Énoncé

Un tableau de nombres étant donné, on doit ranger les valeurs de ce tableau par ordre croissant.
On considère deux méthodes : le *tri par insertion* et le *tri par fusion*. On va implémenter des tris "en place" en utilisant des variables globales pour désigner le tableau et la taille du tableau. Le tableau initial sera donc modifié par les procédures. Les noms utilisés sont

`A` : tableau donné

`n` : entier ; `A` est indexé de 1 à `n` .

Des corrigés sont proposés sur une feuille de travail à télécharger.

1. Tri par insertion.

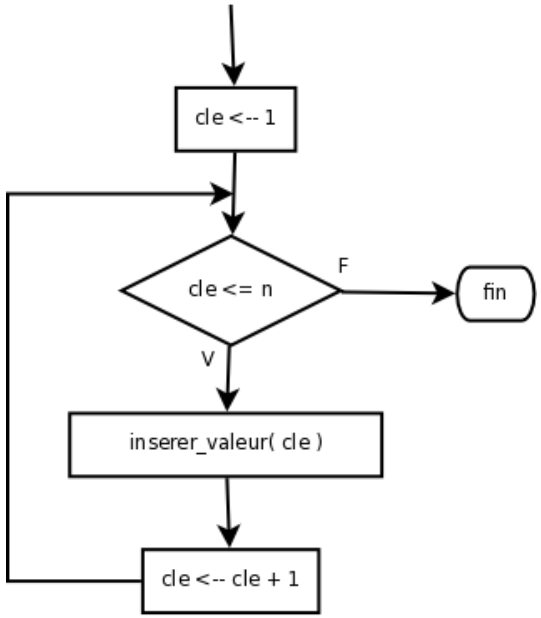

FIG. 1 – Tri par insertion : procédure principale

Le principe de la procédure principale `ins()` est le suivant (figure 1) :

- Parcourir le tableau donné de 1 à `n`.
- Insérer la *valeur courante* du tableau dans le début du tableau qui se trouve ainsi partiellement ordonné.

La procédure principale utilise une autre procédure `inserer_valeur(p)` qui insère à la bonne place la valeur `A[p]` dans le tableau entre 1 et `p`. Il est à noter que au moment de l'appel de `inserer_valeur(p)`, les valeurs de 1 à `p-1` sont croissantes. La tâche de `inserer_valeur(p)` est donc de trouver la bonne place et d'y placer `A[p]` en décalant les valeurs suivantes (figure 2).

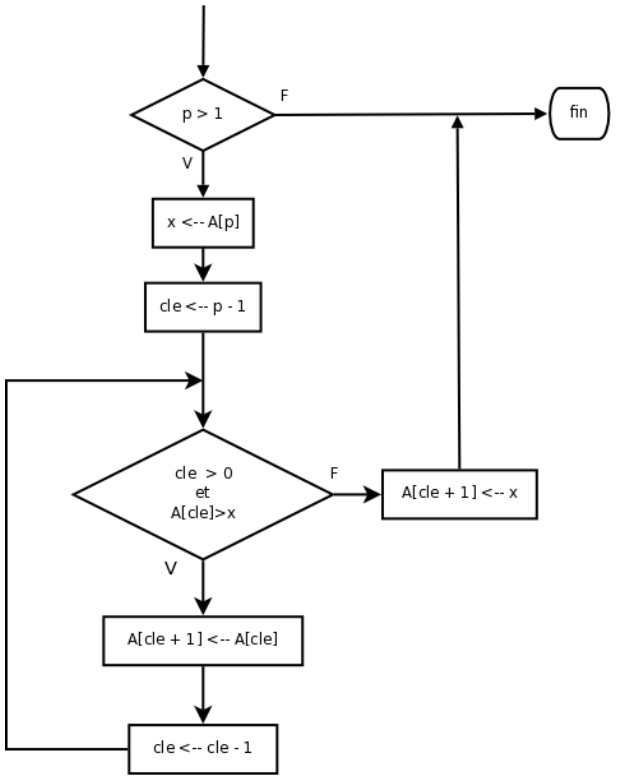

FIG. 2 – Tri par insertion *inserer_valeur*(p)

Le point important dans cette deuxième procédure est de parcourir le tableau en *descendant* de `p` à 1. Il est à noter que le nom `cle` est local à `inserer_valeur()`. Vous devez rédiger ces procédures en syntaxe Maple.

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

2. Implémentation récursive du tri par fusion.

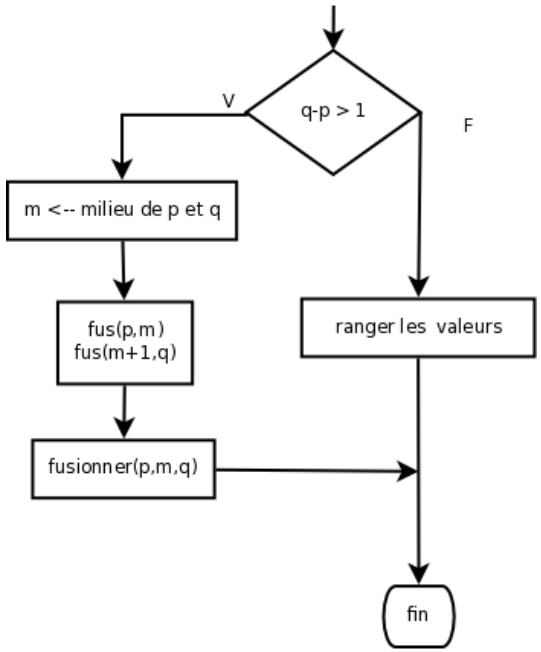

FIG. 3 – Tri par fusion : procédure principale récursive $fus(p, q)$

Le principe du tri *par fusion* repose sur le fait qu'il est assez facile de fusionner deux listes déjà triées en une liste triée.
Imaginons deux piles de copies de maths. Chaque pile est triée par ordre alphabétique avec le nom vers le haut, la copie du dessus (la seule visible) est la première par ordre alphabétique. On choisit alors entre les deux copies visibles celle dont le nom est avant l'autre dans l'ordre alphabétique et on forme ainsi une troisième pile (en retournant la copie choisie). En continuant le processus, les deux piles diminuent tandis que le tas de copie retournées augmente. Lorsque les deux piles sont vides, le tas retourné est complètement trié avec le début de l'alphabet au dessus. Evidemment, un des deux tas sera vide avant l'autre. On peut alors retourner d'un coup le tas restant puisqu'il est déjà trié.
Ce processus est assez commode à mettre en œuvre pratiquement, il permet aussi à plusieurs personnes de travailler simultanément (au début) en triant des paquets arbitraires avant la fusion finale. Cette manière de procéder à un aspect récursif.

Ce mode de tri sera implémenté avec deux procédures utilisant les variables globales `A` et `n` pour désigner le tableau et sa taille.

La procédure principale (figure 3) est appelée par `fus(p,q)`. Elle doit ordonner les valeurs de `A` entre les indices `p` et `q` passés en paramètres (avec `p` inférieur ou égal à `q`).
Lorsque les paramètres sont proches, la procédure range directement le tableau. Lorsque `p` et `q` sont éloignés, la procédure calcule un `m` entre les deux, s'appelle elle même deux fois (récursivité) entre `p` et `m` puis entre `m+1` et `q` et appelle ensuite la procédure `fusionner(p,m,q)` qui fusionne les deux parties du tableau.

La procédure `fusionner(p,m,q)` doit ordonner les valeurs du tableau `A` entre les indices `p` et `q` comme expliqué (figure 4 au début avec l'exemple des copies. Le tableau étant partiellement trié (de p à m et de $m+1$ à q). On remarque que le diagramme utilise une variable locale `B` destinée à recevoir transitoirement les valeurs ordonnées. Il utilise aussi trois clés, une pour chaque "petite" pile et une pour le tableau transitoire `B`.
Vous devez rédiger ces procédures en syntaxe Maple.

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

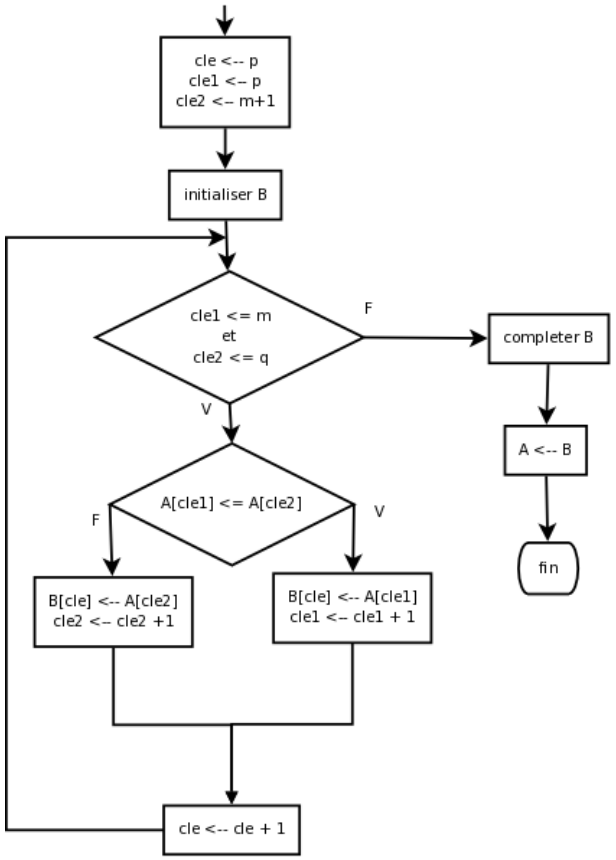

FIG. 4 – Fusion des deux parties du tableau : $fusionner(p, m, q)$

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

Corrigé

1. Tri par insertion.

```
 ins:= proc()
  global n, A;
  local cle;
  for cle from 2 to n do
    inserer_valeur(cle);
  od;
  return;
end proc:

inserer_valeur := proc(p)
  global n,A;
  local cle,x;
  if p=1 then
    return
  else
    cle := p-1;
    x:= A[p];
    while (cle >0) and (A[cle]>x) do
      A[cle + 1] := A[cle];
      cle := cle -1;
    od;
    A[cle + 1] := x;
    return;
  fi;
end proc:

n:= 5;
A:= array(1..n,[2,5,1,3,5]);
ins();
eval(A);
```

2. Tri par fusion

```
 #tri par fusion
fus:=proc(p,q)
  global A,n;
  local truc,m;
  if q-p>1 then
    m := floor((p+q)/2);
    fus(p,m);
    fus(m+1,q);
    fusionner(p,m,q);
  else
    if A[p]>A[q] then
      truc := A[p];
      A[p]:= A[q];
      A[q] := truc;
    fi;
  fi;
  return ;
end proc:

fusionner:=proc(p,m,q)
  global A,n;
  local cle, cle1, cle2, B, truc;
  cle  := p;
  cle1 := p;
  cle2 := m+1;
  B := array(p..q);

  while cle1 <=m and cle2 <=q do
    if A[cle1] <= A[cle2] then
      B[cle] := A[cle1];
      cle1 := cle1 + 1;
    else
      B[cle]:= A[cle2];
      cle2 := cle2 +1;
    fi;
    cle := cle +1;
  od;

  #compléter B
  if cle1>m then
    truc := cle2;
  else
```

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

```
    truc := cle1;
  fi;
  while cle <= q do
    B[cle] := A[truc];
    cle := cle +1;
    truc := truc +1;
  od;

  # A <-- B
  for cle from p to q do
    A[cle]:=B[cle]
  od;
  return ;
end proc:

n:= 5;
A:= array(1..n,[2,1,1,3,1]);
fus(1,5);
eval(A);
```

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/