Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

MASTER SII 2022 /2023

Projet 03 TP Complexité

Fait par le quadrinôme :

ABDOU Maria hind 161631069767

FERRADJI Tarek 181931058767

YOUSFI Zakaria 171732026950

MOUAZ Rayane Hamza 191931030386

Table des matières

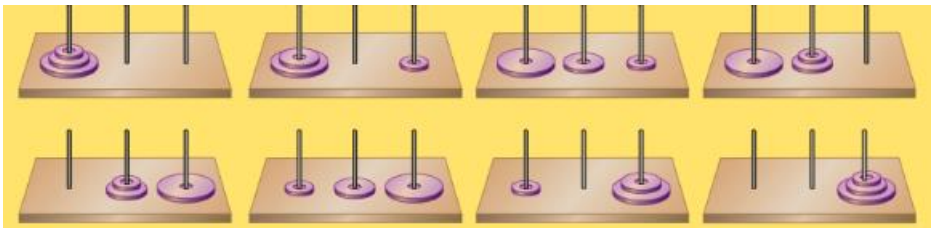
Historique :

Au XIX^{e} siècle, Édouard Lucas a inventé le jeu des « tours de Hanoï », une simple récréation mathématique qui s'est révélée au fil des années une mine de réflexions, comme il a été publier dans le tome 3 de ses Récréations mathématiques, parues à titre posthume en 1892 un de ses amis, N. Claus de Siam.

Lucas a écrit : "*N. Claus de Siam a vu, dans ses voyages pour la publication des écrits de l'illustre Fer-Fer-Tam-Tam, dans le grand temple de Bénarès, au-dessous du dôme qui marque le centre du monde, trois aiguilles de diamant, plantées dans une dalle d'airain, hautes d'une coudée et grosses comme le corps d'une abeille. 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 brahmes tomberont, et ce sera la fin des mondes !*"

Présentation du problème :

La configuration initiale du problème de la tour de Hanoi avec 3 piquets A, B et C et n disques disposés sur le piquet A du plus grand au plus petit et la configuration cible du problème dans laquelle les n disques apparaissent sur le piquet C. Il s'agit de déplacer la pile des
n disques se trouvant sur le piquet A de la configuration initiale vers le piquet C en respectant les
Contraintes suivantes :
- ne déplacer qu'un seul disque à la fois
- ne pas poser un disque sur un autre de plus petite taille
- se servir du piquet B pour des déplacements intermédiaires



Définition formelle du problème :

La Tour de Hanoï est un puzzle mathématique. Il se compose de trois pôles et d'un certain nombre de disques de différentes tailles qui peuvent glisser sur n'importe quel pôle. Le puzzle commence avec le disque dans une pile ordonnée par ordre croissant de taille dans un pôle, le plus petit en haut faisant ainsi une forme conique.

Présentation de la modélisation de la solution :

Pour la version itérative :
On utilise trois piles qui sont initialisé dans la procédure tohIterative avec l'état initiale (une pile source 'A' qui contient tous les disques et deux autres piles destination 'C' et auxiliaire 'B' qui sont vides)

Pour la version récursive :
Dans cette version on a utilisé les chaînes de caractères pour donner les étapes à suivre pour resoudre le problème de tour de hanoi

Présentation des algorithmes de résolution avec calcul détaillé de ses complexité théorique.

-Algorithme récursive :

```
Procédure Hanoi_récursive (n,depart,intermediaire,arrivée)
Debut
Si (n==1) alors nbDeplacement++  Fsi ;
Hanoi_recursive(n-1, depart, arrivee, intermediaire
Afficher("Deplacer le disque %d de %c vers %c\n", n, depart, arrivee);
nbDeplacement++;
Hanoi_recursive(n-1, intermediaire, depart, arrivee);
```

Explécation :

Départ, intermédiaire et arrivée représentent les 3 tours.
N est le nombre de disques à déplacer.
Le disque numéro n représente le plus grand disque se trouvant en bas de l'empilement dans la tour initiale.
Le principe de l'algorithme consiste à déplacer n-1 disques de « départ » vers « intermédiaire », puis le disque n de « départ » vers « arrivée » pour enfin mettre les n- 1 disques sur la tour « arrivée ».
Le déplacement des n-1 se fait en plusieurs étapes à travers les appels récursifs de la procédure, jusqu'à ce qu'il ne reste qu'un disque à déplacer.

-La complexité :

L'évaluation de la complexité de cet algorithme est assez simple.

- Déplacer les n-1 disques de la tour « départ » vers la tour « intermédiaire» nécessite $Dn-1$ déplacements.
- Déplacer le disque n (le plus grand) de la tour « départ » vers la tour « arrivée » se fait en une seule étape.
- Déplacer une seconde fois les n-1 disques de la tour « intermédiaire » vers la tour « départ » requiert également $Dn-1$ déplacements.

La Complexité de l'algorithme recursive c'est le nombre total de déplacements correspond donc à : $Dn = 2^n - 1$

La complexité est donc d'ordre exponentielle $O(2^n)$.

Algorithme itéative :

Explécation :

on déclare le nombre de disques et 3 piles vides dans le programme principale, puis on fait appelle a la procédure *toIterative* comme arguments le nombre de disques et les trois piles. dans la procédure *toIterative* on initialise les piles puis on boucle jusqu'a le nombre de mouvements qui est 2^n-1 et avec les valeurs du compteur de cette boucle modulo trois on effectue le déplacement adéquate selon le contenu des piles.

la démarche :

1. Calculez le nombre total de déplacements requis, c'est-à-dire 2^n - 1 telle que n est le nombre de disques.
2. Si le nombre de disques (n) est pair, alors échanger le piquet destination avec le piquet auxiliaire
3. pour i = 1 au nombre total de déplacements requis faire

si i%3 == 1 alors

déplacement du disque sommet du piquet source vers le piquet destination

si i%3 == 2 alors

déplacement du disque sommet du piquet source vers le piquet auxiliaire

si i%3 == 0 alors

déplacement du disque sommet du piquet auxiliaire vers le piquet destination

-La complexité est donc O(2^n)

Justification : dans la procédure *tohIterative* on empile les éléments de 1 jusqu'à le nombre de disque en utilisant une boucle donc complexité n. puis on utilise une boucle qui réalise les mouvements et qui boucle 2^n-1 fois alors une complexité 2^n. d'où T(n) = n+2^n donc la complexité est O(2^n)

Présentation de l'algorithme de vérification avec pseudo-code et calcul détaillé de sa complexité théorique :

il faut vérifier que la pile qui contient le résultat finale respect les règles du jeux. si on veut vérifier pour des résultats intermédiaire il faut juste supprimer les instructions de la variable count de l'algorithme

Pseudo code :

```
verify(dest: pile d'entier, number_of_disks : entier) : entier
DEBUT
   VAR verif,x,y,count: entier;
   Si(pilevide(dest)) Alors
     retourner(0);  // si pile vide donc retourner 0
   Fsi;
   verif = 0;                  // initialisation de la variable de la vérification
   x = dépiler(dest);                  // dépiler le premier élément de la pile
   count = 1;      // initialisation d'un compteur a 1 pour vérifié si la pile contient tous les élements
   TantQue(pilevide(dest)!=1 && verif) // tq pile n'est pas vide et verif est toujours vrai faire
   faire
     y = dépiler(dest);                // dépiler le prochaine élement de la pile
     Si(y<x) Alors          // si ce élement est petit a l'element dépiler precedement c-a-d pas de respect
                                  de la regle du jeux
          verif=0;     // mettre verif a faux
     Fsi;
     x = y;                          // preparation pour la prochaine iteration
     count++;                        // incrémenter le nombre d'elements
    fait;
   Si(count!=number_of_disks) Alors       // si le nombre d'élements dans la pile est plus petit que le
                                             nombre d'elements donc verif = 0
   verif=0;
   Fsi;
   retourner verif;
FIN
```

La complexité de l'algorithme de vérification est O(n).

Justification : pire cas est quand la solution est correct donc on dépile et on compare tous les éléments de la pile qui est égale a le nombre de disques d'où :

T(n)= 3 instructions d'affectation + n*(4 instructions de traitements) = 3+4*n donc la complexité est O(n)

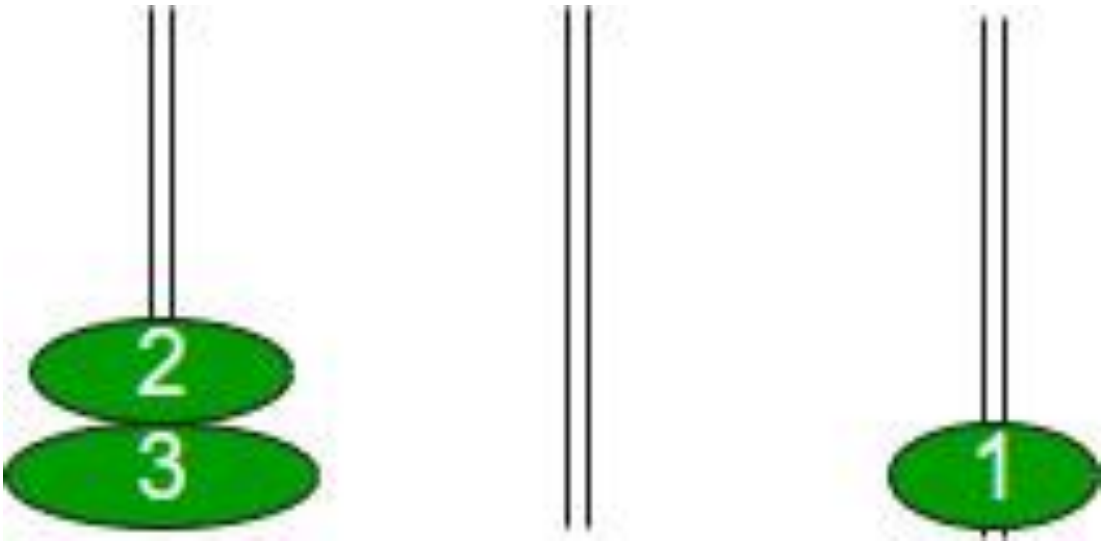
Présentation d'une instance du problème avec sa solution (un exemple) :

Soit un exemple avec nombre de disques = 3

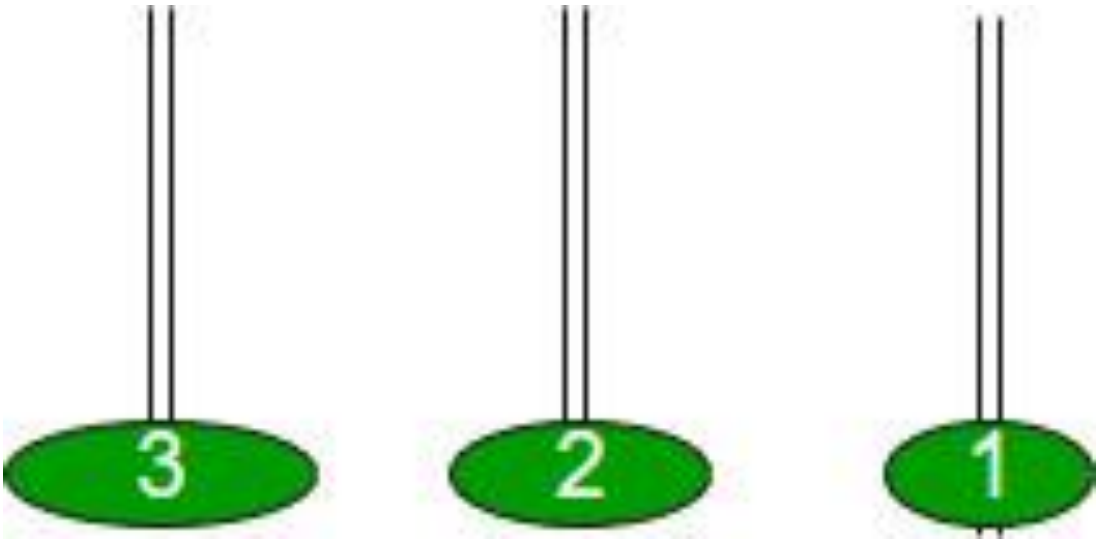
Donc le nombre total des mouvements requis est 2^3 – 1 = 8 – 1 = 7

Configuration initiale :

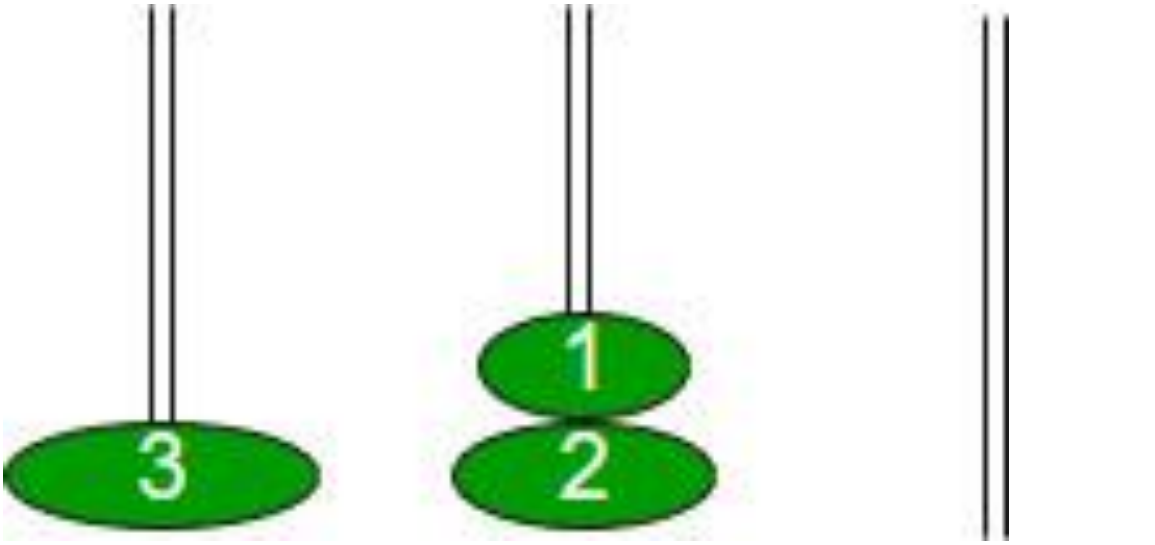
Au début : i = 1 => i % 3 == 1 donc déplacement du 'A' vers 'C'



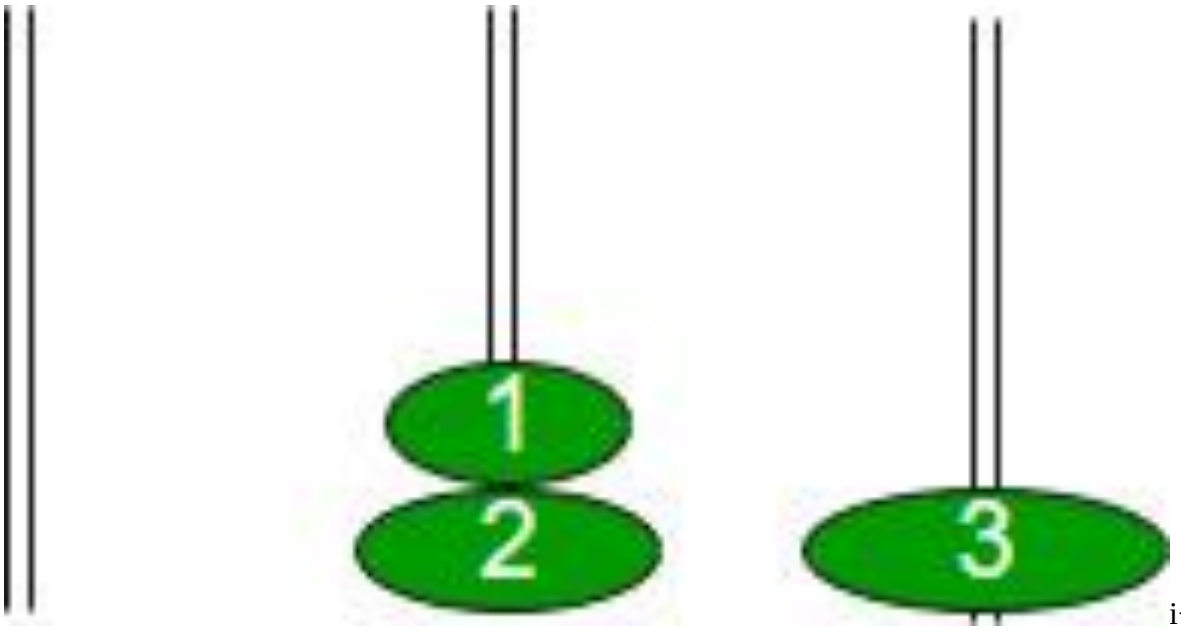
i++ => i = 2 => i % 3 == 2 alors déplacement du 'A' vers 'B'



i++ => i = 3 => i % 3 == 0 donc déplacement du ‘C’ vers ‘B’

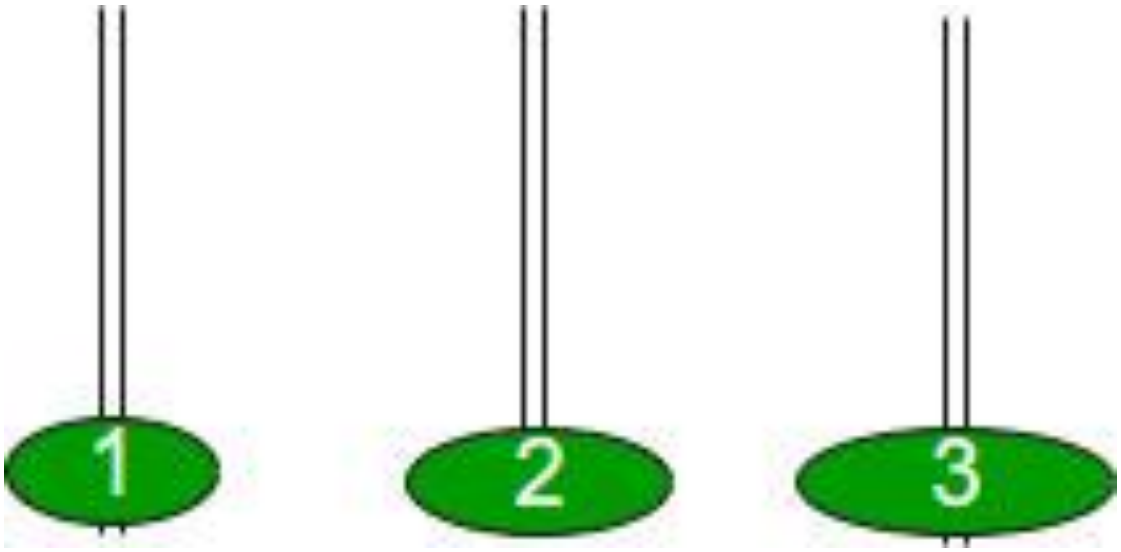


i++ => i = 4 => i % 3 == 1 alors déplacement du ‘A’ vers ‘C’



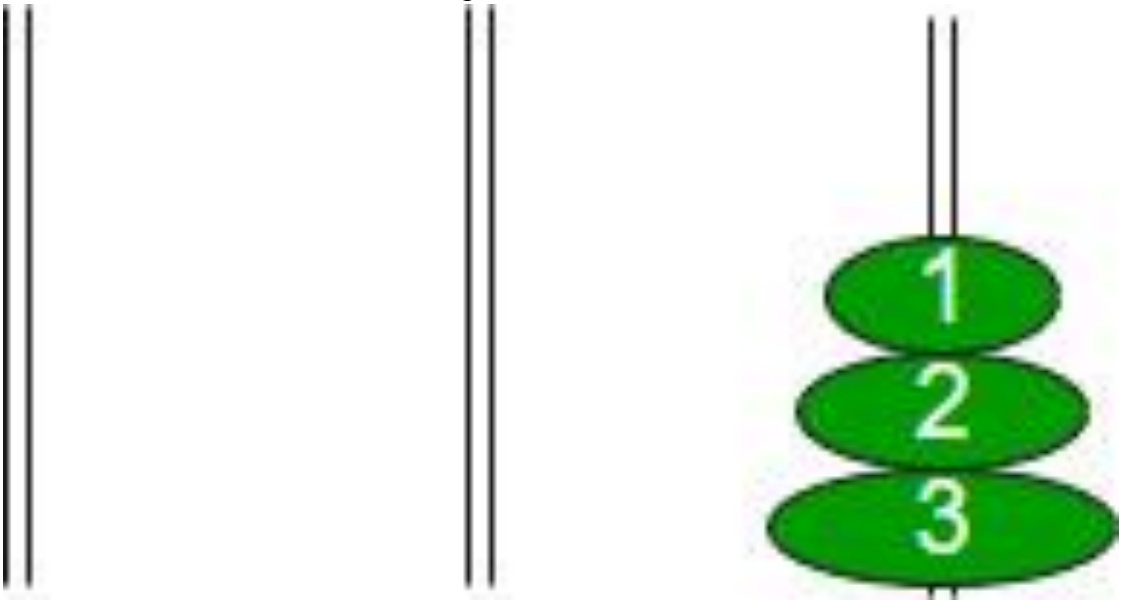
i++

=> i = 5 => i % 3 == 2 donc déplacement du ‘B’ vers ‘A’



i++ => i = 6 => i % 3 == 0 alors déplacement du ‘B’ vers ‘C’

i++ => i = 7 => i % 3 == 1 donc déplacement du ‘A’ vers ‘C’ Fin.



Etude Expérimentale :

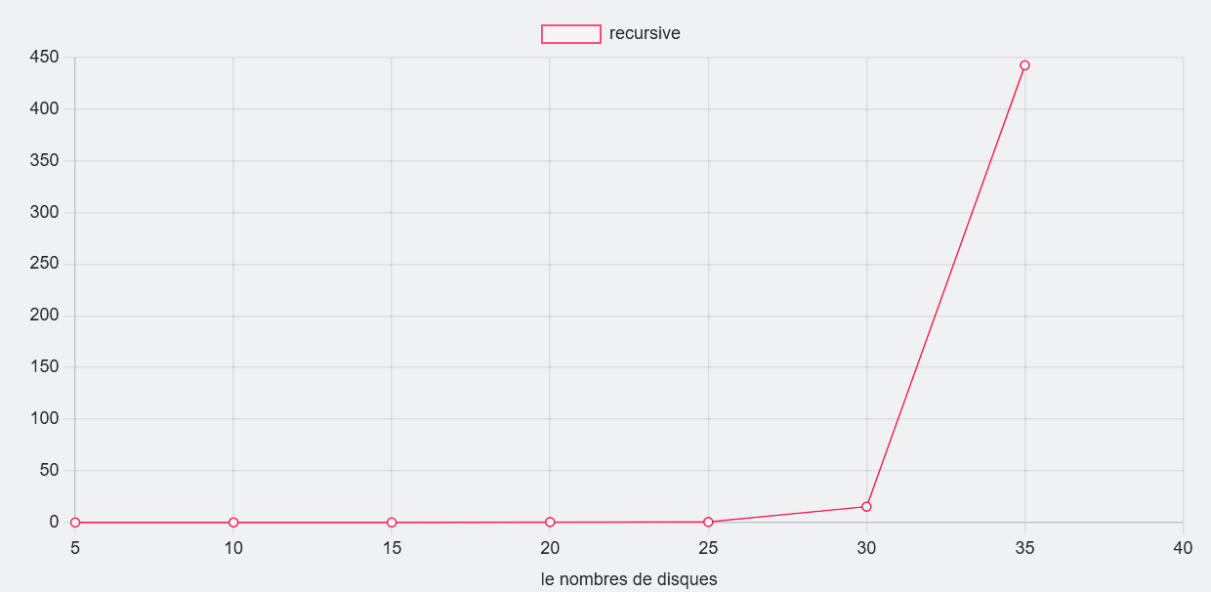
Environnent Expérimental

- *RAM: 8GO*
- *Processeur: Intel(R) CoreTM i5-8210M CPU @ 2.50 GHz.*
- *Windows 10-64 bits.*
- *Programming language: C.*
- *Environnement de développement intégré : Code-Blocks 16.*

La fonction de resolusion « récursive » :

Nombre de disque	5	15	20	25	30	35	40
recursive	0	0.001	0.039	0.429	15.253	442.531	+ de 20 min

Tableau : temps d'exécution de la fonction récursive en fonction du nombre de disques

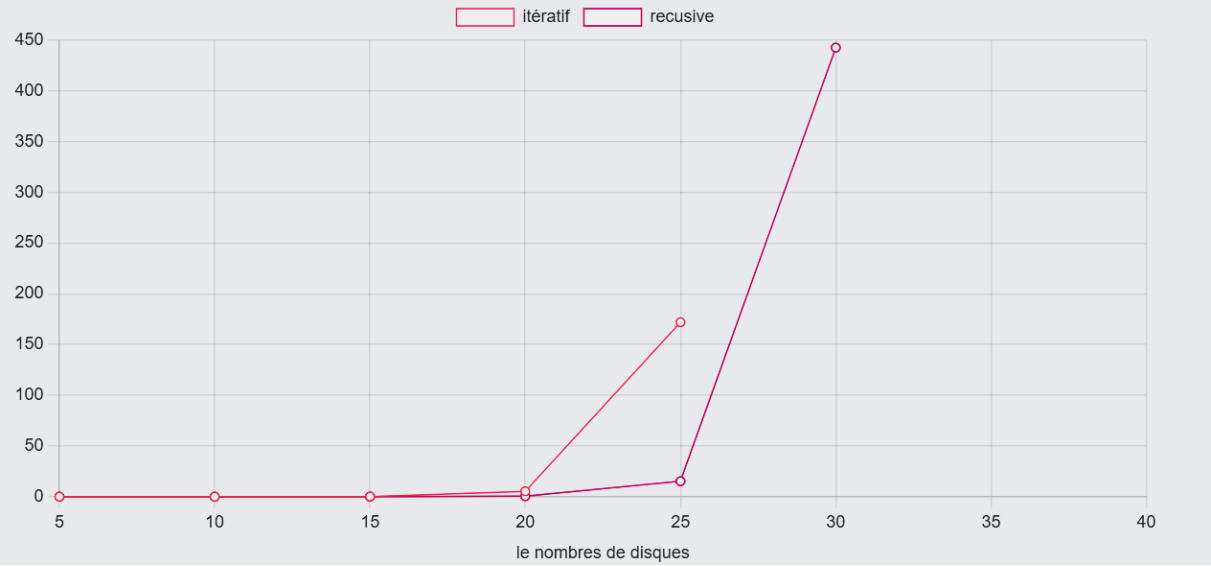


Graphe représentant les temps d'exécution de la fonction récursive en fonction du nombre de disques

- La fonction de resolusion « itirative » :

Nombre de disque	5	15	20	25	30	35	40
etirative		0.003	0.129	5.34	172.502		

Tableau : temps d'exécution de la fonction itérative en fonction du nombre de disques



Graphe représentant les temps d'exécution de la fonction itérative en fonction du nombre de disques

a. Algorithme de résolution :

Récursive :

- Complexité temporelle :

Soit le temps requis pour n disques est T(n) .
Il y a 2 appels récursifs pour n-1 disques et une opération à temps constant pour déplacer un disque du piquet 'départ' à 'arrivée'. Que ce soit k1.
Ainsi,
$T(n) = 2\ T(n-1) + k1$
$T(0) = k_2$, a constant.
$T(1) = 2\ k_2 + k_1$
$T(2) = 4\ k_2 + 2k_1 + k_1$
$T(2) = 8\ k_2 + 4k_1 + 2k_1 + k_1$
Coefficient of $k_1 = 2^n$
Coefficient of $k_2 = 2^n - 1$
Donc la complexité temporelle est $O(2^n)$.

-Complexité spatiale :
L'espace pour le paramètre de chaque appel est indépendant de n, c'est-à-dire constant. Soit k. Lorsque nous effectuons le 2ème appel récursif, le 1er appel récursif est terminé. Ainsi, nous pouvons réutiliser l'espace du 1er appel pour le 2ème appel. D'où ,
T(n) = T(n-1) + k
T(0) = k
T(1) = 2k
T(2) = 3k
T(3) = 4k
Donc la complexité spatiale est O(n)

Itérative :

-Complexité temporelle :

identique à l'approche récursive car l'idée de base et la logique sont les mêmes

-Complexité spatiale :O(n)
Car on a utilisé 3 piles de taille égale à n st d'autres variables constantes comme le nombre de disques donc O(1)

Donc la complexité spatiale est 3n+1 ce qui donne O(n)

b. Algorithme de vérification :

- Complexité temporelle : O(n)

-Complexité spatiale :O(n)
Car on a utilisé une piles en entrée qui contient n élément au maximum ce qui donne O(n)

c. Le meilleur, moyen et pire cas pour chaque algorithme :

Algorithme de résolution : quel que soit le nombre de disques on fait toujours 2^n – 1 itération donc la complexité est toujours O(2^n)
Algorithme de verification :
meilleur cas est quand la pile est vide donc complexité O(1)
pire cas est quand la pile est plein donc complexité O(n)
moyen cas si k est le nombre d'élement du pile donc complexité est O(k)

d .Analyse des résultats :

d'après les résultat de calcule de complexité des deux version itérative et récursive on a remarqué que la complexité temporelle est exponentielle mais la complexité spatiale est linéaire. Il y a souvent un compromis entre la complexité temporelle et spatiale.

Conclusion :

D'après l'étude faite et ses résultats on a obtenu une complexité exponentielle pour les deux versions de l'algorithme récursive et itérative ce qui implique que le temps nécessaire pour résoudre un problème et trop long en pratique ,dans ce cas on va pas pouvoir trouver une solution exacte mais seulement une solution approchée en utilisant des méthodes heuristiques.

la **distribution des tâches :**

RAPPORT fait par : ABDOU Maria hind
LES TESTES fait par : FERRADJI Tarek
LA VERSION ITERATIVE faite par : YOUSFI Zakaria
LA VERSION RECURSIVE faite par : MOUAZ Rayane Hamza

Références :

https://www.stechies.com/tower-of-hanoi-c/

https://www.geeksforgeeks.org/iterative-tower-of-hanoi/