1. evaluation :=proc(P,v)
   local m,horner,i ;
   m :=taille(P) ;
   horner :=0 ;
   for i from m to 1 by -1 do
       horner :=(horner+P[i])*v
   od ;
   return horner ;
   end ;
2. valuation :=proc(P)
   local m,i ;
   m :=taille(P) ;
   for i from 1 to m do
       if P[i]<>0 then
           return i
       fi ;
   od ;
   return 0 ;
   end ;
3. difference :=proc(P1,P2)
   local m1,m2,m,diff ;
   m1 :=taille(P1) ;
   m2 :=taille(P2) ;
   if m1>m2 then
       m :=m1
   else
       m :=m2
   fi ;
   diff :=allouer(m) ;
       for i from 1 to m do
           diff[i] :=0 ;
           if i<m1+1 then
               diff[i] :=diff[i]+P1[i]
           fi ;
           if i<m2+1 then
               diff[i] :=diff[i]-P2[i]
           fi ;
       od ;
   od ;
   return diff ;
   end ;
4. compareNeg :=proc(P1,P2)
   local diff,k ;
   diff :=difference(P1,P2) ;
   k :=valuation(diff) ;
   if k=0 then
       return 0
   elif diff[k]>0 then

```
        return 1-2*irem(k,2)
    else
        return 2*irem(k,2)-1
    fi;
    end;
```

5.
```
01- tri :=proc(t)
02- local n,ttrie,i,j,tempo;
03- n :=taille(t);
04- ttrie :=allouer(n);
05- for i from 1 to n do
06-         ttrie[i] :=t[i]
07- od;
08- for i from n to 2 by -1 do        On cherche d'abord le plus petit, puis le plus petit de ceux qui restent, etc...
09-         for j from 1 to i-1 do        On décrit le segment ⟦ 1, i ⟧ du tableau de la gauche à la droite
10-             if compareNeg(ttrie[j],ttrie[j+1])<0 then
11-                 tempo :=ttrie[j];
12-                 ttrie[j] :=ttrie[j+1];
13-                 ttrie[j+1] :=tempo
14-             fi;
15-         od;        Le plus petit des éléments n°1..i est "descendu" en i-ème position
16- od;
17- return ttrie;
18- end;
```

Variante pour les lignes 08..16 :

```
08- for i from 1 to n-1 do        On cherche d'abord le plus grand, puis le plus grand de ceux qui restent, etc...
09-         for j from n-1 to i by -1 do        On décrit le segment ⟦ i, n ⟧ du tableau de la droite à la gauche
10-             if compareNeg(ttrie[j],ttrie[j+1])<0 then
11-                 tempo :=ttrie[j];
12-                 ttrie[j] :=ttrie[j+1];
13-                 ttrie[j+1] :=tempo
14-             fi;
15-         od;        Le plus grand des éléments n°i..n est "monté" en i-ème position
16- od;
```

6.
```
comparePos :=proc(P1,P2)
local diff,k;
diff :=difference(P1,P2);
k :=valuation(diff);
if k=0 then
    return 0
elif diff[k]>0 then
    return 1
else
    return -1
fi;
end;
verifierPermute :=proc(perm,t)
```

```
local n,i ;
n :=taille(t) ;
for i from 1 to n-1 do
    if comparePos(t[perm[i]],t[perm[i+1]])<0 then
        return faux
    fi ;
od ;
return vrai ;
end ;
```

7.
```
estEchangeurAux :=proc(perm,d)
local c,b,a ;
for c from d+1 to n do
    for b from c+1 to n do
        for a from b+1 to n do
            if perm[b]>perm[d] and perm[d]>perm[a] and perm[a]>perm[c] then
                return faux
            elif perm[c]>perm[a] and perm[a]>perm[d] and perm[d]>perm[b] then
                return faux
            fi ;
        od ;
    od ;
od ;
return vrai ;
end ;
```

8.
```
estEchangeur :=proc(perm)
local n,d ;
n :=taille(perm) ;
for d from 1 to n-3 do
    if estEchangeurAux(perm,d)=faux then
        return faux
    fi ;
od ;
return vrai ;
end ;
```

9.
```
nombreEchangeurs :=proc(n)
local nbre,j,i ;
nbre :=allouer(n) ;
nbre[1] :=1 ;
for j from 2 to n do
    nbre[j] :=nbre[j-1] ;
    for i from 1 to j-1 do
        nbre[j] :=nbre[j]+nbre[i]*nbre[j-i]
    od ;
od ;
    return nbre[n] ;
end ;
```

10.
```
decaler :=proc(t,v)
local n,u ;
```

```
    n :=taille(t) ;
    u :=allouer(n+1) ;
    u[1] :=v ;
    for i from 2 to n+1 do
        if if t[i-1]<v then
            u[i] :=t[i-1]
        else
            u[i] :=1+t[i-1]
        fi ;
    od ;
    return u ;
    end ;
```

11.
```
    enumererEchangeurs :=proc(n)
    local a,tab,aProv,i,tempo,j,v,u ;
    a :=nombreEchangeurs(n) ;
    tab :=allouer(a) ;
    tab[1] :=allouer(1) ;
    tab[1][1] :=1 ;        tab contient actuellement l'unique échangeur dans S_1 ie l'identité de {1}, suivi de n'importe quoi
    aProv :=1 ;        Provisoirement, seul 1 élément de tab est un échangeur (dans S_1)
    tempo :=allouer(a) ;        Servira d'entrepôt
    for i from 2 to n do
        for j from 1 to aProv do
            tempo[j] :=tab[j]
        od ;        On entrepose dans tempo les aProv échangeurs de S_{i-1}
        k :=0 ;
        for v from 1 to i do        On choisit le premier élément pour une permutation dans S_i
            for j from 1 to aProv do        On choisit un échangeur dans S_{i-1}
                u :=decale(tempo[j],v) ;        On fabrique une permutation dans S_i
                if estEchangeurAux(u,1)=vrai then
                    k :=k+1 ;
                    tab[k] :=u
                fi ;
            od ;
        od ;        Les k premiers éléments de tab sont les échangeurs de S_i
        aProv :=k        En fait, ici, k=nombreEchangeurs(i)
    od ;
    return tab ;
    end ;
```