2-approximation algorithm of Ailon to compute un consensus of "ensTri", where each ranking in "ensTri" is a ranking of [n]. The return consensus is always a full-ranking

```
repeatchoice:=proc(ensTri,n)
   local i,k,pi,sigma,sig,S,ties,m,elementRestant,de,rep,egaux;
   pi:=[{seq(k,k=1..n)}];
   sigma:=randomTri(n);
   sig:=convert(sigma,list);
   S:={};
   m:=nops(ensTri);
   elementRestant:={seq(k,k=1..m)};
   ties:=tie_matrix(ensTri,n);
   egaux:=elementsEgaux(pi,ties,n,m);
   while evalb(egaux <> {}) and evalb(elementRestant <> {}) do
        de:=rand(1..nops(elementRestant));
        i:=elementRestant[de()];
        pi:=etoile(ensTri[i],pi,n);
        egaux:=elementsEgaux(pi,ties,n,m);
        S:=S union {i};
        elementRestant:={seq(k,k=1..m)} \minus S;
   od;
rep:=etoile(sig,pi,n);
rep;
end:
```

2-approximation algorithm of Ailon to compute un consensus of "ensTri", where each ranking in "ensTri" is a ranking of [n], without the use of a full ranking sigma to break ties at the end. The return consensus is a ranking with ties:

```
repeatchoiceSansSigma:=proc(ensTri,n)
   local i,k,pi,S,ties,m,elementRestant,de,rep,egaux;
   pi:=[{seq(k,k=1..n)}];
   S:={};
   m:=nops(ensTri);
   elementRestant:={seq(k,k=1..m)};
   ties:=tie_matrix(ensTri,n);
   egaux:=elementsEgaux(pi,ties,n,m);
   while evalb(egaux <> {}) and evalb(elementRestant <> {}) do
        de:=rand(1..nops(elementRestant));
        i:=elementRestant[de()];
        pi:=etoile(ensTri[i],pi,n);
        egaux:=elementsEgaux(pi,ties,n,m);
        S:=S union {i};
        elementRestant:={seq(k,k=1..m)} \minus S;
   od;
```

```
rep:=pi;
rep;
end:
```

Generate a random full ranking (ranking without ties) of [n]

```
randomTri:=proc(n)
   local i,j,k,perm,r,temp,roll;
     perm:=Array([seq({k},k=1..n)]);
     roll:=rand(1..n);
     for j from 1 to n do
          r:=roll();
          temp:=perm[j];
          perm[j]:=perm[r];
          perm[r]:=temp;
   od;
perm;
end:
```

Generate a matrix containing in position [i,j] the number of time i=j in the rankings of set "ens"

```
tie_matrix:=proc(ens,n)
  local cout,i,j,k,m,pos;
  cout:=Array(1..n,1..n);
  for m from 1 to nops(ens) do
       pos:=position(ens[m],n);
       for i from 1 to n-1 do
          for j from i+1 to n do
                if pos[i] = pos[j] then
                      cout[j,i]:=cout[j,i]+1;
                      cout[i,j]:=cout[i,j]+1;
                fi;
          od;
       od;
od;
cout;
end:
```

**Compute all pairs of elements [u,v] in ranking "TriPi" such that u=v in TriPi both u is not congruent to v for the set of rankings "ens". Here "MatriceTies=tie_matrix(ens,n)"
and m is the cardinality of ens.**

```
elementsEgaux:=proc(TriPi,MatriceTies,n,m)
   local i,j,rep,pos;
   rep:={};
   pos:=position(TriPi,n);
   for i from 1 to n-1 do
         for j from i+1 to n do
               if pos[i] = pos[j] then
                    if MatriceTies[i][j] <> m then
                          rep:=rep union {[i,j]};
                    fi;
               fi;
         od;
od;
rep;
end:
```

Compute the * operation defined in Ailon article

```
etoile:=proc(piPrime,pi,n)
   local i,j,k,le,rep,posPrime,posPi,posI,bucket,temp;
   rep:=pi;
   posPrime:=position(piPrime,n);
   posPi:=position(pi,n);
   for i from 1 to n-1 do
            le:=nops(rep);
            posI:=position(rep,n)[i];
            bucket:=rep[posI];
            temp:=etoileI(bucket,posPrime,posPi,i);
            rep:=[op(rep[1..posI-1]),op(temp),op(rep[posI+1..le])];
   od;
rep;
end:
```

```
etoileI:=proc(bucketi,posPrime,posPi,i)
     local j,gauchei,droitei,egali,rep;
     gauchei:={};
     droitei:={};
     egali:={i};
     for j from 1 to nops(bucketi) do
        if i <> bucketi[j] then
           if posPi[i] < posPi[bucketi[j]] then
                    droitei:=droitei union {bucketi[j]};
               elif    posPi[i]=posPi[bucketi[j]] and posPrime[i] < posPrime[bucketi[j]] then
                  droitei:=droitei union {bucketi[j]};
               elif posPi[i] = posPi[bucketi[j]] and posPrime[i] >= posPrime[bucketi[j]] then
                  egali:=egali union {bucketi[j]};
               else
                  gauchei:=gauchei union {bucketi[j]};
        fi;
      fi;
    od;
    if gauchei = {} and droitei = {} then
         rep:=[egali];
      elif gauchei = {} and droitei <> {} then
         rep:=[egali,droitei];
      elif gauchei <> {} and droitei = {} then
         rep:=[gauchei,egali];
      else
         rep:=[gauchei,egali,droitei];
      fi;
rep;
end:
```

Generate an array of length n containing in position i, the position of i in the ranking i.e. the index of the bucket containing i

```
position:=proc(liste,n)
local i,j,pos;
pos:=Array(1..n);
for i from 1 to nops(liste) do
   for j from 1 to nops(liste[i]) do
       pos[liste[i][j]]:=i;
   od:
od;
pos;
end:
```