporelle de cette fonction est essentiellement proportionnelle au nombre d'appels de fonctions (le premier appel inclus) lorsque `card` est appelée pour une liste de longueur n.
On montre par récurrence que ce nombre d'appels est $n+1$.
La complexité en temps de cette fonction est donc linéaire.

2 : Même principe que dans la fonction `card`, ici on ne compte que les éléments strictement inférieurs à p.

Prog. 2 Question 2 : nPetits

```
let rec nPetits p X = match X with
  | [] -> 0
  | t::q ->
    if t<p then
      1 + nPetits p q
    else
      nPetits p q
;;
```

La complexité (en termes d'appels) est toujours $n+1$.

3 : Pratiquement la même question que la précédente. Ici au lieu de compter le nombre d'éléments inférieurs strictement à p, on construit l'ensemble de ces éléments en les ajoutant successivement à l'ensemble vide.
Pour la fonction suivante il suffit d'inverser le test.
Ces deux fonctions ont bien évidemment ausi une complexité en $n+1$

4 : En prenant pour pivot t le premier élement de la liste on scinde l'ensemble en trois : l'ensemble des éléments strictement inférieurs à t, l'ensemble dont t est l'unique élément, et l'ensemble des élements strictement supérieurs à t. En tenant compte du nombre d'éléments strictement inférieurs à t on va chercher l'élément de rang k dans le bon ensemble. Ce programme termine car soit il n'y a pas d'appel récursif, soit cet appel à lieu sur un ensemble de taille strictement inférieure.

Prog. 3 Question 3 : partitionP

```
let rec partitionP p X = match X with
  | [] -> []
  | t::q ->
    if t<p then
      t::(partitionP p q)
    else
      partitionP p q
;;
```

Prog. 4 Question 3 : partitionG

```
let rec partitionG p X = match X with
  | [] -> []
  | t::q ->
    if t>p then
      t::(partitionG p q)
    else
      partitionG p q
;;
```

Prog. 5 Question 4 : elementDeRang

```
let rec elementDeRang k X = match X with
  | [] -> failwith "erreur"
  | t::q ->
    let (G,D) = (partitionP t q, partitionG t q)
    in
      let m = card G
      in
        if k <= m then
          elementDeRang k G
        else if k = m+1 then
          t
        else
          elementDeRang (k-m-1) D
;;
```

5 : La remarque faite dans la question précédente montre que le nombre d'appels (l'appel initial, puis les appels récursifs) à `elementDeRang` est au plus $n+1$. Lors de chacun de ces appels le calcul de `card` a une complexité $m+1$. Le cas le plus défavorable sera celui où on a n appels récursifs, m décroissant de $n-1$ à 0. La complexité sera alors $\frac{n(n+1)}{2}$ appels à `card` et $n+1$ appels à `elementDeRang`. Cette situation se produit par exemple si on cherche l'élément de rang 1 dans un ensemble de cardinal n représenté par une liste strictement décroissante.
La complexité dans le pire des cas peut être quadratique.

6 : On montre par récurrence sur n que pour tout n $T(n) \leq 4ln$. Ce résultat est vrai à l'ordre 0. On se donne $n \geq 1$ et on suppose $T(i) \leq 4li$ pour $0 \leq i \leq n-1$. Deux cas sont à distinguer $n = 2p$ et $n = 2p+1$. Rédigeons la démonstration dans le premier cas, dans le deuxième cas elle est similaire.

$$\begin{aligned} T(n) &\leq ln + \frac{1}{n}\sum_{i=1}^{n} \max(T(i-1), T(n-i)) \\ &\leq ln + \frac{2}{n}\sum_{i=p+1}^{2p} T(i-1) \\ &\leq ln + \frac{2}{n} 4l \sum_{i=p}^{2p-1} i \end{aligned}$$

$$
\begin{aligned}
&\leq ln + \frac{2}{2p} 4l \left(\frac{2p(2p-1)}{2} - \frac{p(p+1)}{2} \right) \\
&\leq ln + 2l(2(2p-1) - (p+1)) \\
&\leq ln + 6lp \\
&\leq 4ln
\end{aligned}
$$

7 : L'énoncé n'est pas tout-à-fait clair ici. La condition $5i < n$ laisse penser qu'on peut considérer des paquets terminaux ayant moins de 5 éléments. Bien que cela ne nuise pas aux performances de l'algorithme, j'ai retenu la condition $5i + 5 \leq n$ pour deux raisons :

1. Comme le dit l'énoncé, si le nombre d'élément d'un paquet est pair, le médian n'est pas clairement défini.
2. Ce choix conduit aux majorations demandées à la question 9, l'autre choix conduisant á un +6 au lieu du +4.

Prog. 6 Question 7 : medians

```
let rec medians = function
  | x1::x2::x3::x4::x5::queue ->
    (elementDeRang 3 [x1;x2;x3;x4;x5])::(medians queue)
  | _ -> []
;;
```

8 :

Prog. 7 Question 8 : elementDeRangBis

```
let rec elementDeRangBis  k x = match x with
  | [] -> failwith "Il n'y a pas d'\'el\'ement dans la liste vide"
  | _ ->
    let Y = medians x
    in
      let m = card Y
      in
        if m>=1 then
          let p = elementDeRangBis ((m+1)/2) Y
          in
            let (G,D) = (partitionP p x, partitionG p x)
            in
              let m = card G
              in
                if k<=m then
                  elementDeRangBis k G
                else if k=m+1 then
                  p
                else
                  elementDeRangBis (k-m-1) D
        else
          elementDeRang k x
```

9 : Tout le travail de maintenance : calcul de Y, calcul du cardinal de G , partition de X et calcul direct quand il n'y a pas d'appel récursif se fait en un temps au maximum proportionnel à n. C'est la partie $l'n$ de la majoration. Il reste la détermination de p qui se fait en $M'(\lfloor \frac{n}{5} \rfloor)$, puisque Y est de cardinal $\lfloor \frac{n}{5} \rfloor$, ainsi que l'appel récursif sur G ou D. Quitte à remplacer $M'(n)$ par $\max_{i \leq n} M'(i)$ on peut supposer que M' est croissante.
Il nous suffit donc de majorer la taille de G et celle de D. La situation étant symétrique majorons la taille de G. Soit m le cardinal de Y. $\lceil \frac{m-1}{2} \rceil$ éléments de Y sont strictement inférieurs à p et $\lceil \frac{m}{2} \rceil$ élements sont supérieurs ou égaux à p (Dans le cas oú m est pair, on choisit le plus grand médian possible). Or un élément de Y est le médian de 5 éléments de X. Si ce médian est strictement inférieur à p, au maximum 5 de ces éléments sont strictement inférieurs à p, sinon au maximum 2 le

sont. Il y a au maximum 4 éléments de X qui n'entrent pas dans la détermination de Y et dont la valeur par rapport à p est indéterminée. Finalement le cardinal de G est au plus

$$5\left\lceil\frac{\lfloor\frac{n}{5}\rfloor-1}{2}\right\rceil+2\left\lfloor\frac{\lfloor\frac{n}{5}\rfloor+1}{2}\right\rfloor+4\leq 7\lfloor\frac{n}{10}\rfloor+4.$$

On obtient ainsi le dernier terme de la majoration.
En remarquant $\lfloor\frac{n}{5}\rfloor+7\lfloor\frac{n}{10}\rfloor+4\sim\frac{9n}{10}$, il existe n_0 tel que pour $n\geq n_0$: $\lfloor\frac{n}{5}\rfloor+7\lfloor\frac{n}{10}\rfloor+4\leq\frac{19n}{20}$. Si on choisit une constante c' telle que $M'(n)\leq c'n$ pour $n\leq n_0$ et $c'\geq 20l'$, on démontre par récurrence que $M'(n)\leq c'n$ pour tout n.

10 : On aurait obtenu une inégalité

$$M'(n)\leq l'n+M'(\lfloor\frac{n}{3}\rfloor)+M'(4\lfloor\frac{n}{6}\rfloor+2)$$

qui ne permet pas de conclure comme ci-dessus car $\frac{1}{3}+\frac{4}{6}=1$.
Mais cela ne constitue pas une preuve !
Pour cela il faut admettre qu'il est possible d'ordonner les élements de X dans la liste de telle sorte qu'on ait à chaque étape une égalité dans l'inégalité précédente (on s'en convainc facilement). On prouve alors par récurrence que si $n=p6^k$ alors $M'(n)\geq l'(k+1)n$. $M'(n)$ n'admet pas de majoration globale de la forme $c'n$.

Second problème : Localisation dans un polygone convexe

11 : L'énoncé est peu précis sur les conditions imposées aux points. Pour que α soit toujours défini il faudrait que Q et R soient toujours distincts de P. On choisit de renvoyer 0 lorsque α n'est pas défini.

Prog. 8 Question 11 : orientation

```
let orientation P Q R =
    let temp = (Q.x - P.x)*(R.y - P.y)- (Q.y - P.y)*(R.x -P.x)
    in
        if temp > 0 then
            1
        else if temp = 0 then
            0
        else
            -1
;;
```

12 : Comme dans la question précédente l'énoncé est peu précis sur les conditions imposées (égalité des points, alignement des points). Il ne précise pas non plus si « à droite » doit être pris au sens strict ou large. Nous retiendrons le sens large de « à droite ».
Cette fonction renvoie toujours un résultat. On peut ne pas être d'accord sur le résultat renvoyé dans certains cas de figure (par exemple si $Q_2=R_2$), mais elle renvoie le résultat naturel ou souhaité (lorsqu'il est conventionnel) dans les cas qui nous intéresseront aux questions 13, 14 et 15, c'est à dire :
– $Q_1\neq Q_2$ et $R_1\neq R_2$ et $R_2\neq Q_2$ (mais peut-être $R_2=Q_1$ ou $R_1=Q_2$),
– $Q_1\neq Q_2$ et $R_1=Q_1$ et $R_2=Q_2$.

13 : Le choix que nous avons fait dans la programmation de `orientation` nous permet de définir le plan tout entier comme $\rho(P_0P_1P_0P_1)$, ce qui nous permet de définir une structure de données plus récursive que celle de l'énoncé (celle-ci étant d'ailleurs peu adaptée au cas $n=3$ admis par l'énoncé).
L'arbre va être construit récursivement à l'aide d'une fonction récursive interne à `construire`. La structure de l'arbre ne dépendant que de n il serait aussi plus intéressant de travailler sur un tableau de points c'est à dire de convertir la liste en un tableau. On s'éloignerait probablement de l'idée de l'auteur. On conserve donc les listes.
Le procédé de construction est récursif et il nécessite de couper une liste en deux morceaux dont les longueurs diffèrent au plus d'une unité. La complexité d'une telle opération est proportionnelle à la taille de la liste. Par le biais de la récursivité on introduit une complexité en $\Theta(n\ln n)$ qui peut être ramenée à une complexité en $\Theta(n)$ par l'artifice qui va suivre. Cette réduction est nécessaire puis que la transformation de la liste en tableau est aussi de complexité en $\Theta(n)$.
La fonction `card` permettant de mesurer la longueur d'une liste ayant été programmée dans le problème précédent, on utilisera sans scrupule la fonction équivalente du langage `Caml`.

Prog. 9 Question 12 : aDroiteDe

```
let ad P Q R = (orientation P Q R) >= 0 ;;

let aDroiteDe Q1 Q2 R1 R2 T =
    if (Q1=R1) && (Q2=R2) then
        true
    else
        if (ad Q1 Q2 R2) then
            if  (ad Q1 Q2 R2) then
                ((ad Q1 Q2 T) && (ad R2 Q2 T)) || ((ad R2 T Q2) && (ad R2 R1 T))
            else
                (ad Q1 Q2 T) && (ad R2 Q2 T) && (ad R2 R1 T)
        else
            if  (ad R2 R1 Q2) then
                ((ad  R2 R1 T) && (ad R2 Q2 T)) || ((ad Q2 R2 T) && (ad Q1 Q2 T))
            else
                (ad Q1 Q2 T) || (ad Q2 R2 T) || (ad R2 R1 T)
;;
```

Prog. 10 Question 13 : construire

```
let construire p =
    let rec aux i j L =
        if j=i+1 then
            match L with (p0::p1::p2::q) ->
                (Feuille(p0,p1,p2),p1::p2::q)
        else
            let (res1,L1) = aux i ((i+j)/2) L
            in
                let (res2,L2) = aux ((i+j)/2) j L1
                in
                    match (L,L2) with (p0::p1::_,p2::p3::_) ->
                        (Noeud(p0,p1,p2,p3,res1,res2),L2)
    in
        fst (aux 0 (list_length p) (p@[hd(p);hd(tl(p))]) )
;;
```

La complexité de cette fonction en temps est pratiquement proportionnelle au nombre $c(n)$ d'appels récursifs à `aux`. Or la fonction c vérifie $c(1) = 1$ et $c(n) = 1 + c(\lfloor \frac{n}{2} \rfloor) + c(\lceil \frac{n}{2} \rceil)$. De cette relation on déduit $n \leq c(n) \leq 2n - 1$.

14 : Il suffit de l'appeler avec P_{i-1} P_i P_i P_{i+1} et T.

15 : Chaque noeud interne représente l'ensemble E des points du plans à droite du zig-zag défini par ce noeud, répartis sur trois zones. La zone Eg définie par le fils gauche, la zone Ed définie par le fils droit et le complémentaire dans E de la réunion de ces deux zones. Pour qu'un point soit à l'intérieur du polygone sachant qu'il est dans la zone E, il faut et il suffit qu'il ne soit ni dans Eg ni dans Ed, ou qu'il soit un point intérieur au polygone dans une de ces zones. Un point dans une zone définie par une feuille ne peut être intérieur au polygone.
Si les feuilles avaient été définies comme des noeuds internes dans lesquels les deux points médians sont égaux, on aurait largement allégé le programme suivant.
La complexité de cette fonction est au plus proportionnelle à la hauteur de l'arbre. Comme l'arbre est équilibré cette complexité est en $\mathcal{O}(\ln n)$.

16 : C'est le même principe que pour `aLInterieurDe`. Cette fois-ci il faut que que le point T soit dans le secteur associé à la feuille à laquelle on aboutit. Si la feuille est étiquetée par (P, Q, R) Q est le point de contact recherché.
On ne fait aucune vérification, on suppose que le point T est à l'extérieur du polygone.

17 : On peut par exemple remplacer `orientation` par son opposée et inverser l'ordre des points dans le polygone.

Prog. 11 Question 15 : aLInterieurDe

```
let rec aLInterieurDe a T = match a with
  | Noeud(P0,P1,P2,P3,aGauche,aDroite) ->
    if aDroiteDe P0 P1 P2 P3 T then
      match (aGauche,aDroite) with
        | ( Noeud (p0,p1,p2,p3,_,_) , Noeud (q0,q1,q2,q3,_,_) ) ->
          (aLInterieurDe aGauche T) || (aLInterieurDe aDroite T) ||
                    not((aDroiteDe p0 p1 p2 p3 T) || (aDroiteDe q0 q1 q2 q3 T))
        | ( Feuille (p0,p1,p2) , Noeud (q0,q1,q2,q3,_,_) ) ->
          (aLInterieurDe aGauche T) || (aLInterieurDe aDroite T) ||
                    not ((aDroiteDe p0 p1 p1 p2 T) || (aDroiteDe q0 q1 q2 q3 T))
        | ( Noeud (p0,p1,p2,p3,_,_) , Feuille (q0,q1,q2) ) ->
          (aLInterieurDe aGauche T) ||
                    not ( (aDroiteDe p0 p1 p2 p3 T) || (aDroiteDe q0 q1 q1 q2 T))
        | ( Feuille (p0,p1,p2) , Feuille (q0,q1,q2) ) ->
          not ((aDroiteDe p0 p1 p1 p2 T) || (aDroiteDe q0 q1 q1 q2 T))
    else
      false
  | Feuille(P0,P1,P2) ->
    not (aDroiteDe P0 P1 P1 P2 T)
;;
```

Prog. 12 Question 16 : tangenteG

```
let rec tangenteG a T = match a with
  | Noeud(P0,P1,P2,P3,aGauche,aDroite) ->
    ( match (aGauche,aDroite) with
      | ( Feuille (p0,p1,p2) , _ ) ->
          if aDroiteDe p0 p1 p1 p2 T then
            p1
          else
            tangenteG aDroite T
      | ( Noeud (p0,p1,p2,p3,_,_) , _ ) ->
          if aDroiteDe p0 p1 p2 p3 T then
            tangenteG aGauche T
          else
            tangenteG aDroite T )
  | Feuille(P0,P1,P2) ->
      P1
;;
```