II. Un tri pour accélérer

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

I. Grand, petit et médian

Question 1. On 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 grand 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 calculeIndiceMaximum := proc (tab, a, b)
2   local k, indiceMax, PlusGrand;
3   PlusGrand := tab[a];
4   indiceMax := a;
5   for k from a+1 to b do
6     if tab[k] > PlusGrand
7       then indiceMax := k; PlusGrand := tab[k] fi;
8   od;
9   return (indiceMax);
10 end;
```

Question 2. On parcourt le sous-tableau $tab[a..b]$ à l'aide de la boucle des lignes 4 à 6 et on comptabilise le nombre d'éléments plus petits que **val** à l'aide du compteur **compt**.

```

1 nombrePlusPetit := proc (tab, a, b, val)
2   local compt, k;
3   compt := 0;
4   for k from a to b do
5     if tab[k] <= val then compt := compt+1 fi;
6   od;
7   return (compt);
8 end;
```

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 10 à 14, 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 16 à 20, 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 permute := proc (tab, a, b)
2   local tmp;
3   tmp := tab[a]; tab[a] := tab[b]; tab[b] := tmp;
4   end;
5
6 partition := proc (tab, a, b, indicePivot)
7   local Pivot, i, pointeur, positionPivot;
8   pointeur := a;
9   Pivot := tab[indicePivot];
10  for i from a to b do
11    if tab[i] < Pivot
12      then permute(tab, i, pointeur); pointeur := pointeur+1;
13    fi;
14  od;
15  positionPivot := pointeur;
16  for i from pointeur to b do
17    if tab[i] = Pivot
18      then permute(tab, i, pointeur); pointeur := pointeur+1;
19    fi;
20  od;
21  return (positionPivot);
22 end;
```

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

```

1 elementK := proc(tab , a , b , k)
2 local i;
3   if k = 1 and a=b
4     then return(tab[a])
5     else i := partition(tab , a , b , a);
6         if i-a+1 > k then return(elementK(tab , a , i-1 , k))
7         elif i-a+1 = k then return(tab.(a+k-1))
8         else return(elementK(tab , i+1 , b , k-i+a-1))
9         fi;
10    fi;
11 end;
```

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 $\mathcal{O}(n + (n-1) + \dots + 1) = \mathcal{O}(n^2)$. 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.

```

1 IndiceElementK := proc(tab , a , b , k)
2 local i;
3   if k = 1 and a=b
4     then return(a)
5     else i := partition(tab , a , b , a);
6         if i-a+1 > k then return(IndiceElementK(tab , a , i-1 , k))
7         elif i-a+1 = k then return(a+k-1)
8         else return(IndiceElementK(tab , i+1 , b , k-i+a-1))
9         fi;
10    fi;
11 end;
```

```

12 choixPivot := proc(tab , a , b)
13 local i , n;
14 n := floor((b-a+1)/5);
15 if n = 0
16   then elementK(tab , a , b , floor((b-a+1)/2+1))
17   else for i from 0 to n do
18     permute(tab , (a+i) , IndiceElementK(tab , a+5*i , a+5*(i+1)))
19   od;
20   permute(tab , (a+n) , IndiceElementK(tab , a+5*n , b , floor((b-5*n+a+1)/2))
21   choixPivot(tab , a , a+n);
22 fi;
23 fi;
24
25 end;
```

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

Question 7 :

```
1 coupeY := proc(tabX, tabY)
2   local n;
3     n := taille(tabX);
4     tabY[indiceMedian(tabY, 1, n)]
5 end;
```

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 $(x, y) - (x_2, y_2)$. On détermine et on renvoie l'angle médian.

```
1 demiDroiteMedianeSup := proc(tabX, tabY, x, y)
2
3   local i, n, tabAngle, compt;
4   n := taille(tabX);
5   tabAngle = alloue(n, 0);
6   compt := 0;
7   for i from 1 to n-1 do
8     if tabY.(i) > y
9       then compt := compt+1;
10        tabAngle[compt] := angle(x, y, tabX[i], tabY[i]);
11    fi;
12  od;
13  return(tabAngle[indiceMediant(tabAngle, 1, compt)]);
14 end;
```

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 18 à 22).

```
1 verifieAngleSecondeDroite := proc(tabX, tabY, x, y, theta)
2   local Pi, i, n, tabAngle, compt1, compt2, l2;
3   Pi := angle(0, 0, -1, 0);
4   n := taille(tabX);
5   tabAngle := alloue(n, 0);
6   compt1 := 0;
7
8   compt2 := 0;
9   for i from 1 to n do
10    if tab.(i) < y
11      then if angle(x, y, tabX[i]) < theta + Pi
12            then compt1 := compt1+1;
13              else compt2 := compt2+1;
14            fi;
15          fi
16    od;
17    l2 := ceil (compt1+compt2)/2;
18    if compt1 > l2 then return(1) fi;
19    if compt2 > l2 then return(-1);
20      else return(0)
21    fi;
22  end;
```

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 6 à 9. Dans la boucle conditionnelle des lignes 11 à 21, 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 16.

```

1 secondeMediane := proc(tabX, tabY, y)
2   local i, n, d, angle, x, alpha, beta, tabRes;
3   alpha := tabX[0];
4   beta := tabX[0];
5   tabRes := alloue(2, 0);
6   for i from 1 to n do
7     if tabX[i] < alpha then alpha := tabX[i] fi;
8     if tabX[i] > beta then beta := tabX[i] fi;
9   od;
10  while true do
11    x := (alpha+beta)/2;
12    angle := demiDroiteMedianeSup(tabX, tabY, x, x);
13    d := verifieAngleSecondeDroite(tabX, tabY, x, y, angle);
14    if d = 0 then tabRes[1] := x;
15                tabRes[2] := angle;
16                return(tabRes)
17    fi;
18    if d < 0 then beta := x
19                else alpha := x
20    fi;
21  od;
22  end;

```