Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

Département Intelligence Artificielle et Sciences des données

Spécialité: Système Informatique Intelligent

Module: Complexité des algorithmes

Rapport du TP 03 : La Tour de Hanoï

Année universitaire : 2022/2023

Ce rapport a été réalisé par :

“TEAM 28”

NOM:	**PRÉNOM:**	**MATRICULE:**
BEHLOUL	HALA LYNA	191931068962
HENDEL	LYNA MARIA	191931068959
NAAR	SARAH	181831087978
NADIR	SOMIA	181833020729

TABLE DES MATIÈRES:

Introduction:

Le problème des tours de Hanoï est un jeu faisant partie de la catégorie des casse-têtes. Dans la recherche en psychologie de la résolution de problème, employé comme épreuve lors d'une évaluation neuropsychologique des fonctions exécutives, Il est très exploité, étudié en mathématiques et souvent utilisé en algorithmique pour montrer la puissance et l'intérêt de la récursivité.

Historique:

Les Tours de Hanoi, (également appelées Tours de Brahma ou Tours de Lucas en référence à son créateur et mathématicien français Édouard Lucas) sont un jeu mathématique ou puzzle inventé en 1883.
le professeur Claus de Siam qui raconte une histoire ayant lieu au cœur du temple Kashi Vishwanath :

Trois aiguilles de diamant, plantées dans une dalle d'airain.Sur une de ces aiguilles, Dieu enfila au commencement des siècles, 64 disques d'or pur, le plus large reposant sur l'airain, et les autres, de plus en plus étroits, superposés jusqu'au sommet.

C'est la tour sacrée du Brahmâ. Nuit et jour, les prêtres se succèdent sur les marches de l'autel, occupés à transporter la tour de la première aiguille sur la troisième, sans s'écarter des règles fixes que nous venons d'indiquer, et qui ont été imposées par Brahma. Quand tout sera fini, la tour et les brames tomberont, et ce sera la fin des mondes !

Fait amusant :

D'après ce qui a été mentionné ci-dessus, si nous utilisons 64 disques, nous aurons donc besoin de 2 puissance 64 − 1 déplacement. En supposant que chaque déplacement dure 1 seconde, on obtient 86 400 déplacements par jour, le jeu se termine après 584,5 milliards d'années approximativement, soit une quarantaine de fois l'âge de l'univers.

Présentation du problème:

Les tours de Hanoï sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, qui consiste à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

-> On peut déplacer au maximum un seul disque par déplacement.

-> Seul un **p**etit disque peut être placé sur un plus gros disque.

Définition formelle du problème:

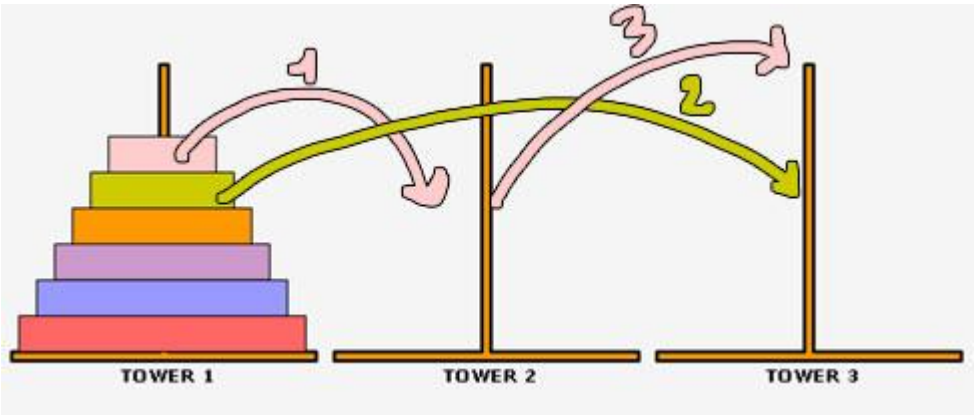
Il est facile de démontrer par récurrence que si n est le nombre de disques, il faut 2^n-1 coups au minimum pour parvenir à ses fins, quantité qui augmente très rapidement avec n. En effet, soient A, B et C les trois emplacements des tours ; notons xn le nombre de déplacements de disques nécessaires au déplacement d'une tour complète. Pour déplacer une tour de n disques de A vers C, on effectue ces trois étapes

SOLUTION DES TOURS DE HANOI:

La résolution des Tours de Hanoi n'est pas difficile lorsque l'on connaît l'astuce.

- Si le nombre total de disques est pair. Alors,il faut placer le premier disque sur la deuxième tour.
- Si le nombre total de disques est impair. Alors, il faut placer le premier disque sur la troisième tour.

S'il n'y a pas de bâtonnet libre, on revient à l'autre extrémité comme s'il s'agissait d'un cercle.



La structure utilisée:

Comme la pile ne permet l'accès qu'à un seul de ses éléments, son usage est très utile pour remplacer la récursivité.
Le jeu entier est représenté alors par 3 piles (tour source / intermédiaire / destination) qui ont chacune un tableau contenant les disques, un élément top pour accéder au disque le plus petit (celui placé en haut de la tour) et un élément indiquant la taille du tableau (c.a.d le nombre totale des disques).

```
Struct Stack {
    int capacity;
    int top;
    int *array; };
```

L'algorithme de la tour de hanoi :

Pour résoudre le problème nous utiliserons deux méthodes, l'une étant récursive et l'autre itérative.
Dans l'état initial, les n disques sont placés sur la tour de départ . Dans l'état final, tous les disques se retrouvent placés dans le même ordre sur la tour d'arrivée.

- **Version itérative**

```
Procédure toH (E/entier nb_disk, E/S Stack *src,Stack *aux, Stack *dest)
```

```
Debut
    entier total_moves,i;
    caractere A='S',B='A',C='D'  //des caractères utilisé pour l'affichage des étapes de déplacement.
    Si (nb_disk mod 2 ==0) Alors   //Si le nb est pair alors on permute la tour d'arrivé et la tour intermédiaire car le commencement de déplacement dépend de la parité des disques
            Stack temp <- dest;
            dest <- aux;
            aux <- temp;
```

```
                char tempc <-C;  C<-B;   B<-tempc;
        Fsi;
        total_moves = pow (2,nb_disk) - 1 ;
        pour i allant de nb_disk a 1 pas -1 Faire
           empiler ( src , i ) // remplissage de la 1ere pile
                               // tour de depart
        pour i allant de 1 a total_moves Faire
           Si( i mod 3 == 1 ) Alors //1er deplacement
               moveDisk(src,dest,A,C);
           Si ( i mod 3 == 2) Alors //2eme deplacement
               moveDisk(src,aux,A,B);
           Si  ( i mod 3 == 0) Alors //3eme deplacement
               moveDisk(aux,dest,B,C)
       Fait;
   Fin
```

Procedure moveDisk(Stack *src,Stack *dest,car A,car B)

```
        entier disk1= depiler(src);
        entier disk2= depiler(dest);
       Si (disk1 == -1) //Si la première tour est vide alors on
deplace ce disque vers la 1ere tour
          empiler(src,disk2);
          move(C,A,disk2);
       Sinon Si (disk2 == -1) //Si le 2eme tour est vide alors on
deplace le disque vers la 2eme tour
          empiler(dest,disk1);
          move(A,C,disk1)
       Sinon Si (disk1>disk2) //si le premier disque de la 1ere
tour est plus grand que le 2eme disque de la 2eme tour alors on
remet le 1ere disque sur la 1ere tour puis on deplace le 2eme
disque vers la tour de depart
          empiler(src,disk1);
```

```
          empiler(src,disk2);
          move(C,A,disk2);
       Sinon    //si le 2eme disque est plus grand alors on fait
le contraire
          empiler(dest,disk2);
          empiler(dest,disk1);
          move(A,C,disk1);
       FSI;
   Fin.
```

```
  Procedure move ( car S, car D,int disk)
  //fonction utilisé juste pour afficher les etapes de la
resolution
```

```
    Ecrire (Move the disk n° disk from S to D);
```

Calcul de Complexité :

- **Temporelle** : puisque la complexité est exprimé en matière de nombre d'opérations de déplacement effectuées, alors on a qu'à évaluer le nombre de déplacement comme suit :
  - Au début on a n-1 déplacement de la tour de départ vers la tour auxiliaire,donc Dn-1 déplacements.
  - puis on déplace le disque le plus grand de la tour de départ vers la tour d'arrivée, donc un seul déplacement.
  - et enfin on aura une seconde fois des n-1 déplacement de la tour auxiliaire vers la tour d'arrivée, Dn-1 déplacements.

Le nombre totale de déplacements correspond donc à :

$D_0 = 0$

$D_n = 2\,D_{n-1} + 1 \quad si\ D >= 1$

D'où $D_n = 2^n - 1$, la complexité est donc d'ordre exponentielle $O(2^n)$ **).**

- **Spatiale** : O(N) où N est le nombre total de disques.

- **Version récursive**

On déplace les n−1 disques de la tour de départ vers la tour auxiliaire par un appel récursif. Puis, le plus grand disque restant sera déplacé vers la tour d'arrivée.
Ensuite, les n − 1 disques qui se trouvaient sur la tour intermédiaire seront déplacés vers la tour d'arrivée par le même processus récursif et dans le cas où il reste un seul disque on le déplace directement.

```
Procedure toHRec (int n,char S,char A, char D)
```

```
      Debut
        Si(n==1) Alors
           //s'il reste un seul disque alors on le déplace
directement
            //deplacer(S,D,disk);
             ecrire("move disk 1 from S to D");
       Fsi;
       toHRec(n-1,S,A,D);
       //deplacer(S,D,disk)   //Deplacer le disque superieur de
la tour de depart vers la tour d'arrivé
       ecrire("move disk from S to D")
       toHrec(n-1,A,D,S);

    Fin
```

Calcul de Complexité :

- **Temporelle** : La même que l'approche récursive car le même principe est appliqué. $O(2^n)$
- **Spatiale** : elle est linéaire d'ordre O(N), tel que N est l'espace mémoire réservée pour les appels récursifs.

L’algorithme de vérification:

l’algorithme de vérification sert à s'assurer que les disques sont bien placés la fin de l'exécution de l’algorithme de la tour de Hanoi
pseudo-code :

```
entier verif ( entier n , Stack * dest)
```

```
entier disk;
pour i allant de 0 a n faire
      disk = dépiler(dest);
      Si ( disk <> i+1 ) Alors  retourner 0;  //faux FSI;
Fait;
retourner 1;     //Vrai
```

Calcul de Complexité :

- La complexité temporelle est égale à O(n), tel que le n est le nombre de disques.
- La complexité spatiale est la même que la complexité temporelle puisqu' elle prend que la tour d'arrivée comme argument

ETUDE EXPÉRIMENTALE :

Pour n disques il faut faire 2 n − 1 déplacements, donc pour de grandes valeurs de n, il est très probable que le programme soit très lent et "plante" au vue des limitations de la pile d’exécution.

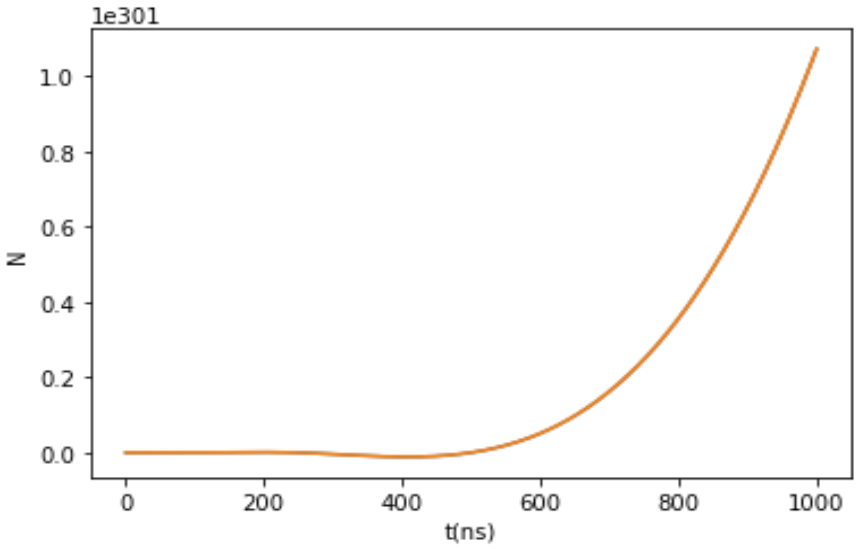
Nombre de nombre	**5**	**10**	**15**	**20**	**25**
temps dex	0.015000	0.078000	2.281000	83.587000	4586.719000

- Simulation de la complexité théorique de l'algorithme de résolution :

-La complexité de l'algorithme de résolution comme calculée précédemment est de l'ordre de O(2n), pour que nous puissions faire la simulation, nous utiliserons le tableau suivant qui représente les temps d'exécutions théoriques selon la variation de la taille du problème.

N	**1**	**5**	**10**	**50**	**100**	**250**	**500**	**1000**
t(ns)	2	31	1024	1.1258999E+15	1.2676506E+30	1.8092514E+75	3.2733901E+150	1.0715099E+301

Représentation graphique de la complexité:

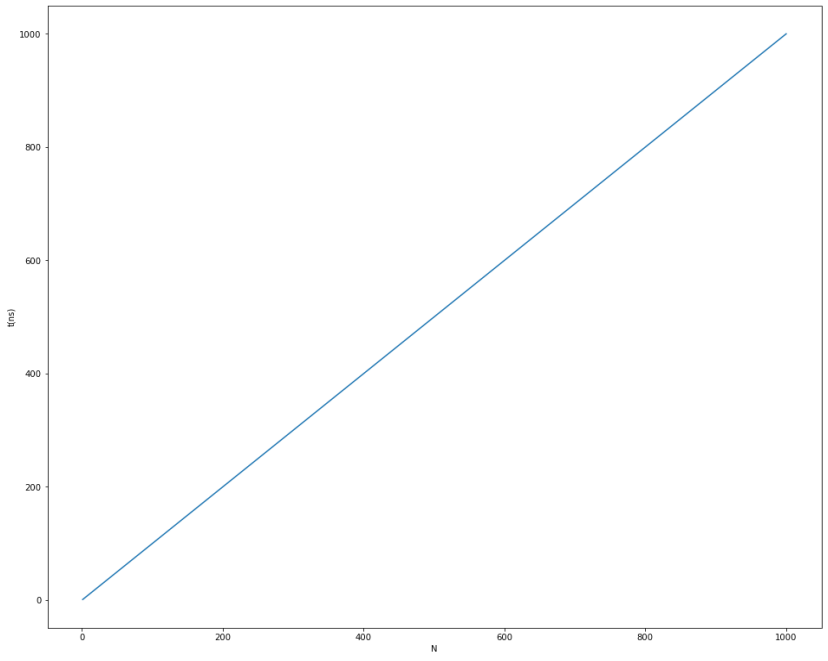


- Simulation de la complexité de l’algo de vérification:

La complexité temporelle de l’algorithme de vérification est linéaire et elle est égale à O(n), vu que cet algo ne prend que la tour d'arrivée (état final) comme argument.
Le tableau suivant représente les temps d’exécution théorique selon la variation de la taille du problème (nombre de disque) :

N	**1**	**10**	**50**	**150**	**500**	**1000**
t(ns)	1	10	50	150	500	1000

Représentation graphique de la complexité:



Algorithme de vérification :

- le meilleur cas est quand le premier disque est plus grand que celui en dessu et dans ce cas l'algorithme s'arrête directement
- le cas moyen correspond au cas ou on trouve l'erreur au milieu de la tour
- et dans le pire cas on une tour d'hanoi bien construite et donc notre algorithme va parcourir toute la pile

- Représentation d'une instance du problème:

On peut illustrer ce problème en prenant comme petit exemple avec trois disques. Donc le nombre total de déplacements sera égal à 7.

Exemple

Pour n = 3, on à la configuration suivante :

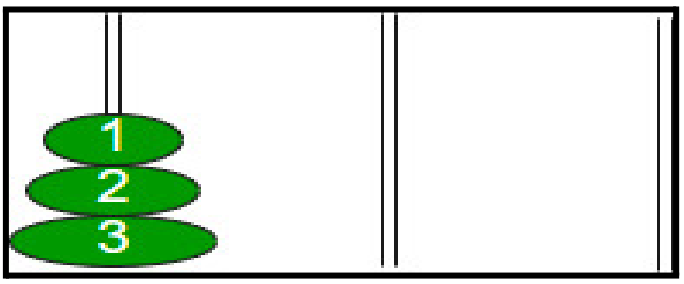
Tours de Hanoï : état initial

Tours de Hanoï : état final

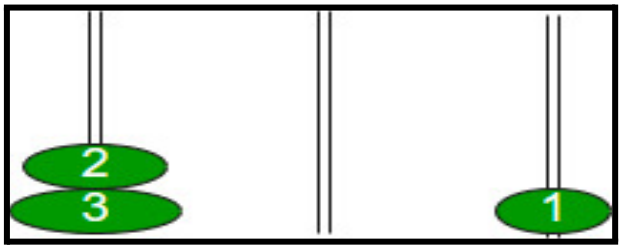
Dans l'état initial, les n disques sont placés sur la tour **« départ »**. Dans l'état final, tous les disques se retrouvent placés dans le même ordre sur la tour **« arrivée ».**

A un moment donné, dans la suite des opérations à effectuer, il faudra déplacer le disque numéro n (le plus grand, placé initialement en dessous de la pile de disques) de la tour « départ » à la tour « arrivée ». Pour pouvoir effectuer ce déplacement, il faut d'une part qu'il n'y ait plus aucun disque sur le disque n et d'autre part que la tour « arrivée » soit vide. En conséquence, il faut que tous les autres disques (de 1 à (n−1)) soient sur la tour « intermédiaire ».

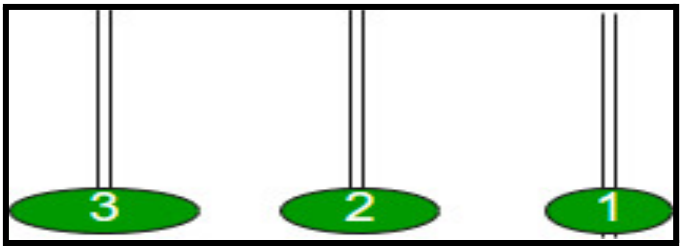
pour faire notre représentation, nous noterons les disques par ; 1 , 2 , 3 du plus petit au plus grand comme ceci :



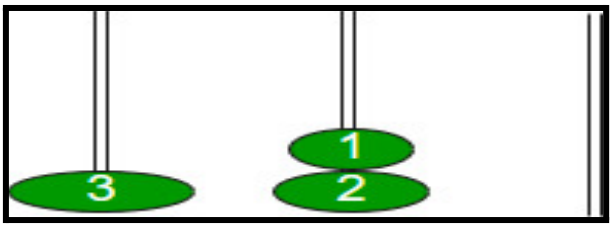
Quand i=1, (i%3 ==1) déplacements entre “`S' et 'D'`



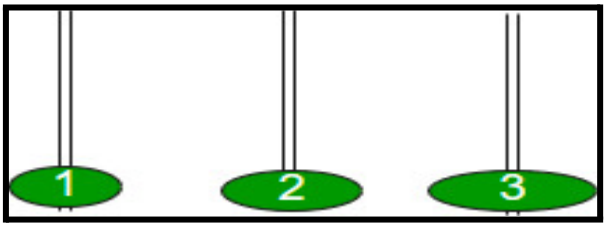
Quand I=2, (i% 3 == 2) déplacements



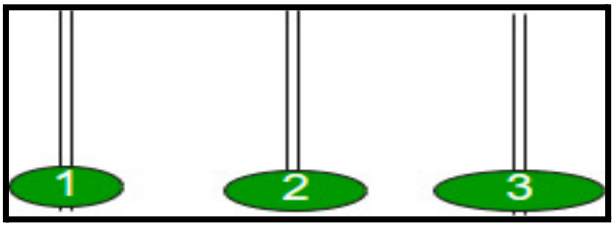
Quand i = 3, (i % 3 ==0) déplacements entre ‘A’ et ‘D’



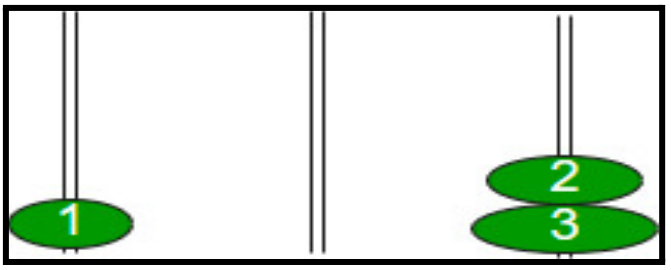
quand i = 4,(i % 3 == 1) déplacements entre ‘S’ et ‘D



Quand i =5, (i%3 ==2) déplacements entre ‘S’ et ‘A’



Quand i=6, (i%3 == 0) déplacements entre ‘A’ et ‘D’



Quand i =7, (i%3 ==1) déplacements entre ‘S’ et ‘D’

Ainsi, après tous ces pôles de destination contiennent tous les disques dans l'ordre de leur taille.
Après avoir observé les itérations ci-dessus, on peut penser qu'après le déplacement d'un disque autre que le plus petit, le prochain disque à déplacer doit être le plus petit car c'est le

disque supérieur reposant sur le pôle de réserve et il n'y a pas d'autres choix pour déplacer un disque.

Comparaison de l’étude théorique et expérimentale :

Nous avons vu en théorie que la complexité de l’algorithme des tours de Hanoi est exponentielle O(2nn).
Ceci a bien été ressenti dans les résultats obtenus après les tests. En effet, nous avons remarqué (d’après le graphe) que le temps d’exécution évolue de manière exponentielle et augmente très rapidement à partir d’un certain nombre de disques (25 dans notre cas).
En effectuant quelques calculs, on peut déduire le temps d’exécution pour 35 disques (Sur la machine du test). Ce dernier est égal à environ 10 jours, raison pour laquelle on a arrêté les tests à 25 disques.

Conclusion:

Les résultats que nous avons obtenus à travers l’étude que nous avons menée démontrent de manière très radicale que même si la version récursive proposée est plus facile à implémenter et comprendre, elle ne donne pas toujours des résultats promettant et donc en l'occurrence elle n’est pas toujours optimale et ne diminue pas pour autant la complexité de tous les problèmes auxquels on essaye de trouver une solution.

Répartition des tâches:

- BEHLOUL Hala Lyna : Réalisation des algorithmes et calcul de leurs complexités, illustration de l'exemple et rédaction du rapport .

- HENDEL Lyna Maria: Réalisation des algorithmes, calcul de leurs complexités et rédaction du rapport .

- NAAR Sarah: Réalisation des algorithmes, calcul de leurs complexités et rédaction du rapport .

- NADIR Somia: Réalisation des algorithmes, calculs de leurs complexités, réalisation des graphes et exécution des tests.

ANNEXE: CODE SOURCE

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>


struct Stack
{
int capacity;
int top;
int *array;
};


struct Stack* createStack(int capacity)
{
	struct Stack* stack =
		(struct Stack*) malloc(sizeof(struct Stack));
	stack -> capacity = capacity;
	stack -> top = -1;
	stack -> array =
		(int*) malloc(stack -> capacity * sizeof(int));
	return stack;
}


int isFull(struct Stack* stack)
{
return (stack->top == stack->capacity - 1);
}


int isEmpty(struct Stack* stack)
{
return (stack->top == -1);
}
```

```
void push(struct Stack *stack, int item)
{
	if (isFull(stack))
		return;
	stack -> array[++stack -> top] = item;
}


int pop(struct Stack* stack)
{
	if (isEmpty(stack))
		return INT_MIN;
	return stack -> array[stack -> top--];
}


void moveDisk(char fromPeg, char toPeg, int disk)
{
	printf("Move the disk %d from \'%c\' to \'%c\'\n",
		disk, fromPeg, toPeg);
}



void moveDisksBetweenTwoRods(struct Stack *from,
			struct Stack *to, char A, char C)
{
	int rod1TopDisk = pop(from);
	int rod2TopDisk = pop(to);


	if (rod1TopDisk == INT_MIN)
	{
		push(from, rod2TopDisk);
		moveDisk(C, A, rod2TopDisk);
	}


	else if (rod2TopDisk == INT_MIN)
	{
		push(to, rod1TopDisk);
		moveDisk(A, C, rod1TopDisk);
```

```
        }


        else if (rod1TopDisk > rod2TopDisk)
        {
                push(from, rod1TopDisk);
                push(from, rod2TopDisk);
                moveDisk(C, A, rod2TopDisk);
        }


        else
        {
                push(to, rod2TopDisk);
                push(to, rod1TopDisk);
                moveDisk(A, C, rod1TopDisk);
        }
}

// **************FONCTION ITERATIVE*****************
void toH(int no_of_disks, struct Stack
                        *from, struct Stack *_using,
                        struct Stack *to)
{
        int i, total_num_of_moves;
        char A = 'S', C= 'D', B = 'A';


        if (no_of_disks % 2 == 0)
        {
                char temp = C;
                C = B;
                B = temp;
                struct Stack *temp1 = to;
                to=_using;
                _using=temp1;
        }
        total_num_of_moves = pow(2, no_of_disks) - 1;


        for (i = no_of_disks; i >= 1; i--)
                push(from, i);
```

```
        for (i = 1; i <= total_num_of_moves; i++)
        {
                if (i % 3 == 1)
                moveDisksBetweenTwoRods(from, to, A, C);

                else if (i % 3 == 2)
                moveDisksBetweenTwoRods(from,_using, A, B);

                else if (i % 3 == 0)
                moveDisksBetweenTwoRods(_using, to, B, C);
        }
}
//******************ALGO RECURSIVE*******************
void toHRec(int n, char rodA, char rodC, char rodB)
{
        if (n == 1)
        {
                printf("\n Move disk 1 from rod %c to rod %c",rodA ,rodC );
                return;
        }
        toH(n-1, rodA, rodB, rodC);
        printf("\n Move disk %d from rod %c to rod %c", n, rodA, rodC);
        toH(n-1, rodB, rodC,rodA);
}
//******************ALGO DE VERIFICATION *******************
int verif(int n,struct Stack
                        * p){
    int disk;

    for(int i=0;i<n;i++){
       disk=pop(p);
       printf("disk nb %d \n",disk);
       if(disk!=i+1){
          return 0;
       }
    }

    return 1;

}
int main()
```

```
{

        int no_of_disks=25;

  clock_t t1,t2;
    double delta;
        struct Stack *from, *_using, *to;


        //toHRec(no_of_disks, 'A','C','B');

  //3 piles pour lalgo iterative
        from = createStack(no_of_disks);
        _using = createStack(no_of_disks);
        to = createStack(no_of_disks);

        toH(no_of_disks, from,_using,to);

     t1=clock();
    if(verif(no_of_disks,to)==0){
       printf("False");
    }else {
        printf("TRUE");
    }
    t2=clock();
    delta=(double) (t2-t1)/CLOCKS_PER_SEC;
    printf(" \n  temps dex dalgo de verif : %lf \n",delta);
        return 0;
}
```