X 2012 : Info pour non-informaticiens

Préparations d'exemples

En plus des exemples je redéfinis ci-dessous taille (alias de ArrayNumElems), AffGA (Affichage Graphique d'un Array sans être emb^été par les longueurs 10 ou 25)

Dans ceratianes fonctions, lors du debugging, je suis passé par des lists ... je n'ai peut être pas tout re-converti en Arrays

Pour ce corrigé, j'ai supprimé toutes mes traces de debugging : c'est bien sûr ce qu'il ne faut pas faire, c'est en révisant ses erreurs qu'on se trompe un peu moins

```
> restart;
> tab11 := Array(1..11, [3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8]);
```

$$tab11 := \begin{bmatrix} 1 .. 11\ Array \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad (1.1)$$

```
> taille := ArrayNumElems;
```

$$taille := ArrayNumElems \quad (1.2)$$

```
> AffGA := proc(a) convert(a, list) end;
```

$$AffGA := \mathbf{proc}(a)\ convert(a, list)\ \mathbf{end\ proc} \quad (1.3)$$

```
> AffGA(tab11);
```

$$[3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8] \quad (1.4)$$

"des fois" (ça m'est arrivé 3 ou 4 fois) quand on définit A:=B avec des Arrays on a un nouvel Array (= on peut modifier A et B indépendemment) et d'autres fois c'est le même (et quand on manipule l'autre ... l'initial ne peut plus servir à des initialisations). Au lieu d'essayer de comprendre 'la logique de Maple', je fabrique une fonction copie pour être tranquille

```
> copie := proc(a) Array(1..11, AffGA(a)) end;
```

$$copie := \mathbf{proc}(a)\ Array(1..11, AffGA(a))\ \mathbf{end\ proc} \quad (1.5)$$

```
> tab2 := copie(tab11); AffGA(tab2); AffGA(tab11); tab2[5] := 1000; AffGA(tab2);
    AffGA(tab11);
```

$$tab2 := \begin{bmatrix} 1 .. 11\ Array \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix}$$

$$[3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8]$$

$$[3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8]$$

$$tab2_5 := 1000$$

$$[3, 2, 5, 8, 1000, 34, 21, 6, 9, 14, 8]$$

$$[3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8] \quad \textbf{(1.6)}$$

Question 1

```
> calculeIndiceMaximum := proc(tab, a, b)
                 local sousmaxi;
                 if a = b
                 then a
                 else sousmaxi := calculeIndiceMaximum(tab, a + 1, b);
                      if tab[a] > tab[sousmaxi] then a else sousmaxi fi
                 fi  end:
> calculeIndiceMaximum(tab11, 1, 11);
```

6 **(2.1)**

Question 2

```
> nombrePlusPetit := proc(tab, a, b, val)
              local k, nbpt; nbpt := 0;
              for k from a to b do
              if tab[k] ≤ val then nbpt := nbpt + 1 fi
              od;
              nbpt
    end:
> nombrePlusPetit(tab11, 1, 11, 5);
```

4 **(3.1)**

Question 3

C'est un classique de la programmation ("drapeau tricolore de Dijkstra") de faire ce qui suit sans place supplémentaire ... sans recopie .... pour ce problème cela me semble du pur vice

http://iml.univ-mrs.fr/~lafont/licence/prog2.pdf est (parmi ... 1000?) une bonne référence

```
> partition := proc(a, b, indicePivot)
          global tab;
          local newtab, ig, id, k, pivot;
          newtab := array(1 .. taille(tab)); AffGA(newtab);
          ig := a; id := b; pivot := tab[indicePivot];
          for k from a to b do
          if tab[k] < pivot
          then newtab[ig] := tab[k]; ig := ig + 1
           else if tab[k] > pivot
                then   newtab[id] := tab[k]; id := id − 1
                fi
```

```
        fi
      od;
      for k from ig  to id do newtab[k] := pivot od;
       for k from a to b do tab[k] := newtab[k] od;
       ig;
  end:
```

```
> tab := tab11; AffGA(tab11); partition(1, 11, 11);
```

$$tab := \begin{bmatrix} 1 .. 11\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix}$$

$$[3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8]$$

$$6 \qquad \textbf{(4.1)}$$

```
> AffGA(tab11);
```

$$[3, 2, 5, 1, 6, 8, 8, 14, 9, 21, 34] \qquad \textbf{(4.2)}$$

Question 4

Après avoir dit (dans la question 3) que tab était global, voilà qu'on doit le passer en variable à elementK .... je le laisse global, na

```
> tab := copie(tab11) : AffGA(tab);
```

$$[3, 2, 5, 1, 6, 8, 8, 14, 9, 21, 34] \qquad \textbf{(5.1)}$$

```
> elementK :=proc(a, b, k)
      local rang, ip, val, val2; global tab;
        if a = b
        then tab(a)
        else ip := partition(a, b, b);#print(a, b, k, ip);
             if ip − a + 1 > k
             then val := elementK(a, ip − 1, k);
             else if ip − a + 1 = k
                  then tab[ip]
                  else val2 := elementK(ip + 1, b, k − ip + a − 1);
                  fi
             fi
        fi
  end:
```

```
> elementK(1, 11, 2);
```

$$2 \qquad \textbf{(5.2)}$$

Question 5

La fonction partition appliquée à un tableau de n choses fait n comparaisons (éventuellement (n-1) si on a pris un des éléments du tableau comme pivot)

Dans le pire cas elementK va récupérer un tableau de taille n-1 dans lequel chercher son k-ième

Si donc ce pire cas se reproduit à chaque fois, on aura au total fait $\sum_{k=1}^{n} k$ comparaisons : la complexité est alors $O(n^2)$

Question 6

Un bon "mediane de 5" prend [a,b,c,d,e] et
1 - trie [a,b] et [c,d] (-> 2 comparaisons) on obtient [a',b'] et [c',d']
2 - fusionne [a',b'] et [c',d'] en [a",b",c",d"] (-> 3 comparaisons car pour fusionner deux listes triées de longueurs p et q il suffit de p+q-1 comparaisons)
3 - insère e dans [a",b",c",d"] (-> 3 comparaisons en commençant par e et b")
4 - on prend le troisième TOTAL : 8

Qui plus est, on trie sur les valeurs, mais c'est le rang de la médiane finale que l'on veut, on pourrait faire tout cela avec un arbre de discussion : pour 5, les 256 cas seraient donc "au moins" 256 lignes dans le programme, et il faut y ajouter les études analogues des cas de listes de longueurs 2,3,4

Je vais ci-dessous écrire une très vilaine fonction de MiniMédiane (est-ce l'opposé de l'esprit de ce problème ... ?)

```
> numerote := proc(l) map(k->[l[k], k], [$1..nops(l)]) end: numerote([3, 5, 1]);
```

$$[[3, 1], [5, 2], [1, 3]] \quad (7.1)$$

```
> OrdreCouple:=proc(x, y) evalb(x[1] <= y[1]) end: OrdreCouple([5, 1], [2, 6]);
```

$$false \quad (7.2)$$

```
> MiniTri := proc(l) sort(numerote(l), OrdreCouple) end: MiniTri([3, 5, 1]);
```

$$[[1, 3], [3, 1], [5, 2]] \quad (7.3)$$

```
> MiniMediane := proc(l) MiniTri(l)[ceil(nops(l)/2)] end: MiniMediane([3, 5, 1, 14, 14]);
```

$$[5, 2] \quad (7.4)$$

```
> RangMiniMediane := proc(l) MiniMediane(l)[2] end: RMM := RangMiniMediane;
```

$$RMM := RangMiniMediane \quad (7.5)$$

```
> map(RMM, [[3, 8, 1], [3, 3, 3, 0], [1, 11, 111, 27], [1, 2, 5, 4, 3]]);
```

$$[1, 1, 2, 5] \quad (7.6)$$

L'exemple donné dans l'énoncé semble dire que ChoixPivot est défini récursivement en choisissant le pivot parmi l'ensemble MiniMedianes ... ça aussi c'est encore (informatiquement sain mais) en désaccord avec les usages de l'X qui après avoir préconisé la récursion s'est mis à l'interdire

La fonction selection qui suit va utiliser un tab global, son but est de rendre le k-

ième des termes de tab qui sont rangés entre les indices i et j. "Selon le bon livre" (Aho-Hopcroft-Ullmann) il faut discuter ainsi sur la longueur de la liste :
Cas petit : quand j-i est inférieur strict à 74 on fait un tri (par exemple trifusion= mergesort dont la complexité est inférieure à n.log(2,n)) puis on prend le k-ième : complexité fixe, majorée par 500
Cas grand : quand j-i>=75
on divise la liste en paquets de ≤ 5 dont on détermine les MiniMediane : on a ~ n/5 telles MM
on selectionne la médiane de cet ensemble de MiniMedianes : elle est $\geq$ ~ n/10 autre MM qui sont chacune ≥ 3 termes de la liste initiale (elle sont supérieures à elles mêmes) elle est finalement $\geq$ 30% des termes de la liste initiale et elle est de même $\leq$ ~ 0.3 n des termes
on use de partition pour "fendre" la liste en deux sous listes : la plus longue des deux sera de longueur au plus 0.7 n

On vient de réduire le problème "sélection parmi n" = S(n) à
La sélection de la Médiane des MiniMedianes " : S(n/5) = S(0.2 n)
la sélection parmi la liste non éliminée : S(0.7 n)

"Nyapuka" tenir compte des n<75, entrer là dedans la complexité (linéaire) de 'partition', la complexité des MiniMedianes ... à chaque fois que je l'ai fait j'ai trouvé que ça ne marchait pas, quand on lit AHU, ça marche facile

Détails de calcul : voir Aho-Hopcroft-Ullman pages 290 à 292

finalement, je n'écris aucune fonction ChoixDuPivot, j'en ferai une utilisant sort (donc en O(n.log(n))) pour pouvoir continuer

j'en écris une quand même ...

```
> map(tab11, [$11..11]);
```

$$[34] \qquad (7.1.1)$$

```
> coupeEnPaquets5 := proc(A)
    local long, tas, nbP;
      long := taille(A); nbP := iquo(long, 5);
      tas := seq(map(k→A[5·i+k], [$1..5]), i=0..nbP−1);
      if long > 5·nbP
      then tas := tas, map(A, [$5·nbP+1..long])
      fi;
      [tas]
  end:
> coupeEnPaquets5(tab11);
```

$$[[3, 2, 5, 1, 6], [8, 8, 14, 9, 21], [34]] \qquad (7.1.2)$$

```
> ExtraitMM := proc(ll) map(sl→sl[RMM(sl)], ll) end;
```

$$ExtraitMM := \mathbf{proc}(ll)\ map(sl \rightarrow sl[RMM(sl)], ll)\ \mathbf{end\ proc} \qquad (7.1.3)$$

```
> ExtraitMM(coupeEnPaquets5(tab11));
```

(7.1.4)

```
                                  [3, 9, 34]                                  (7.1.4)
```

ma fonction "fend" sépare en petits, egaux, grands et rend
[petits, egaux, grands, longueur-de-petit-plus-egaux]

```
> fend := proc(A, pivot)
    local long, k, petits, egal, grands;
      petits := NULL; egal := NULL; grands := NULL; long := taille(A);
      for k from 1 to long
       do print(k, A[k], [petits], [egal], [grands]);
          if A[k] = pivot
          then egal := pivot, egal
          else if A[k] < pivot
               then petits := A[k], petits
               else grands := A[k], grands
               fi
          fi
       od;
  convert([petits, egal, grands, nops([petits, egal])], Array);
  end:
> kIemeLineaire := proc(A, k)
     local long, clubMed, pivot, fendu, compte;
    long := nops(A);
    if long ≤ 5
    then sort(A)[k]
    else clubMed := ExtraitMM(coupeEnPaquets5(A));
         pivot := kIemeLineaire(clubMed, iquo(nops(clubMed), 2));
         fendu := fend(A, pivot);
         compte := fendu[-1];
         if compte ≥ k
        then kIemeLineaire(fendu[1..compte], k)
        else kIemeLineaire(fendu[compte + 1..-2], k − compte)
         fi
    fi
  end:
> AffGA(tab11);
                          [3, 2, 5, 1, 6, 8, 8, 14, 9, 21, 34]                (7.1.5)
> map(x→kIemeLineaire(tab11, x), [$1..taille(tab11)]);
                          [1, 2, 3, 5, 6, 8, 8, 9, 14, 21, 34]                (7.1.6)
```

Question 7

J'ai traité "après" la Q6. Pour avoir une médiane utilisable dans la suite je vais utiliser ci-dessous IndiceMedian de complexité nlog(n) (je présume que le tri de Maple en O(n.ln(n)))

```
> with(plottools) : with(plots) :
> IndiceMedian := proc(tab) local long, tabi, stabi;
      long := nops(tab); tabi := map(k→[tab[k], k], [$1..long]);
      stabi := sort(tabi, (x, y)→(x[1] ≤ y[1]));
```

```
        stabi[ceil(long/2), 2]
    end:
> IndiceMedian([23, 42, 1, 5, 9, 2, 6, 7, 3, 4, 8, 10]);
                                    7                                    (8.1)
```

Mes noms :

CoupeY : va fournir l'ordonnée médiane

ExempleX : j'ai relevé les coordonnées du dessin de l'énoncé X 2012

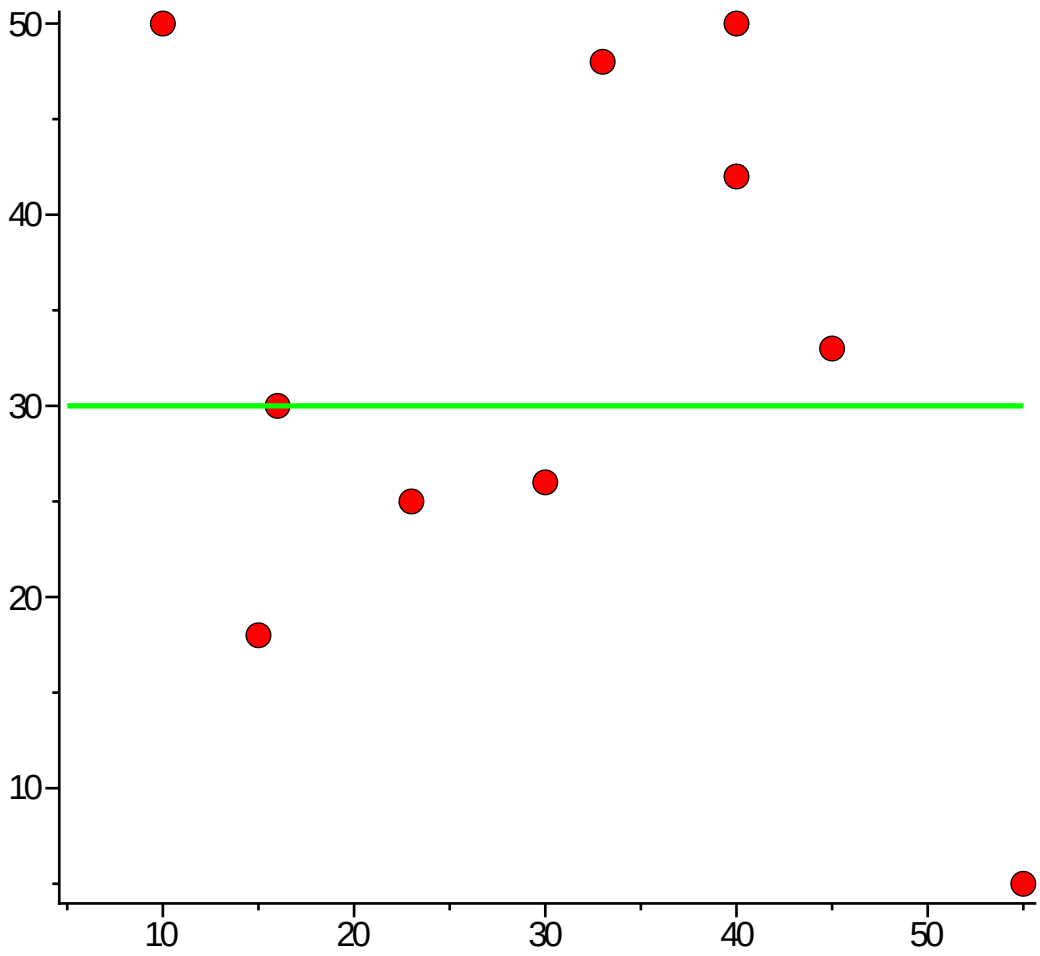
ExempleXX : les abcisses

ExempleXY : les ordonnées

CH : la valeur de la coupe horizontale

il n'y a plus qu'à transformer en images et à afficher

```
> CoupeY := proc(tabX, tabY) tabY[IndiceMedian(tabY)] end:
> ExempleX := [[10, 50], [33, 48], [40, 50], [40, 42], [45, 33], [16, 30], [23, 25], [15, 18],
      [30, 26], [55, 5]]:
> ExXX := map(t→t[1], ExempleX) : ExXY := map(t→t[2], ExempleX) :
> CH := CoupeY(ExXX, ExXY);
                                  CH := 30                               (8.2)
> MontreCoupeY := line([5, CH], [55 , CH], color = green, thickness = 2);
MontreCoupeY := CURVES([[5., 30.], [55., 30.]], COLOUR(RGB, 0., 1.00000000, 0.),   (8.3)
    THICKNESS(2))
> ImExX := map(k→disk(k, 0.61, color = red), ExempleX) : display(ImExX, MontreCoupeY,
      scaling = constrained);
```



Question 8

Je nomme AngleBarbelé la détermination des angles dans C \ R^- , l'axe réel négatif étant couvert de barbelés

```
> angleBarb := proc(x, y) 2·arctan( y / (x + sqrt(x^2 + y^2)) ) end:
```

```
> demiDroiteMedianeSup := proc(tabX, tabY, x, y)
      local PointsNum, IndicesNord, PointsNord, CardNord, PointsNordX, VectNordX,
  PointsNordY, VectNordY, AnglesNord;
      PointsNum := numerote(tabY);
      IndicesNord := map(C→C[2], select(z→evalb(z[1] > y), PointsNum));
      CardNord := nops(IndicesNord);
      PointsNordX := map(k→tabX[k], IndicesNord); VectNordX := map(k→k − x,
  PointsNordX);
      PointsNordY := map(k→tabY[k], IndicesNord); VectNordY := map(k→k − y,
  PointsNordY);
      AnglesNord := evalf( map(k→ angleBarb(VectNordX[k], VectNordY[k]), [$1
  ..CardNord]) );
      AnglesNord[IndiceMedian(AnglesNord)];
```

```
end:
> AffdDMS := proc(tabX, tabY, x, y) local ang; ang := demiDroiteMedianeSup(tabX, tabY, x,
      y); line([x, y], [x + 40·cos(ang), y + 40 sin(ang)] ) end:
> demiDroiteMedianeSup(ExXX, ExXY, 55, 30);
```

2.466851712 **(9.1)**

```
> map(k→demiDroiteMedianeSup(ExXX, ExXY, ExXX[k], 30), [$1 ..nops(ExXX) ]);
```

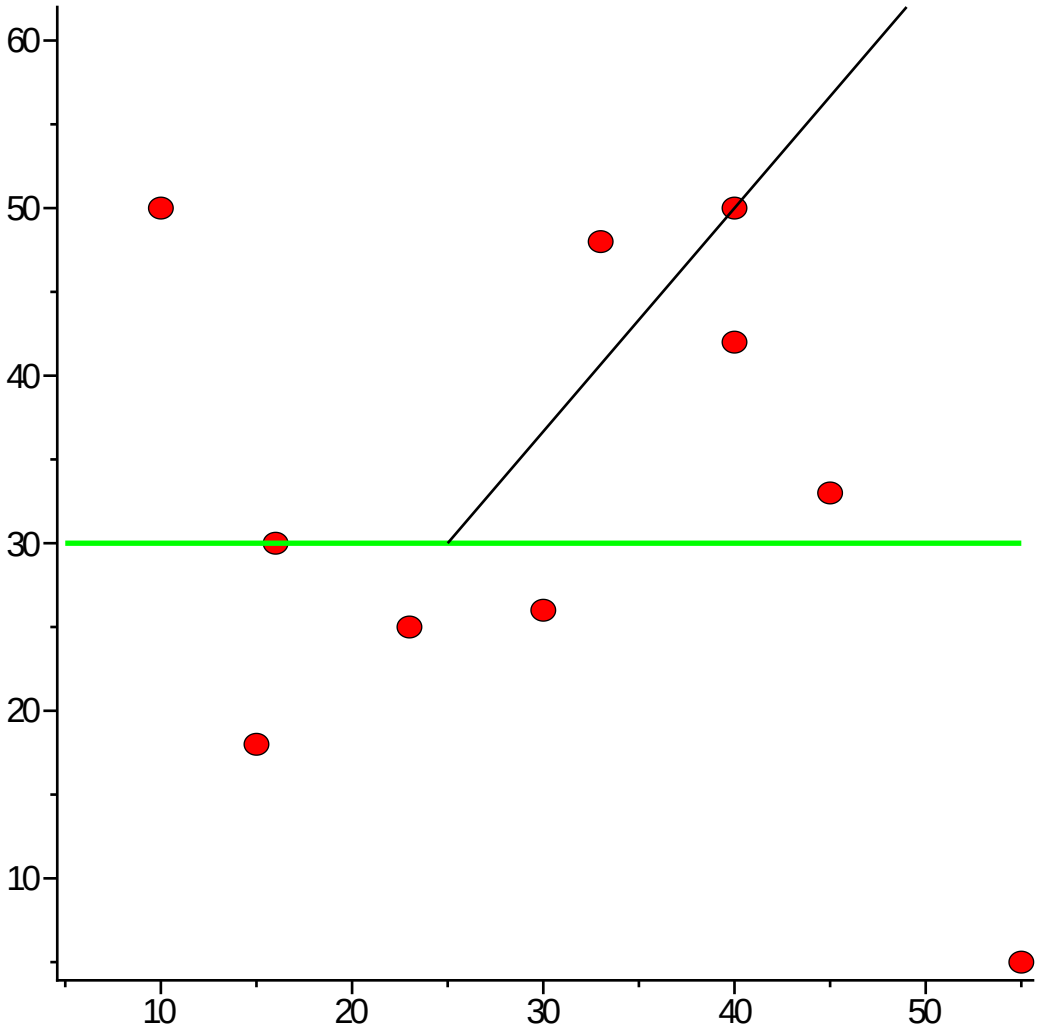
[0.5880026036, 1.234121507, 1.570796327, 1.570796327, 1.965587446, 0.6947382762, 0.8663022624, 0.6747409422, 1.107148718, 2.466851712] **(9.2)**

```
> montretout := proc(tabX, tabY, x, y) [ImExX, MontreCoupeY, AffdDMS(tabX, tabY, x, y) ]
      end;
```

```
montretout := proc(tabX, tabY, x, y)
    [ImExX, MontreCoupeY, AffdDMS(tabX, tabY, x, y) ]
end proc
```

(9.3)

```
> display(op(montretout(ExXX, ExXY, 25, 30) ) );
```




```
> IndiceMedian([2.214297436, 1.152571997, 0.9272952180, 0.6747409425, 0.1488899467]);
```

3 **(9.4)**

Question 9

Je trouve le nom plus parlant a*vec* VerifieAngleSud

Pour séparer en ceux < et ceux
> theta on est "presque" à réutiliser la fonction déja écrite "partition",
mais on ne veut pas les nombres mais seulement savoir combien on en a

Le tableau exemple est amusant : *il n'y a aucun nombre qui le partage en deux* !
Un autre exemple plus net : un tableau constant, il sera 'toujours'
fendu en morceaux inégaux par les algorithmes de ce problème

```
> compte := proc(liste, valeur)
    local k, petit, grand;
          petit := 0; grand := 0;
          for k from 1 to nops(liste)
          do if liste[k] < valeur
             then petit := petit + 1
             else  if liste[k] > valeur
                   then grand := grand + 1
                   fi;
             fi;
          od;
    if grand > petit then 1
    elif grand < petit then -1
    else 0
    fi
  end:
```

```
> LEx := [3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8];
```

$$LEx := [3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8] \quad \textbf{(10.1)}$$

```
> compte(LEx, 4); compte(LEx, 8); compte(LEx, 20);
```

$$1$$

$$-1$$

$$-1 \quad \textbf{(10.2)}$$

```
> VerifieAngleSud := proc(tabX, tabY, x, y, theta)
    local PointsNum, IndicesNord, PointsNord, IndicesSud, CardSud, PointsSudX, VectSudX,
    PointsSudY, VectSudY, AnglesSud, PiN;
      PiN := evalf(Pi);
      PointsNum := numerote(tabY);
      IndicesSud := map(C->C[2], select(z->evalb(z[1] < y), PointsNum));
      CardSud := nops(IndicesSud);#print("CS", CardSud);
      PointsSudX := map(k->tabX[k], IndicesSud); VectSudX := map(k->k - x,
    PointsSudX);
      PointsSudY := map(k->tabY[k], IndicesSud); VectSudY := map(k->k - y,
    PointsSudY);
      AnglesSud := evalf( map(k-> angleBarb(VectSudX[k], VectSudY[k]), [$1
    ..CardSud]));
      compte(AnglesSud, theta - PiN);
```

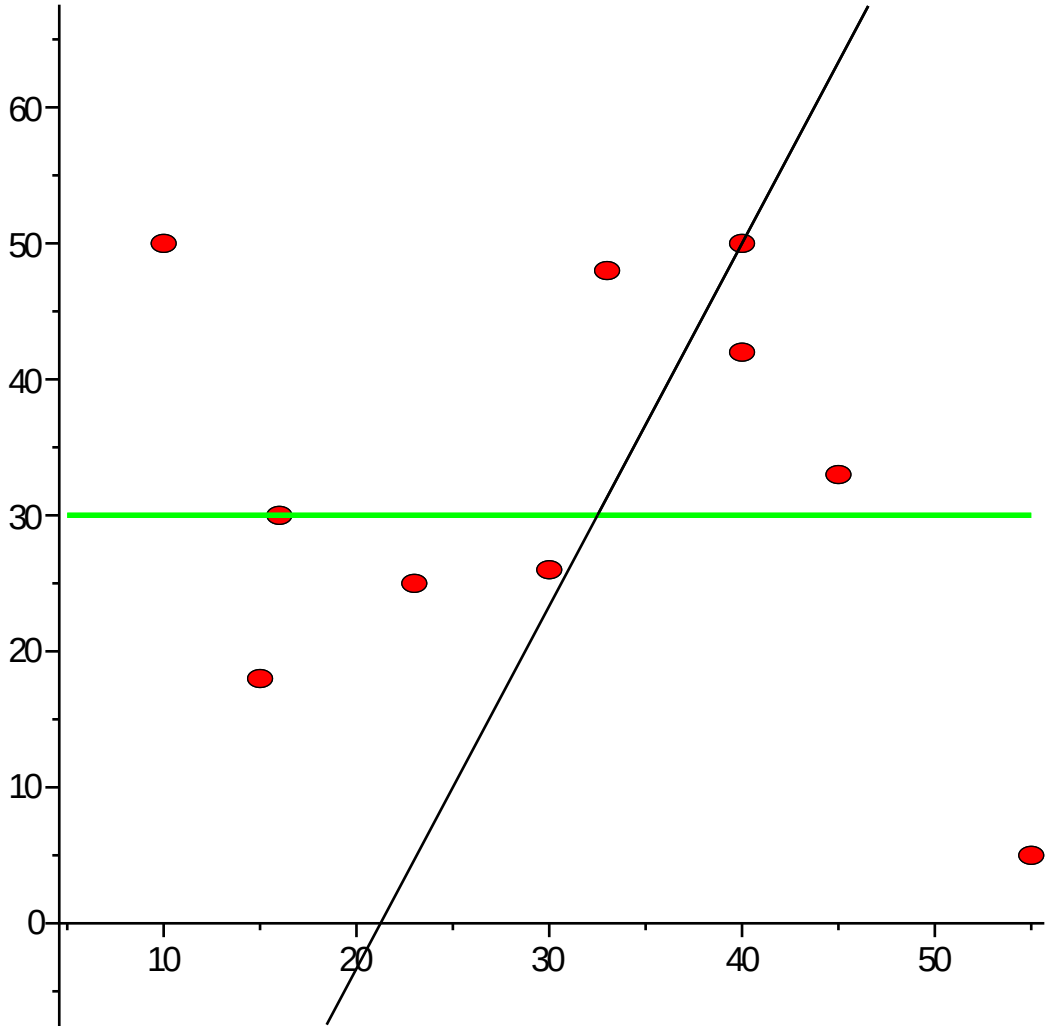
```
end:
> demiDroiteMedianeSup(ExXX, ExXY, 32.5, 30);
```

$$1.212025657 \qquad (10.3)$$

```
> VerifieAngleSud(ExXX, ExXY, 32.5, 30, demiDroiteMedianeSup(ExXX, ExXY, 32.5, 30));
```

$$-1 \qquad (10.4)$$

```
> AffdDMI := proc(tabX, tabY, x, y) local ang; ang := demiDroiteMedianeSup(tabX, tabY, x,
      y); line([x, y], [x-40·cos(ang), y-40 sin(ang)]) end:
> montretout2 := proc(tabX, tabY, x, y) [ImExX, MontreCoupeY, AffdDMS(tabX, tabY, x, y),
      AffdDMI(tabX, tabY, x, y)] end:
> display(op(montretout(ExXX, ExXY, 32.5, 30)), op(montretout2(ExXX, ExXY, 32.5, 30)));
```



Question 10

```
> secondeMediane := proc(tabX, tabY, y)
      local xmin, xmax, essaye;
        essaye := proc(gauche, droite)
          local P, aN, res;
          P := evalf( (gauche + droite)/2 );
```

```
        aN := demiDroiteMedianeSup(tabX, tabY, P, y);
        res := VerifieAngleSud(tabX, tabY, P, y, aN);
        if res = 0
        then [P, aN]
        else if res = 1
             then essaye(P, droite)
             else essaye(gauche, P)
             fi
        fi
    end;
  xmin := min(tabX); xmax := max(tabX);
  essaye(xmin, xmax)
end:
> ExXY[IndiceMedian(ExXY)];
```

$$30 \qquad (11.1)$$

```
> secondeMediane(ExXX, ExXY, ExXY[IndiceMedian(ExXY)]);
```

$$[26.87500000, 0.9900399732] \qquad (11.2)$$

```
> display(op(montretout(ExXX, ExXY, 26.875, 30)), op(montretout2(ExXX, ExXY, 26.875,
    30)));
```