Version itérative en langage algorithmique:

```
Algorithme itératif;
 Structure Stack
        top : entier
        capacity : entier
        array : tableau d'entiers
        name : Chaîne de caractères

    Fonction createStack(capacity: entier, name: chaine de caractère) → Stack

      Allouer(stack);
        stack.capacity ← capacity;
        stack.top ← -1;
        allouer (stack.array) de taille capacity
        stack.name ← name;

        Retourner( stack);
    Fin Fonction

    Procédure push(stack:* Stack, item: entier)
        stack.array[++stack.top] ← item
    Fin Procédure

    Fonction pop(stack: *Stack) → entier
        Si (stack.top = -1) Alors
            Retourner (-1);
        Sinon
            Retourner (stack.array[stack.top--]);
        Fin Si;
    Fin Fonction

    Fonction top(stack: *Stack) → entier
        Si (stack.top = -1) Alors
            Retourner (-1);
        Sinon
            Retourner (stack.array[stack.top]);
        Fin Si;
    Fin Fonction

    Procédure moveDiskIteratif(source: *Stack, target: *Stack)
        Variables
            diskFromSource : entier;
            diskFromTarget : entier;
        Fin Variables
```

```
        diskFromSource ← top(source);
        diskFromTarget ← top(target);

        Si (diskFromSource ≠ -1) ET (diskFromTarget = -1 OU diskFromSource <
diskFromTarget) Alors
            diskFromSource ← pop(source);
            push(target, diskFromSource);
            Afficher( "Déplacement du disque " + diskFromSource + " de " + source.name + "
vers " + target.name);
        Sinon Si (diskFromTarget ≠ -1) ET (diskFromSource = -1 OU diskFromTarget <
diskFromSource) Alors
            diskFromTarget ← pop(target);
            push(source, diskFromTarget);
            Afficher ("Déplacement du disque " + diskFromTarget + " de " + target.name + " vers
" + source.name);
        Fin Si
    Fin Procédure

    Fonction tohIteratif(n: entier, source: *Stack, target:*Stack, auxiliary: *Stack) → entier
        Variables
            total_moves : entier;
            moveCount : entier;
        Fin Variables

        total_moves ← (1 << n) - 1;
        moveCount ← 0;

        Si (n MOD 2 = 0) Alors
          Stack *temp = target;
        target = auxiliary;
        auxiliary = temp;
        Fin Si

        Tant que (moveCount < total_moves) Faire
            Si (moveCount MOD 3 = 0) Alors
                moveDiskIteratif(source, target);
            Sinon Si (moveCount MOD 3 = 1) Alors
                moveDiskIteratif(source, auxiliary);
            Sinon
                moveDiskIteratif(auxiliary, target);
            Fin Si
          moveCount←moveCOunt +1;
        Fin Tant que

        Retourner (moveCount);
    Fin Fonction

    Algorithme principal
```

```
    Variables
       n : entier
       source, auxiliary, target : *Stack;
       moveCount : entier;
    Fin Variables

    Afficher "Entrez le nombre de disques: ";
    Lire (n);

    source ← createStack(n, 'A');
    auxiliary ← createStack(n, 'B');
    target ← createStack(n, 'C');

    Pour (i de n a 0)Faire
       push(source, i);
      i←i-1;
    Fin Pour

    Afficher "Solution Itérative: ";
    moveCount ← tohIteratif(n, source, target, auxiliary);

    Afficher ("Nombre de décalages: " + moveCount);
  Fin Algorithme principal
```

Version recursive en langage algorithmique:

```
Structure Stack
  Variables
     top : entier;
     capacity : entier;
     array : tableau d'entiers;
     name : caractère;
  Fin Variables

  Fonction createStack(capacity: entier, name: caractère) → Stack
     Variables
        stack : *Stack
     Fin Variables

 ALlouer(Stack);
     stack.capacity ← capacity;
     stack.top ← -1;
     allouer(stack.array) de taille capacity
     stack.name ← name;
```

```
    Retourner( stack);
  Fin Fonction

  Procédure push(stack: *Stack, item: entier)
    stack.array[++stack.top] ← item;
  Fin Procédure

  Fonction pop(stack: *Stack) → entier
    Si (stack.top = -1) Alors
      Retourner (-1);
    Sinon
      Retourner (stack.array[stack.top--]);
    Fin Si
  Fin Fonction

  Fonction top(stack: référence à Stack) → entier
    Si (stack.top = -1) Alors
      Retourner (-1):
    Sinon
      Retourner (stack.array[stack.top]);
    Fin Si
  Fin Fonction

  Procédure moveDiskRecursif(source: *Stack, target: *Stack)
    Variables
      disk : entier;
    Fin Variables;

    disk ← pop(source);
    push(target, disk);
    Afficher ("Déplacement du disque " + disk + " de " + source.name + " vers " +
target.name);
  Fin Procédure

  Fonction tohRecursif(n: entier, source:*Stack, auxiliary:*Stack, target: *Stack) → entier
    Variables
      moveCount : entier;
    Fin Variables

    moveCount ← 0;
    Si (n > 0) Alors
      moveCount ← moveCount + tohRecursif(n - 1, source, target, auxiliary);
      moveDiskRecursif(source, target);
      moveCount←moveCount+1;
      moveCount ← moveCount + tohRecursif(n - 1, auxiliary, source, target);
    Fin Si
```

```
    Retourner (moveCount);
  Fin Fonction

  Algorithme principal
    Variables
      n : entier
      source, auxiliary, target : *Stack
      moveCount : entier;
    Fin Variables

    Afficher( "Entrez le nombre de disques: ")
    Lire (n);

    source ← createStack(n, 'A');
    auxiliary ← createStack(n, 'B');
    target ← createStack(n, 'C');

    Pour (i de n a 0) Faire
      push(source, i);
    i←i-1;
    Fin Pour

    Afficher ("Solution Récursive: ");
    moveCount ← tohRecursif(n, source, auxiliary, target);

    Afficher ("Nombre de décalages: " + moveCount);
  Fin Algorithme principal
```

Algorithme de verification en langage algorithmique:

```
Algorithme Structure Stack
  Variables
    top : entier;
    capacity : entier;
    array : tableau d'entiers;
    name : caractère;
  Fin Variables

  Fonction createStack(capacity: entier , name: caractère) → Stack
    Variables
      stack :*Stack
    Fin Variables
```

```
    stack ← allouer mémoire pour Stack
    stack.capacity ← capacity
    stack.top ← -1
    allouer mémoire pour stack.array de taille capacity
    stack.name ← name

    Retourner stack
Fin Fonction

Procédure push(stack: référence à Stack, item: entier)
    stack.array[++stack.top] ← item
Fin Procédure

Fonction pop(stack: référence à Stack) → entier
    Si stack.top = -1 Alors
        Retourner -1
    Sinon
        Retourner stack.array[stack.top--]
    Fin Si
Fin Fonction

Fonction top(stack: référence à Stack) → entier
    Si stack.top = -1 Alors
        Retourner -1
    Sinon
        Retourner stack.array[stack.top]
    Fin Si
Fin Fonction

Fonction isEmpty(stack: référence à Stack) → entier
    Retourner stack.top == -1
Fin Fonction

Fonction isComplete(target: référence à Stack, n: entier) → entier
    Si target.top ≠ n - 1 Alors
        Retourner 0
    Fin Si

    Pour i ← 0 à n - 1 Faire
        Si target.array[i] ≠ n - i Alors
            Retourner 0
        Fin Si
    Fin Pour

    Retourner 1
Fin Fonction

Procédure moveDisk(source: référence à Stack, target: référence à Stack)
```

```
    Variables
        disk : entier
    Fin Variables

    disk ← pop(source)
    push(target, disk)
    Afficher "Déplacer disque " + disk + " de tour " + source.name + " à tour " + target.name
Fin Procédure

Procédure hanoi(n: entier, source: référence à Stack, auxiliary: référence à Stack, target: référence à Stack)
    Si n = 1 Alors
        moveDisk(source, target)
    Sinon
        hanoi(n - 1, source, target, auxiliary)
        moveDisk(source, target)
        hanoi(n - 1, auxiliary, source, target)
    Fin Si
Fin Procédure

Fonction verification(n: entier, source: référence à Stack, auxiliary: référence à Stack, target: référence à Stack) → entier
    Si isEmpty(source) ET isEmpty(auxiliary) ET isComplete(target, n) Alors
        Afficher "Les disques ont été déplacés avec succès vers la tour C."
        Retourner 1
    Sinon
        Afficher "Les disques n'ont pas été déplacés avec succès."
        Retourner 0
    Fin Si
Fin Fonction

Algorithme principal
    Variables
        n : entier
        source, auxiliary, target : références à Stack
    Fin Variables

    n ← 3  // Nombre de disques

    source ← createStack(n, 'A')
    auxiliary ← createStack(n, 'B')
    target ← createStack(n, 'C')

    Pour i ← n à 1 Pas de -1 Faire
        push(source, i)
    Fin Pour

    Afficher "État initial des tours :"
```

```
    Afficher "Tour A : "
    Pour i ← source.top à 0 Pas de -1 Faire
        Afficher source.array[i] + " "
    Fin Pour
    Afficher "\nTour B : "
    Pour i ← auxiliary.top à 0 Pas de -1 Faire
        Afficher auxiliary.array[i] + " "
    Fin Pour
    Afficher "\nTour C : "
    Pour i ← target.top à 0 Pas de -1 Faire
        Afficher target.array[i] + " "
    Fin Pour
    Afficher "\n\n"

    hanoi(n, source, auxiliary, target)

    verification(n, source, auxiliary, target)

    Libérer la mémoire allouée pour les piles
Fin Algorithme principal
```