Annexes

Quelques exercices d'échauffements

- Algorithme 1
- Algorithme 2

Préparer le terrain

- Algorithme 1
- Algorithme 2
- Algorithme 3
- Algorithme 4
- Algorithme 7
- Algorithme 8
- Algorithme 9

Appliquer

- Algorithme 1
- Algorithme 3

Fonctions avancées

- Algorithme 1
- Programme 2 – Algorithme des fonctions utilisées + Algorithme avec fonctions

Durant tous les algorithmes, pour insérer quelques choses dans une liste on a utilisé append. Cette fonction n'est pas utilisable en algorithme mais il n'existe pas d'autre manière de faire ceci.

QUELQUES EXERCICES D'ECHAUFFEMENT

Programme 1

Objectif : Écrire un programme qui permet de colorer les ***N*** premiers nombres entiers en deux couleurs, telle que les nombres pairs sont affichés en rouge, et les nombres impairs sont affichés en bleu.

ALGO : Coloration nombre pair et impair

```
Variables locales :   N, entier, nombre de nombre à colorier
                      i, entier, nombre actuel
Début
    Afficher « Choisir le nombre de nombres à colorier : »
    N ← saisir

    Tant que (N≤0) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Choisir le nombre de nombres à colorier : »
        N ← saisir
    Fin tant que

    Pour i allant de 1 à N
        Si (i%2=0) //si i est pair
            Afficher (i) + « est rouge »
        Sinon : //si i est impair
            Afficher (i) + « est bleu »
        Fin si
    Fin pour
Fin
```

Programme 2

Objectif : Écrire un programme qui permet de colorer les ***N*** premiers nombres entiers en ***n*** couleurs choisies aléatoirement (les ***n*** couleurs doivent être présentes).

ALGO : Coloration nombre pair et impair

```
Données :               color,1D de 6 entiers, contenant les 6 couleurs
                        c, 1D de n entiers, contenant les n couleurs aléatoires
                        w,1D de N entiers, associant chaque N à une couleur n
Variables locales :     N, entier, nombre de nombre à colorier
                        n, entier, nombre de couleur
                        W, chaine de caractère, couleur blanche
                        R, chaine de caractère, couleur rouge
                        G, chaine de caractère, couleur verte
                        O, chaine de caractère, couleur orange
                        B, chaine de caractère, couleur bleu
                        P, chaine de caractère, couleur violet
                        i, entier, compteur 1
                        cpt, entier, compteur 2
                        j, entier, compteur 3
                        a, entier, conserver la valeur aléatoire de color
                        b, entier, conserver la valeur aléatoire de w
Début
    Afficher « Choisir le nombre de nombres à colorier : »
    N ← saisir
    Tant que (N≤0) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Choisir le nombre de nombres à colorier (>0) : »
        N ← saisir
    Fin tant que

    Afficher « Choisir un nombre de couleur (entre 1 et 6) : »
    n ← saisir
    Tant que (n<1) or (n>6) or (n>N) //saisie sécurisé n entre 1 et 6 et n<N
        Afficher « Choisir le nombre de nombres à colorier (<N) : »
        n ← saisir
    Fin tant que

    //Initialisation de la liste color
    color = ['W', 'R', 'G', 'O', 'B', 'P']

    //Création d'une liste aléatoire c avec n couleurs différentes
    Pour i allant de 1 à n
        a ← choix aléatoire d'une valeur dans (color)
        Tant que (a est dans c)
            a ← choix aléatoire d'une valeur dans (color)
        Fin tant que
        c.insert (i - 1, a)
    Fin Pour

    //Insertion de N valeur (couleur) dans la liste w parmi c
    w = c
    Pour cpt allant de (n+1, N)
        b ← choix aléatoire d'une valeur dans (w)
        w.insert (cpt, b)
    Fin pour
    Mélange de la liste de manière aléatoire
    //Attribution de chaque valeur de la liste w à chaque Nombre
    Pour j allant de (1, N + 1)
        Afficher (j) + « est de la couleur » + (w[j-1])
    Fin pour
Fin
```

PREPARER LE TERRAIN

Programme 1

Objectif : Écrire un programme qui, à partir de trois entiers et leurs trois couleurs respectives, affiche VRAI si le triplet est monochromatique, FAUX sinon

ALGO : Triplet monochromatique ou non

Données : color, 1D de 6 entiers, contenant les 6 couleurs
Nombre, 1D de 9 entiers, contenant les chiffres de 1 à 9

Variables locales : n1, entier premier nombre
n2, entier, deuxième nombre
n3, entier, troisième nombre
c1, chaîne de caractère, couleur du premier nombre
c2, chaîne de caractère ,couleur du deuxième nombre
c3, chaîne de caractère, couleur du premier nombre
W, chaîne de caractère, couleur noire
R, chaîne de caractère, couleur rouge
G, chaîne de caractère, couleur verte
O, chaîne de caractère, couleur orange
B, chaîne de caractère, couleur bleue
P, chaîne de caractère, couleur violette

```
Début
    // Initialisation d'une liste contenant les 6 couleurs
    color = ['W', 'R', 'G', 'O', 'B', 'P']

    // Initialisation de la liste comprenant les chiffres de 1 à 9
    nombre = [1,2,3,4,5,6,7,8,9]

    // Affectation aléatoires dans les valeurs
    c1 ← choix aléatoire d'une variable dans (color)
    c2 ← choix aléatoire d'une variable dans (color)
    c3 ← choix aléatoire d'une variable dans (color)
    n1 ← choix aléatoire d'une variable dans (nombre)
    n2 ← choix aléatoire d'une variable dans (nombre)
    // n3 est la somme de n1 et n2
    n3 ← n1+n2

    // détermination de la monochromie du triplet
    Si (c1=c2 et c1=c3)
        Afficher « triplet : (» + (c1, n1) + « , » + (c2, n2) + « , » + (c3, n3) + « ) VRAI »
    Sinon
        Afficher « triplet : (» + (c1, n1) + « , » + (c2, n2) + « , » + (c3, n3) + « ) FAUX »
    Fin SI
    // l'écriture (c1, n1) permet de colorier la valeur de n1 en la couleur que représente c1
Fin
```

Programme 2

Objectif : Écrire un programme qui à partir d'un entier p ≤ 100, génère et affiche la liste de tous les triplets (a, b, a + b) avec a, b, a + b ≤ p

ALGO : Afficher tous les triplets jusqu'à un certain rang

Variables locales : a, entier, premier nombre du triplet
b, entier, deuxième nombre du triplet
p, entier, valeur maximale autorisée pour a+ b

```
Début
    // détermination de la valeur maximale de a + b avec saisie sécurisée
    Afficher « Entrer un nombre entre 2 et 100 »
    p ← saisir
    Tant que (p < 2) OU (p > 100)
        Afficher « Entrer un nombre entre 2 et 100 »
        p ← saisir
    Fin tant que

    // détermination des triplets en enlevants les triplets déjà obtenu
    a ← 1
    b ← 1
    Tant que (a + b ≤ p)
        Si (a ≤ b)        // permet de déterminer si un triplet est déjà apparu
            Afficher « ( » + (a) + « , » + (b) + « , » + (a + b) + « ) »
        Fin si
        // augmentation des valeurs du triplet
        Si (b < p - a)
            b ← b+1
        Sinon
            b ← 1
            a ← a+1
        Fin si
    Fin tant que
Fin
```

Programme 3

Objectif : Écrire in programme qui utilise deux couleurs pour colorier N nombres entiers aléatoirement. Ce programme doit afficher VRAI si tous les triplets ne sont pas monochromatiques, FAUX sinon.

ALGO : Vrai si tous pas monochromatiques

Données : color, 1D de 2 entiers, rouge et bleu
cn, 1D de N entiers, couleur de chaque nombre

Variables locales : N, entier, valeur max de la somme
nb1, entier, premier nombre
nb2, entier, deuxième nombre
cpt, entier, compteur pour triplets monochromatiques
R, chaîne de caractères, rouge
B, chaîne de caractères, bleu
W, chaîne de caractères, noir
a, chaîne de caractères, couleur
i, entier compteur

```
Début
    color ← [R, B]      // initialisation des couleurs
    // Saisie User : Nombre de nombres à colorier
    Afficher « Entrer un nombre entre 2 et 100 »
    N ← saisir
    Tant que ((N<2) OU (N>100))
        Afficher « Entrer un nombre entre 2 et 100 v »
        N← saisir
    Fin Tant que

    //Création d'une liste aléatoire c avec 2 couleurs différentes et N valeurs
    Pour i allant de 0 à N-1
        a ← choix aléatoire dans la liste color
        cn[i] ← a
        Afficher (a, i+1)
    Fin Pour

    //Toutes les combinaisons possibles
    nb1 ← 1
    nb2 ← 2
    cpt ← 0
    Tant que (nb1 + nb2 ≤ N)
        Si (nb1≤nb2)
            Afficher « ( » + (cn[nb1-1], nb1), W + « , » + (cn[nb2-1], nb2),W + « , » + (cn[nb1+nb2-1],nb1+nb2), W +
            « ) »
            Si ((cn[nb1-1] = cn[nb2-1]) ET (cn[nb1-1] = cn[nb1+nb2-1])
                cpt ← 1
            Fin Si
        Fin Si
        Si (nb2 + nb1 < N)
            nb2 ← nb2 + 1
        Sinon
            nb2 ← 1
            nb1 ← nb1 + 1
        Fin Si
    Fin Tant que

    //Affichage de vraie ou faux
    Si (cpt=1)
        Afficher « Il y a au moins un triplet qui est monochromatique : FAUX »
    Sinon
        Afficher « Tous les triplets ne sont pas monochromatiques : VRAI »
    Fin Si
Fin
```

Programme 4

Objectif : Écrire un programme qui à partir d'un nombre de couleurs ***n*** et d'un nombre d'entiers ***N***, génère une coloration aléatoire. Il affiche VRAI si la coloration générée inclut les ***n*** couleurs utilisées. FAUX sinon.

```
ALGO : Coloration nombre pair et impair
Données :                color,1D de 6 entiers, contenant les 6 couleurs
                         c, 1D de n entiers, contenant les n couleurs aléatoires
Variables locales :  N, entier, nombre de nombre à colorier
                         n, entier, nombre de couleur
                         W, chaine de caractère, couleur blanche
                         R, chaine de caractère, couleur rouge
                         G, chaine de caractère, couleur verte
                         O, chaine de caractère, couleur orange
                         B, chaine de caractère, couleur bleu
                         P, chaine de caractère, couleur violet
                         i, entier, compteur 1
                         cpt, entier, compteur 2
                         j, entier, compteur 3
                         a, entier, conserver la valeur aléatoire de color
                         b, entier, conserver la valeur aléatoire de w
                         test, entier, si valide ou non
Début
    Afficher « Choisir le nombre de nombres à colorier : »
    N ← saisir
    Tant que (N≤0) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Choisir le nombre de nombres à colorier (>0) : »
        N ← saisir
    Fin tant que

    Afficher « Choisir un nombre de couleur (entre 1 et 6) : »
    n ← saisir
    Tant que (n<1) or (n>6) or (n>N) //saisie sécurisé n entre 1 et 6 et n<N
        Afficher « Choisir le nombre de nombres à colorier (<N) : »
        n ← saisir
    Fin tant que

    //Initialisation de la liste color
    color = ['W','R', 'G', 'O, 'B', 'P']
    //Création d'une liste aléatoire c avec n couleurs différentes
    Pour i allant de 1 à n
        a ← choix aléatoire d'une valeur dans (color)
        c.insert (i - 1, a)
    Fin Pour

    //Test s'il y a deux couleurs identiques dans c
    test ← 0
    Pour j allant de ('W','R','G','O','B','P')
        x ← c.count(j)
        Si (x > 1)
            test ← 1
        Fin si
    Fin Pour

    //Modification de la liste avec N valeurs
    Pour cpt allant de (n+1,N):
        b ← random.choice (c)
        c.insert (cpt,b)
    Fin pour
    random.shuffle (c)
    //Attribution de chaque valeur de la liste c à chaque Nombre
    Afficher "Pour n =" + (n) + "et N =" + (N) + "le coloriage"
    Pour p allant de (0, N-1)
        Afficher (p) de couleur c[p]
    Fin pour
    //Affichage si valide ou non
    Si (test = 1)
        Afficher (W) + " n'est pas valide"
    Sinon
        Afficher (W) + " est valide"
    Fin
Fin
```

Programme 7

Objectif : Écrire un programme qui convertit un nombre décimal en un nombre binaire.

ALGO : Conversion nombre décimal en binaire

Données : binaire, 1D de N entiers, valeurs binaires avec N valeurs

Variables locales : n, entier, nombre décimal choisit par l'user
quotient, réel
reste, entier
rang, entier, rang du reste obtenue dans la liste binaire
i, entier, compteur

```
Début
    Afficher « Saisir un nombre décimal pour le convertir en binaire : »
    n ← saisir

    Tant que (n≤0) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Saisir un nombre décimal POSITIF pour le convertir en binaire : »
        n ← saisir
    Fin tant que

    //1ère division
    reste ← n % 2
    quotient ← n/2
    quotient ← la partie entière du quotient

    rang ← 0 //Initialisation

    binaire.insert (rang, reste)

    //Toutes les autres divisions
    Tant que (quotient ≠ 0)
        reste ← quotient % 2
        quotient ← quotient/2
        quotient ← la partie entière du quotient
        rang ← rang-1
        binaire.insert (rang, reste)
    Fin tant que

    Pour i allant de 0 à -n
        print (binaire[i])
    Fin pour

Fin
```

Programme 8

Objectif : Écrire un programme qui convertit un nombre décimal en un nombre de base B.

ALGO : Conversion nombre décimal en binaire

```
Données :           conv, 1D de N entiers, valeurs de la base avec N valeurs
                    alphabet, 1D de 26 entiers, contenant toutes les lettres de l'alphabet

Variables locales :     n, entier, nombre décimal choisit par l'user
                        base, entier, nom de la base entre 1 et 36 choisit par l'user
                        quotient, réel
                        reste, entier
                        rang, entier, rang du reste obtenue dans la liste binaire
                        i, entier, compteur

Début
    Afficher « Saisir la base de conversion : »
    base ← saisir
    Tant que (base ≤ 1) or (base ≥ 36) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Saisir la base de conversion entre 1 et 36 : »
        base ← saisir
    Fin tant que

    Afficher « Saisir un nombre décimal pour le convertir en binaire : »
    n ← saisir
    Tant que (n≤0) //saisie sécurisé d'un entier supérieur à 0
        Afficher « Saisir un nombre décimal POSITIF pour le convertir en binaire : »
        n ← saisir
    Fin tant que

    alphabet=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

    //1ère division
    reste ← n % base
    Si (reste ≥ 10)
        reste ← alphabet[reste-10]
    Fin si

    quotient ← n/base
    quotient ← la partie entière du quotient

    rang ← 0 //Initialisation

    conv.insert (rang, reste)

    //Toutes les autres divisions
    Tant que (quotient ≠ 0)
        reste ← quotient % base
        Si (reste ≥ 10)
            reste ← alphabet[reste-10]
        Fin si

        quotient ← quotient/base
        quotient ← la partie entière du quotient
        rang ← rang-1
        conv.insert (rang, reste)
    Fin tant que

    Pour i allant de 0 à taille(conv)-1
        print (conv[i])
    Fin pour

FIN
```

Programme 9

Objectif : Écrire un programme qui stocke la liste de toutes les colorations possibles à partir de ***n*** couleurs pour ***N*** nombres entiers et où les ***n*** couleurs y figurent. Le programme doit afficher la liste obtenue à la fin.

ALGO : Conversion nombre décimal en binaire

Données : conv, 1D de N entiers, valeurs de la base avec N valeurs
alphabet, 1D de 26 entiers, contenant toutes les lettres de l'alphabet
liste, 1D de n^N listes, contenant toute les combinaisons possibles

Variables locales : n, entier, nombre décimal choisit par l'user
N, entier, nombre de nombre à colorier saisie par l'user
base, entier, nom de la base entre 1 et 36 choisit par l'user
quotient, réel
reste, entier
rang, entier, rang du reste obtenue dans la liste binaire
a, entier, compteur de combianison

```
Début
    //SAISIE USER
    //Nombre de nombres à colorier
    Afficher « Choisir le nombre de nombres à colorier : "
    N ← saisir
    Tant que (N ≤ 0)
        Afficher « Choisir le nombre de nombres à colorier : "
        N ← saisir
    Fin tant que

    //Nombre couleur
    Afficher « Choisir un nombre de couleur (entre 1 et 6) : "
    n ← saisir
    Tant que (n<1) ou (n>6) ou (n>N)
        Afficher « Choisir un nombre de couleur (entre 1 et 6) : »
        n ← saisir
    Fin tant que

    //PROGRAMME
    Pour a allant de(0,(n^N)-1)
        //COLORATION ...  Ici base = N
        //1ere division
        reste ← a%n
        quotient ← a/n
        quotient ← int(quotient) //On garde la partie entière du quotient
        rang ← 0 //Initialisation du rang
        conv.insert(rang,reste)

        //Toutes les autres divisions
        Tant que (quotient ≠ 0)
            reste = quotient%n
            quotient = quotient/n
            quotient = int(quotient) //On garde la partie entière du quotient
            rang = rang-1
            conv.insert(rang,reste)
        Fin tant que

        //On rajoute un 0 en plus pour pouvoir commencer la liste a l'index 1 et pas 0
        Si (len(conv) ≤ N)
            Pour I allant de (0,N-taille(conv)-1):
            conv.insert(0,0)
            Fin pour
        Fin si

        //Insertion de la combinaison dans une liste
        //Vérification si les n couleurs sont présents dans conv alors on insère conv dans liste
            liste.append(conv) //utilisation de append pour insérer une liste dans une liste
    Fin pour

    //AFFICHAGE
    Afficher « Il y a : » + (n**N) + « combinaisons. »
    Afficher « Les combinaisons sont : » +(liste)
Fin
```

APPLIQUER

Programme 1

Objectif : En se basant sur les programmes écrits précédemment, ainsi que le premier exemple illustratif, écrire un programme qui calcule ***S (2)***.

ALGO : Calcul de S(2)

```
Données :            binaire, entier, premier nombre du triplet

Variables locales :  N, entier, nombre de nombre
                     n, entier, compteur de combinaison
                     c_valide, entier, indicateur de combinaison valide
                     N_valide, entier, indicateur de nombre N valide
                     triplet_mono, entier, compteur de triplet mono
                     reste, entier
                     quotient, entier
                     rang, entier, rang du nombre à insérer dans binaire
                     a et b entier, rang 1 et 2 des triplet
Début
    N ← 1
    N_valide <--1

    //PROGRAMME
    Tant que (N_valide=1)
        N ← N+1
        c_valide ← 0
        Pour n allant de 0 à 2**N
            //COLORATION ...
            //1ere division
            reste ← n%2
            quotient ← n/2
            quotient ← partie entière du quotient
            rang ← 0 //Initialisation du rang

            binaire.insert(rang,reste)

            //Toutes les autres divisions
            Tant que (quotient ≠ 0)
                reste ← quotient%2
                quotient ← quotient/2
                quotient ← partie entière du quotient
                rang ← rang-1
                binaire.insert (rang,reste)
            Fin tant que

            Afficher (binaire)

            //LISTE DES TRIPLETS POUR LA COLORATION...
            // détermination des triplets en enlevant les triplets déjà apparu
```

```
                    a ← 1
        |           b ← 1
        |           triplet_mono ← 0
        |
        |           Tant que (a+b ≤ N+1)
        |               |   Si (a ≤ b ):
        |               |   |       Aficher (binaire[a]) + (binaire[b]) + (binaire[a+b])
        |               |   |       Si (binaire[a] = binaire[b]) et (binaire[a] = binaire[a+b])
        |               |   |       |       triplet_mono ← triplet_mono+1
        |               |   |       Fin si
        |               |   Fin si
        |               |
        |               |   Si (a+b < N)
        |               |   |       b ← b+1
        |               |   Sinon
        |               |   |       b ← 1
        |               |   |       a ← a+1
        |               |   Fin si
        |               Fin tant que
        |
        |               //Compteur de combinaison valide
        |               Si (triplet_mono = 0)
        |               |       c_valide ← c_valide+1
        |               Fin si
        |       Fin pour
        |
        |       //Condition pour que N soit valide
        |       Si (c_valide ≠ 0)
        |       |       N_valide ← 1
        |       Sinon
        |       |       N_valide ← 0
        |       Fin si
        |   Fin tant que
        |
        |   //AFFICHAGE FINAL
        |   Afficher « DONC S(2) = » + (N-1)
Fin
```

Programme 3

Objectif : Généraliser le programme et calculer ***S(n) avec n > 2***.

ALGO : Calcul de S(n)
Données : binaire, entier, premier nombre du triplet

Variables locales : N, entier, nombre de nombre
n, entier, nombre de couleur
cpt, entier, compteur de toutes les combinaisons
c_valide, entier, indicateur de combinaison valide
N_valide, entier, indicateur de nombre N valide
triplet_mono, entier, compteur de triplet mono
reste, entier
quotient, entier
rang, entier, rang du nombre à insérer dans binaire
a et b entier, rang 1 et 2 des triplet

```
Début
    N ← 1
    N_valide <--1

    //PROGRAMME
    Tant que (N_valide=1)
        N ← N+1
        c_valide ← 0
        Pour cpt allant de 0 à n**N
            //COLORATION ...
            //1ere division
            reste ← cpt%n
            quotient ← cpt/n
            quotient ← partie entière du quotient
            rang ← 0 //Initialisation du rang

            binaire.insert(rang,reste)

            //Toutes les autres divisions
            Tant que (quotient ≠ 0)
                reste ← quotient%n
                quotient ← quotient/n
                quotient ← partie entière du quotient
                rang ← rang-1
                binaire.insert (rang,reste)
            Fin tant que

            Afficher (binaire)

            //LISTE DES TRIPLETS POUR LA COLORATION...
            // détermination des triplets en enlevant les triplets déjà apparu
            a ← 1
            b ← 1
            triplet_mono ← 0
```

```
        Tant que (a+b ≤ N+1)
            Si (a ≤ b ):
                Aficher (binaire[a]) + (binaire[b]) + (binaire[a+b])
                Si (binaire[a] = binaire[b]) et (binaire[a] = binaire[a+b])
                    triplet_mono ← triplet_mono+1
                Fin si
            Fin si

            Si (a+b < N)
                b ← b+1
            Sinon
                b ← 1
                a ← a+1
            Fin si
        Fin tant que

        //Compteur de combinaison valide
        Si (triplet_mono = 0)
            c_valide ← c_valide+1
        Fin si
    Fin pour

    //Condition pour que N soit valide
    Si (c_valide ≠ 0)
        N_valide ← 1
    Sinon
        N_valide ← 0
    Fin si
Fin tant que

//AFFICHAGE FINAL
Afficher « DONC S (» + (n) + «)  = » + (N-1)
Fin
```

FONCTIONS AVANCEES

Programme 1

Objectif : Écrire un programme qui permet d'afficher pour n'importe quelle valeur de ***n*** (***n >= 6)***, l'intervalle dans lequel se trouverait son résultat ***N***.

ALGO : fact(n : entier) : entier
Donnée copiée : n, entier, nombre à calculer
Variables locales : i, entier, compteur
x, entier, calcul de n !

```
Début
    x ←1
    Pour i allant de 2 à n+1:
        x ← x * i
    Fin pour
    return x`
Fin
```

ALGO : Intervalle de S(n)
Variables locales : n, entier, nombre de couleur saisie par l'user

```
Début
    Afficher « Saisir une entier n ≥ 6 : »
    n ← saisir
    Tant que (n<6)
        Afficher « Saisir une entier n ≥ 6 : »
        n ← saisir
    Fin tant que

    Afficher « On sait que : » + (((3^n)-1)/2) + « ≤ S( » + (n) + « ) ≤ » + (3*fact(n)-1))
Fin
```

Programme 2

Objectif : Pour les plus motivés : Réécrire le programme ***S(n)*** en utilisant les fonctions.

LES FONCTIONS :

```
ALGO : conversion (nb,base,N : entiers) : entier
Donnée copiée :        nb, entiers, nombre à convertir
                       base, entiers, base de conversion
                       N, entiers, taille de la liste de conversion (nombres de nombres)

Donnée :               conv, 1D de N entiers, liste contenant la conversion

Variables locales :    base, entier, nom de la base entre 1 et 36 choisit par l'user
                       quotient, réel
                       reste, entier
                       rang, entier, rang du reste obtenue dans la liste binaire
                       i, entier, compteur
Début
    //1ere division
    reste = nb%base
    quotient = nb/base
    quotient = partie entière du quotient
    rang=0 //Initialisation du rang
    conv.insert(rang,reste)

    //Toutes les autres divisions
    Tant que (quotient ≠ 0)
        reste ← quotient%base
        quotient ← quotient/base
        quotient ← partie entière du quotient
        rang ← rang-1
        conv.insert(rang,reste)
    Fin tant que

    //On rajoute un 0 en plus pour pouvoir commencer la liste a l'index 1 et pas 0
    Si (taille(conv) ≤ N)
        Pour i allant de 0 à N-taille(conv)
            conv.insert(0,0)
        Fin pour
    Fin si

    return (conv)
Fin
```

```
ALGO : fact(n : entier) : entier
Donnée copiée :        n, entier, nombre a calculer

Variables locales :    x, entier, calcul
                       i, compteur
Début
    x ← 1
    Pour i allnt de 2 à n+1
        x ← x * i
    Fin pour
    return (x)
Fin
```

```
ALGO : intervalle_de_N (n : entier) : entier
Donnée copiée :        n, entier, nombre a calculer

Début
    Afficher « Pour une raison de temps on peut seulement donner un intervalle »
    Afficher « On sait que : » + (((3^n)-1)/2) + « ≤ S (» + (n) + «) ≤ » + (3*fact(n)-1))
Fin
```

```
ALGO : combinaison_valide (conv,N : entiers): entier
Donnée copiée :          conv, 1D de N entiers, combinaison
                         N, entiers, nombres de nombres

Variables locales :      triplet_mono, entier, indicateur de triplet mono
                         a, entier, valeur 1 des triplets
                         b, entier, valeur 2 des triplets
                         c_valide, entier, indicateur de combinaison valide

Début
    //LISTE DES TRIPLETS POUR LA COLORATION...
    //détermination des triplets en enlevant les triplets déjà apparu
    a ← 1
    b ← 1
    triplet_mono ← 0
    c_valide ← 0

    Tant que (a+b ≤ N)
        Si (a ≤ b )      // comme b augmente avant a, on enlève les valeurs de a qui sont plus grandes que b
            //AFFICHAGE DES TRIPLETS
            Si (conv[a] = conv[b]) et (conv[a] = conv[a+b])
                triplet_mono ← triplet_mono+1
            Fin si
        Fin si

        Si (a+b < N)      // permet d'augmenter la valeur de b en vérifiant que b ne seras pas trop grand
            b ← b+1
        Sinon             // permet d'augmenter a de 1 et de remettre b à 1 quand b est trop grand
            b ← 1
            a ← a+1
        Fin si
    Fin tant que

    //Compteur de combinaison valide
    Si (triplet_mono = 0) //S'il y a pas de triplet monochromatique
        c_valide ← 1
    Fin si

    return(c_valide)
Fin
```

```
ALGO : nombre_N_valide (n : entier) : entier
Donnée copiée :          n, entier, correspond à c_valide

Variables locales :      x, entier, correspond à N_valide

Début
    //Si affichage 1 alors combinaison valide
    //Si affichage 0 alors combinaison pas valide
    //Condition pour que N soit valide
    Si (n ≠ 0) //S'il y a au moins une combinaison valide
        x ← 1
    Sinon
        x ← 0
    Fin si

    return (x)
Fin
```

```
ALGO : affichage (n,N : entiers) : entiers
Donnée copiée :          n, entier, nombre de couleur
                         N, entier, nombres de nombres
Début
    //AFFICHAGE FINAL
    Afficher « DONC S (» + (n) + «) = » + (N-1) //valide pour Nmax=N-1 car pour N plus valide
Fin
```

```
ALGO : saisie_de_n() : entier
Début
    //Saisie de N
    Afficher « Saisir n ≥ 2 : »
    n ← saisir

    Tant que (n<2)
        Afficher « Saisir n ≥ 2 : »
        n ← saisir
    Fin tant que

    Si (n>2) et (n<6)
        Afficher « Merci de patienter ... »
    Fin si

    return n
Fin
```

PROGRAMME :

```
ALGO : Programme
Donnée :            conv, 1D de N entiers, combinaison

Variables locales :     N_valide, entier, indicateur de N valide
                        c_valide, entier, indicateur de combinaison valide
                        N, entier, nombres de nombres
                        n, entier, nombre de couleur
                        a, entier, compteur

Début
    //Initialisation
    N_valide ←1
    N ← 1

    //Saisie de N
    n ← la partie entière de (saisie_de_n())

    //PROGRAMME POUR n<6
    Si (n < 6)
        Tant que (N_valide ≠ 0)
            N ←N+1
            c_valide ←0
            Pour a allant de 0 à n^N
                conv ←conversion(a,n,N)
                c_valide ← c_valide + combinaison_valide(conv,N)
            Fin pour
            N_valide ← nombre_N_valide(c_valide)
        Fin tant que
        affichage (n,N)

    //PROGRAMME POUR n≥6
    Sinon
        intervalle_de_N(n)
    Fin si
Fin
```