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 à 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 Shell :

```
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 que (pas < 1)
Fin
```

projet : Math complexe.

Correction :

Def fonction calc : (2-2i)²

o2+2+a+b+b²

fonction calc (reg:String):String;

begin

//separateur;

13/12/24 :

Correction devoir Synthèse n°1 :

P.P

Algorithme point_equilibre ;

Debut

Saisir(l, c)

fillmat(l, c, m)

src ← "equilibre.dat"

remplir(m, src, l, c)

src1 ← "mon_Sommet.txt"

remplir(src, src1, m)

Fin

TDO :

Var	Type
l, c	entier
src, src1	chaine
m	mat

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

Debut

i ← -1

Repeter

i ← i + 1

check ← somme[l, i] ≠ m[i][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
    pour i de 2 a n (pas -1) faire
      f ← f x i
    fin pour
  fin si
  retourner f
Fin
```

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

Exemple 2 :
on calcule la somme de n premier entier positive de 2 manieres :

```
fonction Som (n: entier) : entier
Debut
  S ← 0
  pour i de 0 a 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 entier 0 noter A et B, en utilisant uniquement l'addition entière.

en effet A x B = A + A + ... + A (B fois)

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

```
fonction Rec(a:entier, b:entier): entier
Debut
  si b:=0 alors
    Retourner 0
  sinon
    retourner a+Rec(a, (b-1))
  Fin si
Fin
```

Act 2 : Soit une chaine de charactères S qui inverse l'inverse de cette chaine

```
fonction Rev(str:chaine): chaine
Debut
  str1 ← str
  Pour i de 0 a n div 2 faire
    tmp ← str[i]
    str[i] ← str[n-i-1]
    str[n-i-1] ← tmp
  Fin pour
```

Retourner ch

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

polyjndrone Algo:

Iterative:

Fonction est polyjndrone (ch: chaine): boolean

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 polyjn (ch : chaine) boolean

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:

pblm 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 p + ProdRec(p, q-1)

Fin Si

Fin

Problème 2:

Itérative:

Fonction sum(n: entier): entier

Debut

S ← 0

pour i de 1 à n faire

S ← S + i

Fin pour

retourner S

Fin

```
Fonction sum (n : entier) : entier
  Debut
    Si n = 0 alors
      retourner 0

    Sinon
        retourner n + sum(n-1)

    Fin Si

  Fin
```

pbl 3

Methode Euclide :

```
Fonction pgcd(a: entier, b: entier) : entier
    Debut
      tant que b ≠ 0 faire
        r ← a mod b
        a ← b
        b ← r
      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 des diff

Iterative :

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

    Debut
        tantque a ≠ b
            Si a > b alors

                a ← a - b


            sinon
                b ← b - a

            Fin si
        Fin tantque

        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)

         Fsi
      Fsi
    Fin
```

---

problème 4 :

Iterative :

```
Fonction isprime (n: entier): bool
  Debut
    toReturn ← faux
    i ← 0
    Repeter
       toReturn ← n mod i = 0
       i ← i + 1
    jusqu'a  i = n div 2 ou toReturn
```

retourner non toReturn

Fin

Recursive :

Fonction isprime (n : entier, i : entier) : bool

Debut

{initial value of i is 1,}

si n div 2 = i alors

retourner Vrai

sinon

retourner Vrai et n mod i ≠ 0 et isprime (n, i+1)

Fin Si

Fin

problème 5 :

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:

Itérative

```
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
```

Recursive :

```
Fonction pol ( ch : chaine ) : bool
  Debut
    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 :

Iterative :

```
procedure Rev (@t : tab) :
  Debut
  pour i de 0 a 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 alors
            temp ← t[0]
            t[0] ← t[Long(t)-1]
            t[Long(t)-1] ← temp
            Rev(t[1, Long(t)-2])
        Fin si
    Fin
```

Itérative

```
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
Fin
```

Recursive

```
procedure Rev (@t : tab, p : entier, n : entier)
  Debut
    si long(t) div 2 < p alors
      temp ← t[p]
      t[p] ← t[n]
      t[n] ← temp
      Rev (t, p+1, n-1)
    Fin si
  Fin
```

problème 8

```
Rec :
Fonction rev (ch : chaine, n : entier) : chaine
  Debut
    si n ≥ 0 alors
      retourner
        ch[n] + rev (ch, n-1)
    Fin si
  Fin
Fin Fonction
```

Si check
retourner mid

sinon
retourner - 1

Fin Si

Fin

P

Iterative :

```
Fonction Rev (ch : chaine) : chaine
Debut
    ch2 ← ""
    pour i de long(ch) - 1 downto 0 faire
        ch2 ← ch2 + ch[i]
    Fin pour
    retourner ch2
Fin
```

Problème 9 :

Iterative :

```
Fonction binary (t : tab, b : entier, e : entier, ele)
Debut
    check ← faux
    tantque check = faux et b ≤ e faire
        mid ← (b + e) 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 : entier ) : entier
  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
```