Énoncé

Tri "rapide".
On commence par partitionner le tableau en deux parties en permutant certains éléments de telle sorte que toutes les valeurs de la partie gauche du tableau soient inférieures aux valeurs de la partie droite.
Pour obtenir un tel partionnement, on utilise une valeur x du tableau. À gauche se trouvent des valeurs toutes inférieures ou égales à x, à droite toutes les valeurs doivent être plus grandes que x (mais pas forcément strictement plus grandes).
Ensuite la procédure s'appelle récursivement sur chaque partie du tableau.

Former les diagrammes conventionnels puis rédiger des procédures `partitionner(p,q)` et `tri_rap(p,q)`
implémentant ce processus.
Les procédures utilisent la variable globale `A` comme nom du tableau.
L'appel de la procédure `partitionner(p,q,x)` affecte `A[p]` à une variable locale `x` et renvoie un entier i entre `p` et `q-1` après avoir permuté des valeurs de `A` pour que

$$q \leq k \leq i \Rightarrow A[k] \leq x$$
$$i+1 \leq k \leq q \Rightarrow x \leq A[k]$$

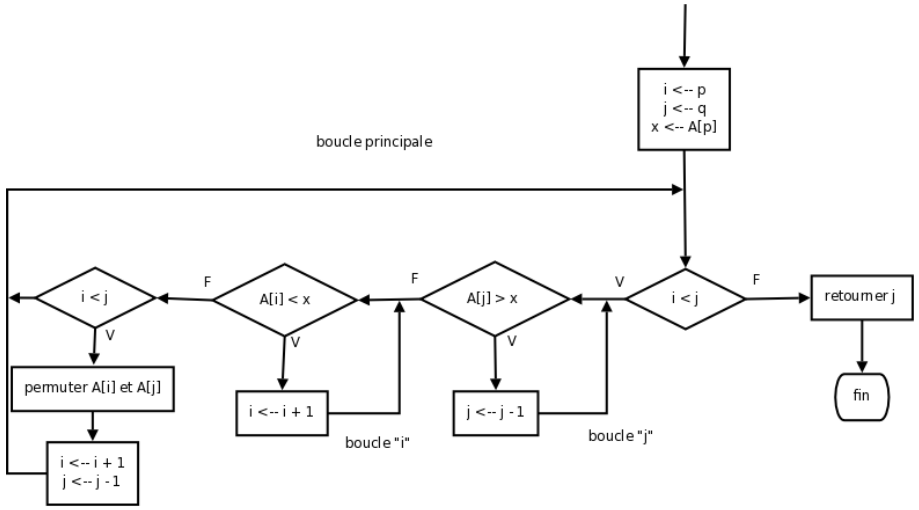

FIG. 1 – Partitionner

Corrigé

La procédure récursive principale `tri_rap` est simple.

```
tri_rap := proc(p,q)
  local i;
  global A;
  if q-p>1 then
    i:= partitionner(p,q);
    tri_rap(p,i);
    tri_rap(i+1,q);
  else
    if A[p] > A[q] then
      truc := A[p];
      A[p] := A[q];
      A[q] := truc;
    fi;
  fi;
end proc:
```

La procédure `partitionner` est plus compliquée. Voir son diagramme en figure 1. Il est à noter que, lors du premier passage dans la boucle "principale", comme $x = A[p]$, on sort de la boucle "j" avec une valeur $> p$ pour `j` et de la boucle "i" avec la valeur p pour `i`. On passe donc obligatoirement au moins une fois par l'échange des valeurs, l'incrément de `i`, le décrément de `j`.

```
 partitionner:=proc(p,q)
  global A;
  local i,j,truc,x;
  i:=p; j:=q; x := A[p];
  #début boucle "principale"
  while j-i >= 0 do
    #début boucle "j"
    while A[j]>x do
      j:=j-1;
    od;
    #fin boucle "j"
    #début boucle "i"
    while A[i]< x do
      i:= i+1;
    od;
    #fin boucle "i"
    if i<j then
      truc:= A[i];
      A[i]:= A[j];
      A[j]:=truc;
      i := i+1;
      j := j-1;
    fi;
  od;
  #fin boucle "principale"
  print(p,q,x,j,A);#pour examen, à supprimer ensuite
  return(j);
end proc:

n:=7;
A:=array(1..n,[6,4,3,3,1,5,2]);
#partitionner(1,n,4);
tri_rap(1,n);
print(A);
```

Lorsque le tableau à trier est `[2,3,1]`, le premier passage dans la boucle principale permute les deux extrèmes. Le tableau devient `[1,3,2]` avec `i` et `j` désignant 2. On passe donc une deuxième fois dans la boucle principale, le `j` est décrémenté à 1 mais le `i` reste à 2 et on sort en renvoyant 1 ce qui est permis.

Vous pouvez télécharger la feuille de calcul.