CORRIGÉ TD N°7

(Tronc commun-1ère année)

Exercice n°1 : Recherche du minimum d'un tableau (liste Maple).

```
> mini := proc(t::list)
  local i, l, m;
  l := nops(t);
  if l = 0
       then RETURN(‘Erreur :  tableau vide‘);
       else
           m := t[1];
           for i from 1 to l do
                if t[i] < m then m := t[i] fi;
           od;
           RETURN(m)
  fi;
  end;
```

mini := **proc**(*t*::*list*)
local *i*, *l*, *m*;
l := nops(*t*) ;
if *l* = 0 **then** RETURN(‘*Erreur : tableau vide*‘)
else *m* := t_1 ; **for** *i* **to** *l* **do** **if** t_i < *m* **then** *m* := t_i **end if** **end do** ; RETURN(*m*)
end if
end proc

```
> mini([10,1,2,3,4,5,-17,6,7,9,56,5,-200]);
```

$$-200$$

```
> mini([]);
```

Erreur : tableau vide

Exercice n°2 : Recherche du maximum d'un tableau (liste Maple).

```
> maxi := proc(t::list)
  local i, l, m;
  l := nops(t);
  if l = 0
       then RETURN(‘Erreur :  tableau vide‘);
       else
           m := t[1];
           for i from 1 to l do
               if t[i] > m then m := t[i] fi;
           od;
           RETURN(m)
  fi;
  end;
```

maxi := **proc**(*t*::*list*)
local *i*, *l*, *m*;
l := nops(*t*) ;
if *l* = 0 **then** RETURN(‘*Erreur : tableau vide*‘)
else *m* := t_1 ; **for** *i* **to** *l* **do** **if** *m* < t_i **then** *m* := t_i **end if** **end do** ; RETURN(*m*)
end if
end proc

```
> maxi([10,1,2,3,4,5,-17,6,7,9,56,5,-200]);
```

Exercice n°3 : fonction qui calcule le produit scalaire de 2 vecteurs.

```
> prodscal := proc(t1,t2 ::  list)
  local l1, l2, i, s;
  l1 := nops(t1); l2 := nops(l2);
     if l1 = 0 then RETURN(‘Erreur :  un des vecteurs est vide‘);
  elif l2 <> l2 then RETURN(‘Erreur :  les deux vecteurs n’ont pas la
  même longueur‘);
  else
       s:= 0;
       for i from 1 to nops do
           s := s + t1[i] * t2[i];
           od;
       RETURN(s);
  fi;
  end;
```

prodscal := **proc**(*t1*, *t2*::*list*)
local *l1*, *l2*, *i*, *s*;
l1 := nops(*t1*) ;
l2 := nops(*l2*) ;
if *l1* = 0 **then** RETURN(‘*Erreur : un des vecteurs est vide*‘)
elif *l2* ≠ *l2* **then**
RETURN(‘*Erreur : les deux vecteurs n′ont pas la mme longueur*‘)
else *s* := 0 ; **for** *i* **to** *nops* **do** *s* := *s* + *t1*$_i$ ∗ *t2*$_i$ **end do** ; RETURN(*s*)
end if
end proc

Exercice n°4 : fonction qui retourne la première position d'un élément dans un tableau (-1 si l'élément n'est pas dans le tableau).

```
> position := proc(e, t ::  list)
  local i;
  for i from 1 to nops(t) do
       if t[i] = e then RETURN(i) fi;
       od;
  RETURN(-1)
  end;
```

position := **proc**(*e*, *t*::*list*)
local *i*;
for *i* **to** nops(*t*) **do if** $t_i = e$ **then** RETURN(*i*) **end if end do** ; RETURN(−1)
end proc

```
> position(5,[1,2,3,11,12,15,4,5,6,7,8,9,5,4,6]);
```

8

```
> position(5,[]);
```

−1

Exercice n°5 : recherche dichotomique itérative dans un tableau rangé en ordre croissant.

```
> dichoto := proc(e, t ::  list)
  local a, b, m;
  a := 1; b := nops(t);
  while (b-a) >= 0 do
       m := floor((a+b)/2);
         if  e < t[m] then b := m-1
       elif e > t[m] then a := m+1
       else RETURN(true)
       fi;
  od;
  RETURN(false)
  end;
```

$$
\begin{array}{l}
dichoto := \mathbf{proc}(e,\ t{::}list) \\
\mathbf{local}\ a,\ b,\ m; \\
\quad a := 1\,; \\
\quad b := \mathrm{nops}(t)\,; \\
\quad \mathbf{while}\ 0 \le b - a\ \mathbf{do} \\
\quad\quad m := \mathrm{floor}(1/2 * a + 1/2 * b)\,; \\
\quad\quad \mathbf{if}\ e < t_m\ \mathbf{then}\ b := m - 1 \\
\quad\quad \mathbf{elif}\ t_m < e\ \mathbf{then}\ a := m + 1 \\
\quad\quad \mathbf{else}\ \mathrm{RETURN}(true) \\
\quad\quad \mathbf{end\ if} \\
\quad \mathbf{end\ do}; \\
\quad \mathrm{RETURN}(false) \\
\mathbf{end\ proc}
\end{array}
$$

```
> dichoto(8,[1,2,3,4,5,6,7,9,10,12,13,14,14,15,16,17,18,19,22,23,28,44,
  45,46]);
```

$$false$$

```
> dichoto(15,[1,2,3,4,5,6,7,9,10,12,13,14,14,15,16,17,18,19,22,23,28,44
  ,45,46]);
```

$$true$$

Exercice n°6 : recherche dichotomique récursive dans un tableau rangé en ordre croissant.

```
> dichotorec := proc(e, t ::  list)
  local l, m;
  l := nops(t);              # calcul de la longueur du tableau
  if nops(t) = 0
       then
           RETURN(false)        # cas du tableau vide
       else
           m := floor((nops(t)+1)/2);
             if e < t[m] then RETURN(dichotorec(e,t[1..m-1]))
           elif e > t[m] then RETURN(dichotorec(e,t[m+1..l]))
           else RETURN(true)    # cas où t[m] = e
           fi;
  fi;
  end;
```

$$
\begin{array}{l}
\mathit{dichotorec} := \mathbf{proc}(e,\ t{::}\mathit{list}) \\
\mathbf{local}\, l,\ m; \\
\quad l := \mathrm{nops}(t)\,; \\
\quad \mathbf{if}\, \mathrm{nops}(t) = 0\, \mathbf{then}\, \mathrm{RETURN}(\mathit{false}) \\
\quad \mathbf{else} \\
\qquad m := \mathrm{floor}(1/2 * \mathrm{nops}(t) + 1/2)\,; \\
\qquad \mathbf{if}\, e < t_m\, \mathbf{then}\, \mathrm{RETURN}(\mathrm{dichotorec}(e,\ t_{1..m-1})) \\
\qquad \mathbf{elif}\, t_m < e\, \mathbf{then}\, \mathrm{RETURN}(\mathrm{dichotorec}(e,\ t_{m+1..l})) \\
\qquad \mathbf{else}\, \mathrm{RETURN}(\mathit{true}) \\
\qquad \mathbf{end\ if} \\
\quad \mathbf{end\ if} \\
\mathbf{end\ proc}
\end{array}
$$

```
> dichotorec(8,[1,2,3,4,5,6,7,9,10,12,13,14,14,15,16,17,18,19,22,23,28,
  44,45,46]);
```

$$\mathit{false}$$

```
> dichotorec(15,[1,2,3,4,5,6,7,9,10,12,13,14,14,15,16,17,18,19,22,23,28
  ,44,45,46]);
```

$$\mathit{true}$$

```
> trace(dichotorec);
```

$$\mathit{dichotorec}$$

```
> dichotorec(8,[1,2,3,4,5,6,7,9,10,12,13,14,14,15,16,17,18,19,22,23,28,
  44,45,46]);

{--> enter dichotorec, args = 8, [1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13,
14, 14, 15, 16, 17, 18, 19, 22, 23, 28, 44, 45, 46]
```

$$l := 24$$
$$m := 12$$

```
{--> enter dichotorec, args = 8, [1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13]
```

$$l := 11$$
$$m := 6$$

```
{--> enter dichotorec, args = 8, [7, 9, 10, 12, 13]
```

$$l := 5$$
$$m := 3$$

```
{--> enter dichotorec, args = 8, [7, 9]
```

$$l := 2$$
$$m := 1$$

```
{--> enter dichotorec, args = 8, [9]
```

$$l := 1$$
$$m := 1$$

```
{--> enter dichotorec, args = 8, []
```

$$l := 0$$

```
<-- exit dichotorec (now in dichotorec) = false}

<-- exit dichotorec (now in dichotorec) = false}

<-- exit dichotorec (now in dichotorec) = false}

<-- exit dichotorec (now in dichotorec) = false}

<-- exit dichotorec (now in dichotorec) = false}

<-- exit dichotorec (now at top level) = false}
```

false

Exercice n°7 : fonction qui calcule l'isobarycentre d'un famille de points.

```
> barycentre := proc(t ::  list)
  local i, n, dim, s;
  n := nops(t);
  if l = 0 then RETURN(‘Erreur :  liste vide‘) fi;
  dim := nops(t[1]);
  # On vérifie que tous les vecteurs ont la même longueur
  s := t[1];
  for i from 2 to n do
       if nops(t[i]) <> dim
          then RETURN(‘Erreur dans la dimension des tableaux‘)
          else s := s + t[i] fi;
       od;
  RETURN(s/n)
  end;
```

barycentre := **proc**(*t*::*list*)
local *i*, *n*, *dim*, *s*;
n := nops(*t*) ;
if *l* = 0 **then** RETURN(‘*Erreur : liste vide*‘) **end if**;
dim := nops(t_1) ;
s := t_1 ;
for *i* **from** 2 **to** *n* **do**
if nops(t_i) ≠ *dim* **then** RETURN(‘*Erreur dans la dimmension des tableaux*‘)
else *s* := *s* + t_i
end if
end do;
RETURN(*s*/*n*)
end proc

```
> barycentre([[1,2],[2,3],[4,5]]);
```

$$[\frac{7}{3}, \frac{10}{3}]$$

```
> barycentre([[4,1,2],[0,2,3],[4,5,4]]);
```

$$[\frac{8}{3}, \frac{8}{3}, 3]$$

Exercice n°8 : Tri d'un tableau (liste Maple)

```
> tri := proc(t ::  list)
  local i, j, n, m, a, tt;
  tt := t;
  n := nops(tt);
  if n = 0 then RETURN(‘Erreur :  tableau vide‘) fi;
  for i from 1 to n-1 do
       m := tt[i]; a := i;
       for j from i+1 to n do  # boucle pour trouver le plus  petit élément et sa position
           if tt[j] < m then a := j; m := tt[j] fi;
       od;
       tt[a] := tt[i]; tt[i] := m; # On intervertit l'élément en position i avec le
plus petit élément
  od;
  RETURN(tt);
  end;
```

```
tri := proc(t::list)
local i, j, n, m, a, tt;
  tt := t;
  n := nops(tt);
  if n = 0 then RETURN(‘Erreur : tableau vide‘) end if;
  for i to n - 1 do
    m := tt_i;
    a := i;
    for j from i + 1 to n do if tt_j < m then a := j; m := tt_j end if end do;
    tt_a := tt_i;
    tt_i := m
  end do;
  RETURN(tt)
end proc
```

```
> l := [seq(rand(1000)(),i=1..100)]; tri(l); ‘temps mis (en s.)‘ =
  time(tri(l));
```

$l :=$ [674, 171, 464, 717, 424, 879, 571, 177, 504, 831, 953, 668, 18, 982, 649, 385, 792,
872, 48, 783, 198, 358, 264, 229, 251, 952, 879, 801, 796, 901, 768, 946, 369,
$\cdots$,
246, 341, 848, 794, 711, 714, 639, 381, 685, 729, 124, 694, 888, 505, 754, 491]

[1, 4, 4, 18, 37, 38, 47, 48, 64, 69, 77, 92, 101, 124, 171, 177, 198, 222, 229, 243, 245,
ldots,
871, 872, 879, 879, 888, 892, 901, 911, 946, 950, 951, 952, 953, 956, 982]

$$\textit{temps mis (en s.)} = .549$$

Exercice n°9 : Tri par bulles

```
> tribulle := proc(t ::  list)
  local i, j, n, temp, tt;
  n := nops(t); tt := t;
  if n = 0 then RETURN(‘Erreur :  tableau vide‘) fi;
  for i from n to 2 by -1 do
       for j from 1 to n-1 do
           if tt[j] > tt[j+1] then temp := tt[j]; tt[j] := tt[j+1]; tt[j+1] := temp
fi;
       od;
  od;
  RETURN(tt);
  end;
```

```
tribulle := proc(t::list)
local i, j, n, temp, tt;
  n := nops(t);
  tt := t;
  if n = 0 then RETURN(‘Erreur : tableau vide‘) end if;
  for i from n by -1 to 2 dofor j to n - 1 do
        if tt_{j+1} < tt_j then temp := tt_j; tt_j := tt_{j+1}; tt_{j+1} := temp end if
    end do
  end do;
  RETURN(tt)
end proc
```

```
> l := [seq(rand(1000)(),i=1..100)];
  tribulle(eval(l)); ‘temps mis (en s.)‘ = time(tribulle(l));
```

$l :=$ [800, 434, 575, 94, 662, 701, 834, 972, 413, 53, 911, 488, 921, 830, 343, 163, 879,
$\cdots$,
444, 614, 716, 685, 316, 636, 594, 689, 555, 660, 758, 705, 662, 307, 507, 864,
600, 9, 803, 845, 82, 880, 398, 773, 136, 631, 323, 77, 985, 932, 663, 420, 202]

[9, 21, 53, 59, 77, 77, 82, 89, 94, 133, 136, 150, 155, 163, 202, 203, 229, 239, 242, 245,
307, 316, 323, 343, 358, 359, 367, 398, 413, 415, 420, 425, 434, 444, 444, 445,
$\cdots$
864, 866, 869, 875, 879, 880, 911, 913, 913, 915, 918, 921, 932, 968, 972, 985]

temps mis (en s.) $= 9.533$

L'exécution des programmes de tri précédents sur des grands tableaux nous donne ceci:

```
> l := [seq(rand(1000)(),i=1..1000)]:  tri(l);
```

```
Error, (in tri) assigning to a long list, please use arrays
```

Ce qui nous oblige à travailler avec le type "**array**" (ici il y a une incohérence de Maple où deux types qui n'ont pas le même comportement se représente de la même manière). Pour palier aux limitations du type **list,** nous pouvons réécrire le programme de la manière suivante :

```
> tri := proc(t ::  array)
  local i, j, n, m, a, tt;
  tt := t;
  n := nops(convert(tt,list));   # artifice pour déterminer la longueur du tableau
  if n = 0 then RETURN(‘Erreur :  tableau vide‘) fi;
  for i from 1 to n-1 do
       m := tt[i]; a := i;
       for j from i+1 to n do  # boucle pour trouver le plus  petit élément et sa position
           if tt[j] < m then a := j; m := tt[j] fi;
       od;
      # On intervertit l’élément en position i avec le plus petit élément
       tt[a] := tt[i]; tt[i] := m;
  od;
  RETURN(eval(tt));
  end;
```

tri := **proc**(*t*::*array*)
local *i*, *j*, *n*, *m*, *a*, *tt*;
tt := *t*;
n := nops(convert(*tt*, *list*));
if *n* = 0 **then** RETURN(‘*Erreur : tableau vide*‘) **end if**;
for *i* **to** *n* − 1 **do**
m := *tt*$_i$;
a := *i*;
for *j* **from** *i* + 1 **to** *n* **do if** *tt*$_j$ < *m* **then** *a* := *j*; *m* := *tt*$_j$ **end if end do**;
tt$_a$:= *tt*$_i$;
tt$_i$:= *m*
end do;
RETURN(eval(*tt*))
end proc

```
> l := convert([seq(rand(1000)(),i=1..1000)],array): ‘temps mis (en s.)‘ = time(tri(l));
```

temps mis (en s.) $= 47.399$

Pour évaluer la complexité de nos deux programmes, il nous faut choisir des opérations pertinentes et compter le nombre de fois que l'on fait appel à ces opérations en fonction de n, la longueur du tableau. On peut remarquer que les opérations auxquelles on fait le plus appel sont :

- l'accès à l'élément d'un tableau,
- la comparaison de deux éléments,
- l'échange de deux éléments d'un tableau.

Un décompte des opérations précédentes pour nos deux programmes donne le tableau suivant :

	nb d'accès à un élément	nb de comparaisons	nb d'échanges de 2 éléments
tri	$\sim n(n+1)/2$	$\sim n(n+1)/2$	n
tribulle	$\sim n(n+1)$	$\sim n(n+1)/2$	$< n(n+1)/2$

Sur le papier, le premier programme semble plus avantageux que le second, ce qui semble se confirmer par les temps de calcul observés.