Tri Selection:

```
procedure Selection(n: entier, @ t: tab):
    debut
        pour i de 1 a n-1 faire
            j ← i
            tmp ← t[i]
            tant que j > 1 et tmp < t[j-1] faire
                t[j] ← t[j-1]
                j ← j-1
            fin tant que
            t[j] ← tmp
        Fin pour
    Fin
```

Var	type
i	entier
j	entier
tmp	entier ou réel!

Tri a bulle:

```
procedure bubble(n: entier, @ t: tab):
    Debut
        pour i de 0 à n-2 faire:
            Si t[i] > t[i+1] alors
                tmp ← t[i]
                t[i] ← t[i+1]
                t[i+1] ← tmp
                bubble(n, t)
            Fin Si
        Fin pour
    Fin
```

Var	type
i	entier
tmp	entier ou réel.
bubble	procedure

Tri Shelle:

```
Procedure Shell (n: entier, @ t: tab):
  Debut:
    Pas ← 1
    tant que Pas × 3 + 1 ≤ n faire
        Pas ← Pas × 3 + 1
    Fin tant que.
    Repeter:
    Pour i de Pas a n-1 faire
        tmp ← t[i]
        j ← i
        tant que j ≥ 1 et t[j - Pas] < tmp faire
            t[j] ← t[j - Pas]
            j ← j - Pas
        Fin tant que
        t[j] ← tmp
    Fin pour
    Pas ← Pas div 3
    jusqu'a ce (Pas ≤ 1)

  Fin
```

tmp j j entier
Pas entier

project". Math complex:.

Cartésienne?:

Def funct calc: $(2-2i)^2$

$a^2 + 2 \cdot a \cdot b + b^2$

function calc (reg: String): String:

Begin

// separate?.

13/12/24:

Correction devoir Synthèse n°1:

P.P

Algorithme point-equilibre

Début

    Saisir(l,c)
    fillmat(l,c,m)
    src ← "equilibre.dat"
    remplif(m,src,l,c)

    src₁ ← "mon_Somme.txt"
    remplir 2t(src, src₁, m)

    Fin

TDO:

Nom	type
l,c	entier
src, src₁	chaîne
m	mat

Fonction(m: mat; l: entier, c: entier): boolean:

    Début
        i ← -1
        Repeter
            i ← i+1
            check ← m[l,i] ≠ m[l,c]
        Jusqu'à check = faux ou i = c-1
        Retourner check.
    Fin

TDO:
i: entier
check: bool

Recursivités:

fonction fact (n: entier): entier

    Debut
    Si n=0 alors f←1
    Sinon
        f←1
        pour i de 2 ant (pas-1) faire
            f←f×i
    Fin Si pour
    retourner f

    Fin

fonction rec (n: entier): entier
    Debut
    Si n=0 alors
        retourner 1
    Sinon
        retourner n × rec (n-1)
    Fin Si
    Fin

---

exemple 2:
en calcule La somme de n premier entier positive de 2 manieres;

fonction Som (n: entier): entier
    Debut
    S←0
    pour i de 1 à n faire
        S←S+i
    Fin pour
    Retourner S
    Fin

fonction Som (n: entier): entier
    Debut
    Si n=0 alors
        retourner 0
    Sinon
        retourner n + Som (n-1)
    Fin Si
    Fin

Act = Ecrire une fonction Recursive permettant de effectuer
la multiplication de 2 entiers ≥ 0 notter A et B,
en utilisant uniquement L'addition entiere.
en effet A×B = A + A ..... } B×

```
Fonction Rec(a: entier, b: entier): entier
    Début
        S ← 1
        Pour i de 1 a b faire
            S ← S * a
        Fin pour
        retourner S
    Fin
```

```
Fonction Rec(a: entier, b: entier): entier
    Début
        Si b = 0 alors
            Retourner 1
        Sinon
            retourner a * Rec(a, (b-1))
        Fin Si
    Fin
```

Act 2: Soit une chaine de charon; écrire une f qui nous renvoie l'inverse de cette chaine

```
Fonction Rev(Str: chaine): chaine
    Début
        Str ← Str
        Pour i de 0 a n div 2 faire
            tmp ← Str[i]
            Str[i] ← Str[m-i-1]
            Str[m-i-1] ← tmp
        Fin pour
```

Retourner Soln

Fin

Fonction Rec (ch : chaine, n = 0   ) : entier) : chaine

    Debut

        Si ch = " " alors

            retourner ""

        Sinon

            retourner (ch[0] + Rec (Sous chaine(ch, 1, Long (ch))

        Fin Si

    Fin

palyindrone Algo :

Iterative :

Fonction est polyindrone (ch : chaine) : booleen
    Debut
        i ← 0
        check ← vrai
        tant que i < Long (ch) div 2 et check faire
            check ← ch [i] = ch [Long (ch)-1-i-1]

            i ← i + 1
        Fin tant que
        retourner check

    Fin

ou :

    Fonction polyin (ch : chaine) booleen
        Debut
            pour i de 0 a Long (ch) - 1 div 2 faire
                Si ch [i] ≠ ch [Long(ch) -1 - i] alors
                    retourner faux

```
                    Fin Si
            Fin pour
            retourner Vrai
            Fin
```

Recursive

```
Fonction recString (myString°: chaine, l: entier, b: entier, e: entier): booleen
    Debut
        Si b = e ou b+1 = e alors
            retourner Vrai
        Sinon
            Si myString [b] = myString [e] alors
                retourner Vrai et recString (myString, l, b+1, e-1)
            Sinon
                retourner faux
            Fin Si
        Fin Si
    Fin
```

Exercices:

Problem 1:

Iterative:

```
Fonction prod (p: entier, q: entier): entier
    Debut
        S ← 0
        pour i de 1 a q faire
            S ← S + P
        Fin pour
```

retourner S

Fin

Rec :

Fonction ProdRec (p : entier, q : entier) : entier

   Debut

      Si q = 0 alors

         retourner 0

      Sinon

         retourner   + ProdReq $(P, q-1)$

      Fin Si

   Fin

Problème 2 :

  Iterative :

   Fonction Sum (m : entier) : entier

     Debut

       S $\leftarrow$ 0

       pour i de 1 à m faire

         S $\leftarrow$ S + i

       Fin pour

       retourner S

     Fin

Fonction sum (m : entier) : entier
    Debut
        Si m = 0 dors
            retournes 0

        Sinon
            retourner m + Sum (m-1)

        Fin Si

        Fin

problème 3

Méthode Euclide :

Fonction pgcd (a : entier, b : entier) : entier
    Debut
        tant que b ≠ 0 faire
            n ← a mod b
            a ← b
            b ← n
        Fin tant que
        retourner a
        Fin

_____

Recursive

Fonction pgcd (a : entier, b : entier) : entier
    Debut

- Si b = 0 alors
    retourner a

    Sinon
        retourner (b, a mod b)

    Fin Si

Fin

Methode dle diff

Iterative :

Fonction pgcd (a : entier, b : entier) : entier

    Debut
        tant que a ≠ b
            Si a > b alors
                a ← a - b

            Sinon
                b ← b - a

            Fin Si
        Fin tant que

        retourner a
    Fin

Recursive :

Fonction pGcd (a: entier, b: entier): entier
    Debut
        Si a == b alors
            retourner a
        Sinon
            Si a > b alors
                retourner pGcd (a - b, b)
            Sinon
                retourner pgcd (b, b - a)
            Fin Si
        Fin Si
    Fin

Problème 4:

Itérative:

Fonction isprime (m: entier): bool
    Debut
        toReturn ← fausse
        i ← 0
        Repeter
            toReturn ← m mod i = 0
            i ← i + 1
        Jusque i = m div 2 out Return

retourner mon tableau

Fin

Recursive :

Fonction is prime (m : entier, i : entier) : bool
  Debut
      { initial value of i is 1 }

    Si m div 2 = i olos
            retourner Vrai

      Sinon
          retourner vrai et m mod i ≠ 0 et isprime (m, i+1)

      Fin Si

      Fin

problemes :

Iterative :
procedure Rev (myString : chaine) :
    Debut
        pour i de 0 a (Long (myString) - 1) div 2 faire
          temp = myString [i]
          myString [i] ← myString [Long (myString) - 1 - i]
          myString [Long (myString) - 1 - i] ← temp
            Fin pour

        Fin

Recursive
procedure Rev (myString : chaine) :
    Debut
        Si Long (myString) > 1 alors
            temp ← myString[i]

            myString[i] ← myString[Long(myString) - 1]

            myString[Long(myString) - 1] ← temp

            Rev (sous-chaine(myString, 1, Long(myString)-2))

        Fin Si
    Fin

problème 6 :

Iterative

Fonction pal (ch : chaine) : bool
    Debut
        i ← 0
        check ← faux

        Repeter
          check ← ch[i] = ch[Long(ch)-1-i]
          i ← i + 1
          jusqu'a (Long(ch)-1) div 2 = i ou check

        retourner check

        Fin

Récursive :

Fonction pol (ch : chaine) : bool

  Début
      Si Len (ch) ≤ 1 alors
          retourner Vrai

      Sinon
          retourner Vrai et ch[0] = ch[long(ch)-1]
                          et pol (sous-chaine (ch, 1, long(ch)-2))

      Fin Si

    Fin

problème 7 :

Itérative :

  Procedure Rev (@ t : tab) :

    Début
    pour i de 0 à Long (t)-1 div 2 faire
        temp ← t[i]
        t[i] ← t[long(t)-1 -i]
        t[Long(t)-1-i] ← temp

      Fin pour
    Fin

Recursive:

procedure Rev(@t:tab):

   Debut

      si Long (t) > 1 alos

         temp ← t[0]

         t[0] ← t[Long (t)-1]

         t[Long (t)-1] ← temp

         Rev(t[1, Long(t)-2])

      Fin si

   Fin

Iterative

procedure Rev(@t:tab):

   Debut

      P ← 0

      n ← Long (t)-1

      Tant que Long div 2 ≠ P faire
         temp ← t[P]
         t[P] ← t[n]
         t[n] ← temp
         P ← P+1
         n ← n-1

Fin fonction graph

Fin

Récursive

procédure Rev (@ t : tab, p : entier, m : entier)
   Début
     Si long(t) div 2 < p alors
       temp ← t[p]

       t[p] ← t[m]

       t[m] ← temp

       Rev (t, p+1, m-1)

     Fin Si

   Fin

problème 8
  Rec :
  Fonction rev (chr : chaine ; m : entier) : chaine
    Début
     Si m ≥ 0 alors
      retourner
       chr[m] + Rev (chr, m-1)

     Fin Si

   Fin

  Fin fonction

Si cheatt
    retourner mid

Snior
    retourner - 1

F_i S_i

F_i

P

Iterative :

Fonction Rev (ch : chaine) : chaine
    Debut
        ch2 ← ""
        pour i de long(ch) - 1 allant à - o fois
            ch2 ← ch 2 + ch [i]

        Fin pour

        retourne ch2

    Fin

problème 9 :

Iterative :

    Fonction binary (t : tab, b : entier, c : entier, ele)
        Debut
            check ← faux

            tant que check = faux et b ≤ c fait
                mid ← (b + c) div 2
                si t [mid]) = ele
                    check ← vrai

                Sinon
                    si ele > t[mid]
                        b ← b + 1

                    Sinon
                        e ← mid - 1

                    Fin Si

Recursive :

Fonction binary (t : tab, ele : entiers) : entiers
    Debut
        Si t [Long(t) div 2] = ele alors
            retourne Long (t) div 2

        Sinon Si t [Long (t) div 2] > ele alors
            retourner binary (t [1, (Long(t) div 2) - 1], ele)

        Sinon
            retourner Long (t) div 2 + binary (t [Long (t) div 2) +1,
                                                Long(t) - 1]

        Fin Si

    Fin

Lineaire :

Recursive :

Fonction lineaire (t : tab, n : entiers, x, i : entiers) : entier
    Si n = i alors
        retourner -1
    Sinon
        Si t [i] = x alors
            retourner i
        sinon

retourner (t, m, e, i+1)

Fin Si

Fin

Rec 2': with counter:

Fonction Rec(t: liste, i, p, m): entier
    Début
        Si i ≥ m alors
            retourner 0

        sinon
            Si t {i} = p alors
                retourner 1 + Rec(t, i+1, p, m)

            Sinon
                retourner Rec(t, i+1, p, m)

            Fin Si

        Fin Si
    Fin

Fonction binaire (t : tab, m, p : entiers)
    début
        d ← 0

        f ← m - 1

        p ← - 1

        Tant que d ≤ f et p = - 1 faire
            m ← (d + f) div 2
            s ← T{m} ≥ p alors
                p ← m

            sinon
                si p < T{i} alors
                    f ← m - 1
                sinon
                    d ← m + 1

                fin si

            fin si faire
            retourner m
        Fin

Recursive:

Fonction Rec (t : tab, d, f, b : entiers) : entiers
  Début
    Si d > f alors
      retournes -1

    Sinon
      m ← (d + f) div 2
      Si t[m] = b
        alors retournes m

      Sinon si t[m] > b alors
        retournes Rec (t, d, m - 1, b)

      Sinon
        retournes Rec (t, d, m + 1, b)

      Fin Si

  Fin

```
Algo P.P
   Debut
     Size (m)

     superPrime ( m, "C:\file.txt" )

   Fin


Procedure Size (@ in: entier):
       Debut
           Repeter
               lie (m)

           Jusqua  40000 < m < 100 000

           Fin

Procedure SuperPremier(m: entier, path: chaine)
         Debut
```

Procedure finb (en : entries) :

lie (m)

Si mom 40 000 (m < 100 000 dog

full (m)

Fin

$\{5, 0, 3, 9, 0\}$

tant que o_ > 1 forio

tet (0_)

a c_a div no

5 div 2

1   2 | 3   4   5

$\boxed{\dfrac{5}{1}} = 5$ True $\boxed{\dfrac{5}{5}} = 1$

$\dfrac{5}{2} \quad n = 1$

$10 \text{ div } 2 = 5$

$\boxed{1} \quad 2 \quad 3 \quad 4 \quad 5 \quad \boxed{6 \quad 7 \quad 8 \quad 9 \quad 10}$

$\dfrac{10}{2} = ? \quad n = 0$ \qquad $\underline{10 \neq \text{pime}}$

Fonction (m : entier) : bool
Début
   pour i de 2 à m div 2 faire
      si m mod i = 0 alors
         retourner faux
      FinSi
   FinPour
   retourner vrai

fonction suite (m:entier, P:entier, Up:réel) : réel :

booleen

Si m ≥ 1 alors

a ← UP ($power(-2, m)$) / (- ($2 \times UP$))

Si p = m alors

retourner a

fin Si

retourner (Suite (m, P+1, a))

Fin Si

retourner 0

Fin

fonction power (a, b : entier, i : entier) : entier

Debut

Si i ≤ b alors

retourner a × power (a, b, i+1)

fin Si

retourner a

Fin

$$u_0 = \text{"0"}$$

$$u_1 = \text{"01"}$$

$$u_2$$

$$O \rightarrow O1$$

$$u_2 = \text{"01, 10"}$$

$$1 \rightarrow 10$$

$$u_3 = \text{"011, 010 01"}$$

fonction binary (m : entier, i : entier, res : chaîne) : Debut

  Debut

    Si m > 0

      Si i < Long (res) alors

        Si res[i] = "0" alors

          res ← Sous_chaîne(res, 0, i+1) + "1" + Sous_chaîne(res, i+1, Long(res))

        Sinon

          res ← Sous_chaîne(res, 0, i+1) + "0" + Sous_chaîne(res, i+1, Long(res))

        fin si

        retourner binary (m : entier, i+1, res)

      Sinon

        retourner binary (m : entier, 0, res)

      fin si

    Sinon

      retourner Res.

  Fin si

Fin

Défi 2:

ordre de récurrence: 2.
$u_1 = 1$
$u_2 = 2$
$u_n = u_{n-1} + u_{n-2}$

---

fonction Feb (n: entier)