Épreuve d'informatique de l'X - 2012 - MP/PC

I. Grand, petit et médian

Question 1. On part parcourt le tableau de la case d'indice a jusqu'à la case d'indice b. Si le k-ème élément est plus grand le maximum de $tab[a..k-1]$ on mémorise k comme étant l'indice du plus élément du sous-tableau $tab[a..k]$ et on modifie la valeur de `PlusGrand`. À chaque fin de boucle, `PlusGrand` représente le plus grand élément du sous-tableau $tab[a..k]$ et `indiceMax` son indice.

```
1  def calculeIndiceMaximum(tab,a,b):
2      indiceMax = a
3      PlusGrand = tab[a]
4      for k in xrange(a+1,b):
5          if tab[k] > PlusGrand:
6              indiceMax = k
7              PlusGrand = tab[k]
8      return indiceMax
```

Question 2. On parcourt le sous-tableau $tab[a..b]$ à l'aide de la boucle des lignes 3 à 5 et on comptabilise le nombre d'éléments plus petits que `val` à l'aide du compteur `compt`.

```
1  def nombrePlusPetit(tab,a,b,val):
2      compt = 0
3      for k in xrange(a,b):
4          if tab[k] <= val:
5              compt += 1
6      return compt
```


II. Un tri pour accélérer

Question 3 : On commence par écrire une fonction `permute` qui, étant donné un tableau `tab` et deux indices `a` et `b`, permute les éléments en position `a` et `b` dans le tableau `tab`. On parcourt le tableau de la case d'indice `a` à la case d'indice `b`. On place les éléments plus petits que le pivot dans les premières cases du sous-tableau $tab[a..b]$. La variable `pointeur` donne l'indice du premier élément plus grand que le pivot. Dans la boucle des lignes 9 à 12, si en position i on trouve un élément plus petit que le pivot, on le permute avec celui qui est en position «`pointeur`» et on incrémente la variable `pointeur`. Dans la boucle des lignes 14 à 17, on fait redescendre les éléments égaux à `pivot` pour qu'ils succèdent aux éléments strictement plus petits que le pivot. On retourne comme valeur le plus indice d'un élément égal au pivot.

```
1  def permute(t,a,b):
2      tmp = t[a]
3      t[a] = t[b]
4      t[b] = tmp
5
6  def partition(tab,a,b,indicePivot) :
7      Pivot = tab[indicePivot]
8      pointeur = a
9      for i in xrange(a,b):
10         if tab[i] < Pivot:
11             permute(tab,i,pointeur)
12             pointeur += 1
13     positionPivot = pointeur
14     for i in xrange(pointeur, b):
15         if tab[i] == Pivot:
16             permute(tab,i,pointeur)
17             pointeur += 1
18     return positionPivot
```

Question 4 : On applique l'algorithme décrit par l'énoncé en utilisant la récursivité.

```
def elementK(tab,a,b,k) :
    if k == 1 and a == b :
        return tab[a]
    else :
        i = partition(tab,a,b,a)
        if i-a+1 > k:
            return  elementK(tab,a,i-1,k)
        elif i-a+1 == k:
            return tab[a+k-1]
        elif k == 1:
            return tab[a]
        else :
            return elementK(tab,i+1,b,k-i+a-1)
```

Question 5 : si à chaque étape, on choisit comme pivot le plus grand élément et que l'on recherche le plus élément, alors à chaque étape on élimine qu'un seul élément dans notre recherche. La complexité est alors en $\mathcal{O}(n+(n-1)+...+1) = \mathcal{O}(n^2)$. La complexité est donc quadratique.

Question 6 : On reprend la fonction `elementK` pour écrire une fonction qui retourne l'indice du k-ème élément et non sa valeur.

```
def IndiceElementK(tab,a,b,k) :
    K = k
    debut = a
    fin = b
    while K !=1 or debut != fin :
        i = partition(tab,debut,fin,debut)
        if i - debut + 1 > K :
            fin = i-1
        elif i - debut + 1 == K :
            debut = i
            fin = i
            K = 1
        else :
            K = K-i + debut-1
            debut = i+1
    return debut

def choixPivot(tab,a,b) :
    n = int((b-a+1) / 5.)
    if n == 0 :
        return elementK(tab,a,b,int((b-a+1)/ 2.))
    else :
        for i in range(n+1) :
            permute(tab, a+i,IndiceElementK(tab,a+5*i,a+5*(i+1)-1,3)
        permute(tab, a+n, IndiceElementK(tab,a+5*n,b,int((b-5*n+a+1
        return choixPivot(tab,a,a+n)
```

III. De la 1D vers la 2D, des nombres aux points

Question 7 :

```
1  def coupeY(tabX,tabY) :
2      n = taille(tabX)
3      tabY[indiceMedian(tabY,1,n)]
```

Question 8 : On crée un tableau pour stocker les valeurs des angles (ligne 5). On parcourt tous les points. Si le i-ème point est strictement au dessus du point (x, y), on calcule l'angle entre la demi-droite partant de (x, y) et allant vers la droite et le segment $(x, y) - (x2, y2)$. On détermine et on renvoie l'angle médian.

```
1  def demiDroiteMedianeSup(tabX,tabY,x,y) :
2      n = len(tabX)
3      tabAngle = []
4      compt = 0
5      for i in xrange(1,n-1) :
6          if tabY[i] > y :
7              compt += 1
8              tabAngle.append(angle(x,y,tabX[i],tabY[i]))
9      return choixPivot(tabAngle,1,compt)
```

Question 9 : Dans la boucle des lignes 9 à 16, on détermine, parmi les points strictement en dessous du point (x, y), ceux qui sont à gauche de la droite d'angle polaire θ et ceux qui sont à droite et on les comptabilise. On compare les nombres trouvés à $\lceil \frac{n}{2} \rceil$ pour retourner la bonne réponse (lignes 14 à 19).

```
1  def verifieAngleSecondeDroite(tabX,tabY,x,y,theta) :
2      pi = math.pi
3      n = taille(tabX)
4      #      tabAngle = make_vect n 0. in
5      compt1 = 0
6      compt2 = 0
7      for i in xrange(1, n) :
8          if tabY[i] < y :
9              if angle(x,y,tabX[i],tabY[i]) < theta + pi :
10                 compt1 += 1
11             else :
12                 compt2 += 1
13     l2 = math.ceil( (compt1 + compt2)/ 2.)
14     if compt1 > l2 :
15        return 1
16     elif compt2 > l2 :
17         return -1
18     else :
19         return 0
```

Question 10 : On calcule $\alpha = \min_{1 \leq i \leq n} x_i$ et $\beta = \max_{1 \leq i \leq n} x_i$ dans la boucle des lignes 9 à 13. Dans la boucle conditionnelle des lignes 15 à 28, on cherche par dichotomie un x qui convient. On quitte la boucle dès que l'on a trouvé un tel x et le programme par l'instruction « `return` »de la ligne 29.

```
1  def secondeMediane(tabX,tabY,y) :
2      tabXc = copy.copy(tabX)
3      tabYc = copy.copy(tabY)
4      x = 0.
5      alpha = tabX[0]
6      beta = tabX[0]
7      angle = 0.
8      tabRes = allouer(2,0.)
9      for i in xrange(1,taille(tabX)) :
10         if tabX[i] < alpha :
11             alpha = tabX[i]
12         if tabX[i] > beta :
13             beta = tabX[i]
14     trouve = false
15     while not trouve and (beta - alpha) > 0.001 :
16         x = (alpha + beta) / 2.
17         angle = demiDroiteMedianeSup(tabX,tabY,x,y)
18         tabRes[1] = x
19         tabRes[2] = angle
20         d = verifieAngleSecondeDroite(tabX,tabY,x,y,angle)
21         if d == 0 :
22            tabRes[1] = x
23            tabRes[2] = angle
24            trouve = True
25         elif d == -1 :
26            beta = x
27         else :
28            alpha = x
29     return tabRes
```