```
truc :=proc()
....
if ... then
        return ...     { Le cas particulier auquel on se ramène}
else
        ...
        return .... truc(...)...     { On appelle truc appliquée à un cas "plus simple"}
fi;
end;
```

Exemples :

1. Algorithme de Hörner :

$$P(x) = a_0 + a_1x + ... + a_nx^n = a_0 + (a_1 + a_2x + ... + a_nx^{n-1})x$$

On écrit une procedure *truc* qui prend un tableau de nombres complexes a indexé de 0 à $n-1$ (où sont stockés les coefficients d'un polynôme), un nombre complexe x, un entier i tels que $0 \leq i < n$ et qui retourne $\sum_{k=i}^{n-1} a[k]x^{k-i}$; en particulier, *truc*(a,x,0) calcule $P(x)$.

```
truc :=proc(a,x,i)
if i=n-1 then
        return a[i]
else
        return a[i]+truc(a,x,i+1)*x
fi;
end;
```

2. Algorithme d'exponentiation rapide :

$$a^n = (a^2)^{\frac{n}{2}} \text{ si } n \text{ est pair }, \ a^n = a.(a^2)^{\frac{n-1}{2}} \text{ sinon}$$

On écrit une procedure qui prend un entier $n \in \mathbb{N}$ et un nombre complexe a et qui renvoie a^n.

```
truc :=proc(n,a)
local reste,quotient;
if n=0 then
        return 1
else
        quotient :=iquo(n,2);
        reste :=irem(n,2);
        if reste=0 then
                return truc(quotient,a*a)
        else
                return a*truc(quotient,a*a)
        fi
fi;
end;
```

3. Tri-fusion
   Pour trier un tableau t de taille 2^p, indexé de 0 à $2^p - 1$.
   - On le scinde en 2 tableaux t1, t2 de taille 2^{p-1}
   - On trie t1 et t2
   - On "fusionne" les 2 tableaux triés à 2^{p-1} éléments en un tableau trié à 2^p éléments.

```
scinde :=proc(t,p)
local i,t1,t2 ;
t1 :=array(0,2^(p-1)-1) ;
t2 :=array(0,2^(p-1)-1) ;
for i from 0 to 2^(p-1)-1 do
        t1[i] :=t[i] ;
        t2[i] :=t[i+2^(p-1)]
od ;
return [t1,t2] ;
end ;
```

```
fusion :=proc(t1,t2,p)
local t,i1,i2,i ;
t :=array(0,2^p-1) ;
i1 :=0 ;
i2 :=0 ;
while i1<2^(p-1) and i2<2^(p-1) do
        i :=i1+i2 ;
        if t1[i1]>t2[i2] then
                t[i] :=t1[i1] ;
                i1 :=i1+1
        else
                t[i] :=t2[i2] ;
                i2 :=i2+1
        fi
od ;
if i1=2^(p-1) then
        for k from i to 2^p-1 do
                t[k] :=t2[i2+k-i-1]
        od
else
        for k from i to 2^p-1 do
                t[k] :=t1[i1+k-i-1]
        od
fi ;
return t ;
end ;
```

```
tri :=proc(t,p)
local tScinde,t1,t2,t1Trie,t2Trie ;
if p=0 then
        return t
else
        tScinde :=scinde(t,p) ;
        t1 :=tScinde[1] ;
        t2 :=tScinde[2] ;
        t1Trie :=tri(t1,p-1) ;
        t2Trie :=tri(t2,p-1) ;
        return fusion(t1Trie,t2Trie,p)
fi ;
end ;
```