Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département IA et SD

Rapport travaux pratiques n° 3

Module : Conception et Complexité des Algorithmes

Tours de Hanoï

Travail présenté par :

- **BENKOUITEN Aymen———-191931046409**
- **MALKI Omar Chouaab———-191931081333**
- **KENAI Imad Eddine————-191932017671**
- **MEKKAOUI Mohamed———-191931081338**

2022/2023

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Le problème des tours de Hanoï est un jeu faisant parti de la catégorie des casse-têtes. Il est très exploité dans la recherche en psychologie de la résolution de problème, employé comme épreuve lors d'une évaluation neuropsychologique des fonctions exécutives, étudié en mathématiques et souvent utilisé en algorithmique pour montrer la puissance et l'intérêt de la récursivité. **Et donc on va présenter une solution algorithmique a ce problème**.

L'objectif dans ce TP est de mettre en pratique et tester et calculer la complexité de l'algorithme de résolution du problème des "tours de Hanoi", qui est un problème classique en informatique.

Première partie

Développement de l'algorithme en deux version en langage C.

0.1 Implémenter l'algorithme de résolution de la tour de hanoï en version récursive et itérative en langage C :

Le code source de **"l'algorithme de résolution de la tour de Hanoï en version récursive et itérative en langage C et l'algorithme de vérification"** est bien implémenter dans la partie Annexe.

Deuxième partie

Etude théorique du problème.

0.2 Historique et présentation du problème :

Ce problème mathématique a été inventé par le mathématicien Édouard Lucas (1842-1891), qu'il devrait d'après ce qu'il a publié 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 !"***.[4]

0.3 Définition formelle du problème :

Le jeu comporte trois piquets et une série de disques troués en leur milieu de sorte qu'on puisse les enfiler sur n'importe lequel des trois piquets. Les disques sont de tailles toutes différentes et, au départ, ils sont tous enfilés sur un même piquet en respectant la règle de base : un disque ne peut reposer que sur un disque de taille plus grande que la sienne. Le but du jeu est de réussir à enfiler tous les disques sur l'un des deux autres piquets en ne déplaçant à chaque mouvement qu'un seul disque pris au sommet de la pile de l'un des piquets pour l'enfiler sur l'un des deux autres piquets en sommet de sa pile de disques, tout en respectant à chaque mouvement la règle de base.[2]

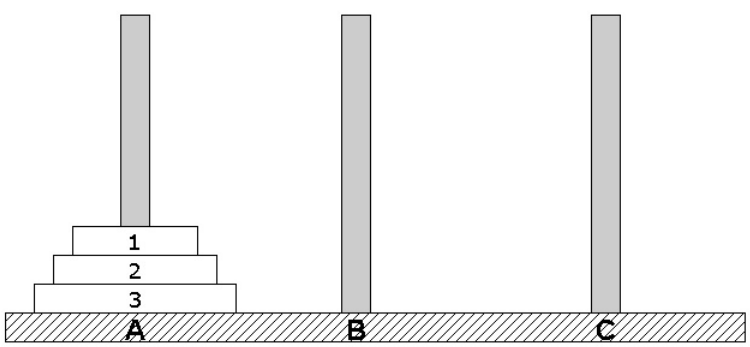

FIGURE 1 – Les tours de Hanoi

0.4 présentation de la modélisation de la solution et l'algorithme de résolution avec le calcul détaillé de la complexité théorique

Algorithme récursive de la tours de Hanoï :

Algorithme récursif : Le raisonnement de la resolution recursive consiste à regrouper les n-1 premiers disques sur l'emplacement intermédiaire I (celui qui n'est ni A ni Pn) et après déplacer le plus grand disque de Pn vers A et a la fin regrouper les n-1 premiers disques en A.[1]

Pseudo code :

```


        procedure hanoi_rec(A,B,C :caractere , n :entiers)

        Debut

          si (n != 0 ) alors
            hanoi_rec( A , C , B , n-1 );
            A <- C ;
            hanoi_rec( B , A , C , n-1 );
          fsi;

        fin;


```

Listing 1 – pseudo code de la version recursive.

Complexité temporelle et spatiale de l'algorithme :

Complexité temporelle CT(n) :
Notons CT(n) la complexité temporelle de l'algorithm de Hanoi

- Au meilleur cas pour 1 disque on a donc
  n = 1 d'ou :
  CT(1) = CT(0)+O(1)+CT(0) // on appel hanoi sur n-1 disque 2 fois
  CT(1) = 2*CT(0) + O(1)
  Ou CT(0) = O(1).

- D'ou au meilleur cas on a :
  CT(n) = O(1) // avec n = 1.

- Au pire cas on a :
  CT(n) = CT(n-1) + 1 + CT(n-1)
  CT(n) = 2*CT(n-1) + 1

  On remarque que c'est une suite numérique :
  CT(n) = 2*CT(n-1) + 1 - (1)

  On a donc : CT(n)-1 = 2*CT(n-2) + 1 - (2) CT(n)-2 = 2*CT(n-3) + 1 - (3)

  En remplace (3) dans (2) et (2) dans (1) on obtient :
  $CT(n) = 2 * (2 * (2 * CT(n-3) + 1) + 1) + 1CT(n) = 2^3 * CT(n-3) + 2^2 + 2^1 + 2^0$

  On peut généraliser à :
  $CT(n) = 2^k * CT(n-k) + 2^k - 1 + ... + 2^0$

  Sachant que k = n-1
  $CT(n) = 2^{(n-1)} * CT(n-(n-1)) + 2^n - 2 + ...2^0$
  $CT(n) = 2^{(n-1)} * CT(1) + 2^n - 2 + ...2^0$

  Avec : CT(1) = O(1)

  On remarque donc facilement que l'expression croit d'une manière exponentielle, d'ou :
  $CT(n) = O(2^n)$.

Complexité spatiale CS(n) :
Notons CS(n) la complexité spatiale de l'algorithme.
On a donc au total environs 2^n appel selon la question précedente.
On peut aussi reutiliser les mêmes paramètres de chaque appel recursifs.
En clair on a : CS(n) = $2^n * n/2^n$ D'ou : $CS(n) = n = O(n)$.

Algorithme itérative de la tours de Hanoï :

Algorithme itératif : Le raisonnement de la resolution itérative :
Pour cette version, on va représenter les tours en les « PILES » car c'est la structure de données la plus adéquate, Lorsque on déroule pour des valeurs différentes de N (nombres de disques) on remarque qu'il y a deux séquences de déplacement [3] qui se répètent :

- Si N est paire :
  - Déplacer entre A et B.
  - Déplacer entre A et C.
  - Déplacer entre B et C.
- Si N est impaire :
  - Déplacer entre A et C.
  - Déplacer entre A et B.
  - Déplacer entre B et C.

Remarque : les déplacements sont effectués dans les deux sens. Pour simplifier l'algorithme, si N est paire on permute entre les deux tours B et C et on va avoir que 03 déplacements qui vont être exécutées.

- A <—> C.
- A <—> B.
- B <—> C.

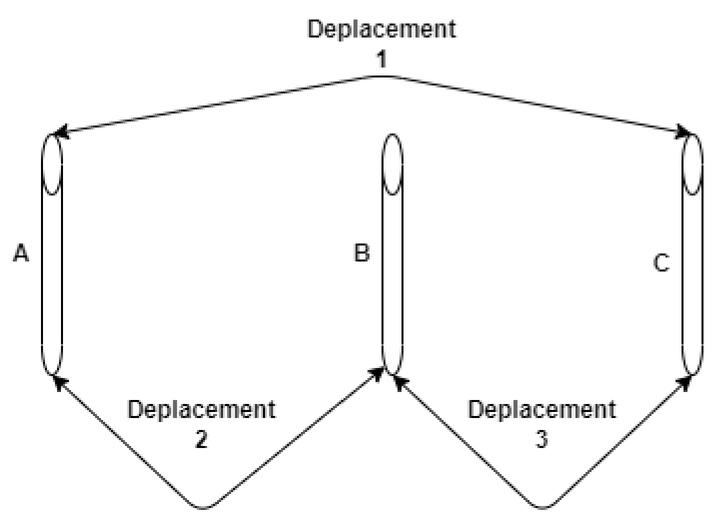

FIGURE 2 – Schéma de deplacement

Pseudo code :

```
procedure deplacer(source ,destination :Pile)

 debut
          x = depiler(source);
          y = depiler(destination);

         // si source est vide
         si (x==nil) alors emplier(source, y)
         // si destination est vide
         sinon si (y==nil)  empiler(destination, x)

               sinon si (x > y) alors

                           empiler(source, x);
                           empiler(source, y);



                       sinon

                           empiler(destination, y);
                           empiler(destination, x);

                    fsi;
              fsi;
         fsi;
 fin;

 procedure hanoi_iter(A,B,C :Pile , n :entiers)
 Debut
    nbr_deplacement = 2^n -1;

    si (n % 2 == 0 ) alors permuter (B,C);

    pour i de 1 a nbr_deplacment:

      faire
             si  (i % 3 == 1) alors
                    deplacer(A, C)

             sinon  si (i % 3 == 2) alors
                     deplacer(A, B)

                    sinon si(i % 3 == 0) alors
                           deplacer(B, C);
                    fsi;
             fsi;

      fait;

 fin;
```

Listing 2 – pseudo code de la version itérative.

Complexité temporelle et spatiale de l'algorithme :

Complexité temporelle CT(n) :
On va exécuter la procédure deplacer() de 1 .. (nbr_deplacement).
Et nbr_deplacement = 2^n - 1.
Alors la complexité temporelle est de CT(n) = O(2^n).

Complexité temporelle CS(n) :
La complexité spatiale est de O(n) (n est le nombre de disques) car on va utiliser 3 piles de taille = n chacune. CS(n) = O(n).

0.5 présentation de l'algorithme de vérification avec le calcul détaillé de la complexité théorique

Algorithme de vérification

Pseudo code :

```
Algorithme verification

 Entree => P :Pile qui represente le tour C
 sortie => Booleen
 Var  d1 , d2 :Entier;

debut

    Depiler (P,d1);
    tant que (non pilevide(P))
    faire
         Depiler (P,d2);

         si (d1>d2) alors Retourner  faux;

         d1=d2;
    fait;

    retourner vrai;

fin;
```

Listing 3 – pseudo code de la version recursive.

Complexité temporelle et spatiale de l'algorithme :

l'algorithme de verification va parcourir toute la pile qui continent l'état finale des disques, donc on peut dire que sa complexité est de O(n) -> complexité polynomiale

0.6 présentation d'une instance du problème avec sa solution :

Déroulement de l'algorithme :

Déroulement sur une tour de Hanoï avec le nombre des disques égale à 3, en montrant à chaque déplacement l'état de la pile (Figure - 3) :

Nombre de Deplacment	Image	Etat de la Pile Pile 1	Pile 2	Pile 3
Etat initiale		DISQUE1 DISQUE2 DISQUE3		
n=1		DISQUE2 DISQUE3		DISQUE1
n=2		DISQUE3	DISQUE2	DISQUE1
n=3		DISQUE3	DISQUE1 DISQUE2	
n=4			DISQUE1 DISQUE2	DISQUE3
n=5		DISQUE1	DISQUE2	DISQUE3
n=6		DISQUE1		DISQUE2 DISQUE3
n=7				DISQUE1 DISQUE2 DISQUE3

FIGURE 3 – Déroulement et état de la pile

Troisième partie

Etude Expérimentale.

0.7 Mesure du temps d'exécution :

Version récursive :

Représentation tabulaire :

Nombre de disque	5	10	15	20	25	30	35	40
Temps d'exécution	0 s	0 s	0 s	0.001 s	0.077 s	2.217 s	79.45 s	1310.03 s
Nombre de déplacements effectués	31	1023	32767	1048575	33554431	1073741823	34359738367	1099511627775

Table 1 – Temps d'exécution de l'algorithme de résolution de probleme de la tour de Hanoï par une variation en nombre de disque -version récursive-

Représentation graphique :

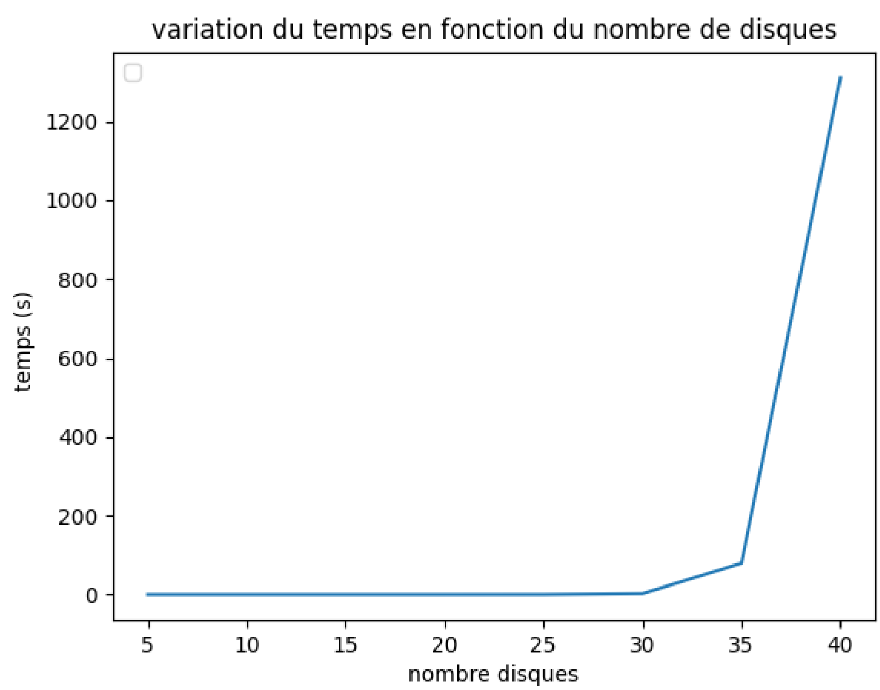

Figure 4 – Représentation graphique des résultats d'exécution de l'algorithme de résolution par une variation en nombre de disque -version récursive-

Analyse des résultats :

Il est clair que le graphe obtenu [Figure 4] a le même comportement qu'une fonction exponentielle.
Pour un nombre de disques inférieur à 40 l'algorithme s'exécute en un temps raisonnable. Mais au-delà de 45 disques, le temps s'accroit très rapidement (de façon exponentielle) et passe de mille et quelques centaines de secondes pour 40 disques et à approximativement 6h, pour 45 disques.

Version itérative :

Représentation tabulaire :

Nombre de disque	5	10	15	20	25	30	35
Temps d'exécution	0 s	0 s	0.002 s	0.07 s	1.751 s	34.388 s	1124.358 s
Nombre de déplacements effectués	31	1023	32767	1048575	33554431	1073741823	34359738367

TABLE 2 – Temps d'éxécution de l'algorithme de résolution de probleme de la tour de Hanoï par une variation en nombre de disque -version iterative-

Représentation graphique :

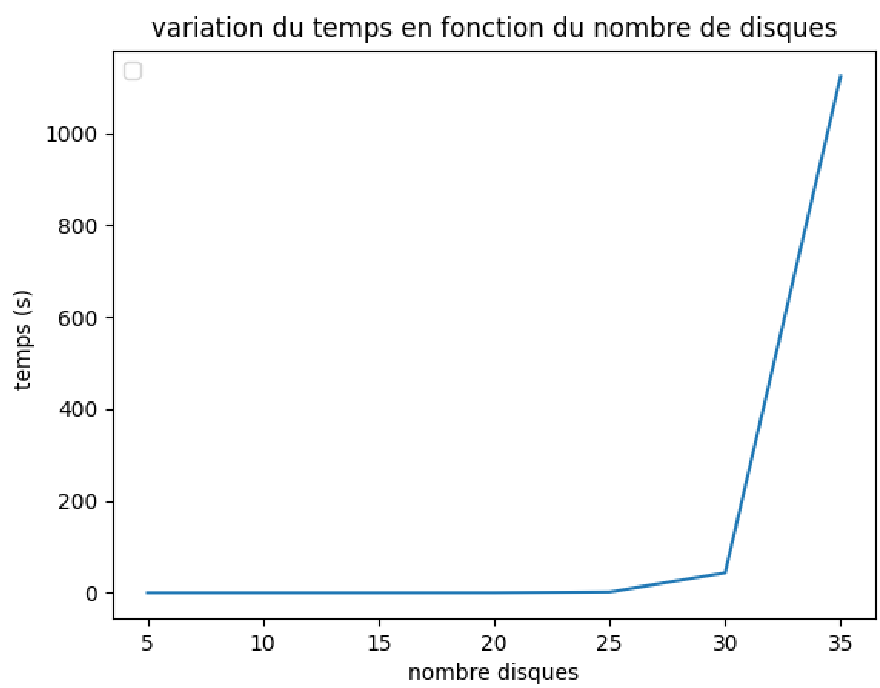

FIGURE 5 – Représentation graphique des résultats d'exécution de l'algorithme de résolution par une variation en nombre de disque -version itérative-

Analyse des résultats :

Il est clair que le graphe obtenu [Figure 5] a le même comportement qu'une fonction exponentielle.
Pour un nombre de disques inférieur à 35 l'algorithme s'exécute en un temps raisonnable. Mais au-delà de 40 disques, le temps s'accroit très rapidement (de façon exponentielle) et passe de mille et quelques centaines de secondes pour 35 disques.

Raisonnement :

Ce problème, peut devenir très vite complexe. En effet, pour n disques le nombre de déplacements nécessaires est au minimum de $2^n - 1$. Le déplacement des disques nécessite donc deux fois plus de temps à chaque fois qu'un disque est ajouté à la tour initiale.
Si on considère l'exemple donné avec 64 disques, il faudra exécuter la série de 2^{64}-1 = 18 446 744 073 709 551 615 déplacements, En admettant qu'il faille une seconde pour déplacer un disque on obtient 86 400 déplacements par jour et donc le jeu se termine après 584,5 milliards d'années approximativement, soit une quarantaine de fois l'âge de l'univers.

Remarque :

L'éxecution de l'algorithme de résolution en version récursive et itérative a été bien réaliser par une machine pour chaque version (récursive par une machine et itérative par une autre).

0.8 simulation de la complexité temporelle et spatiale théorique de l'algorithme de résolution :

Version récursive :

La CT (n) = $O(2^n)$, elle croit d'une manière exponentielle. La CS (n) = O(n).

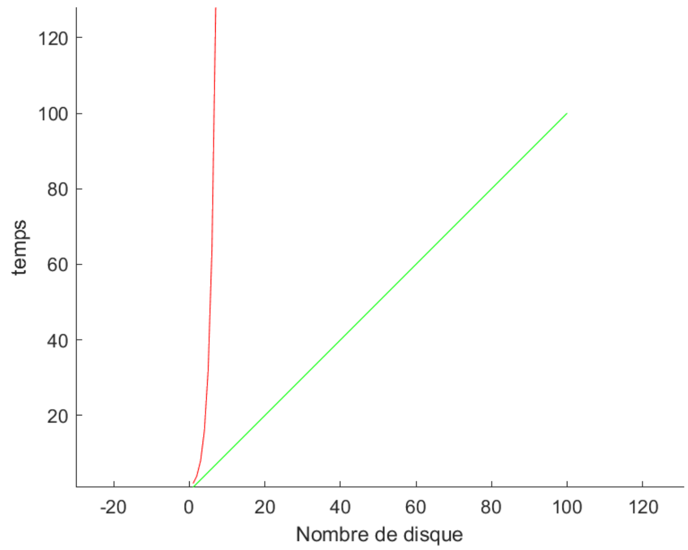

FIGURE 6 – Représentation graphique de la complexité temporelle et spatiale théorique de l'algorithme de résolution en version récursive.

Version itérative :

La CT (n) = $O(2^n)$, elle croit d'une manière exponentielle. La CS (n) = O(n).

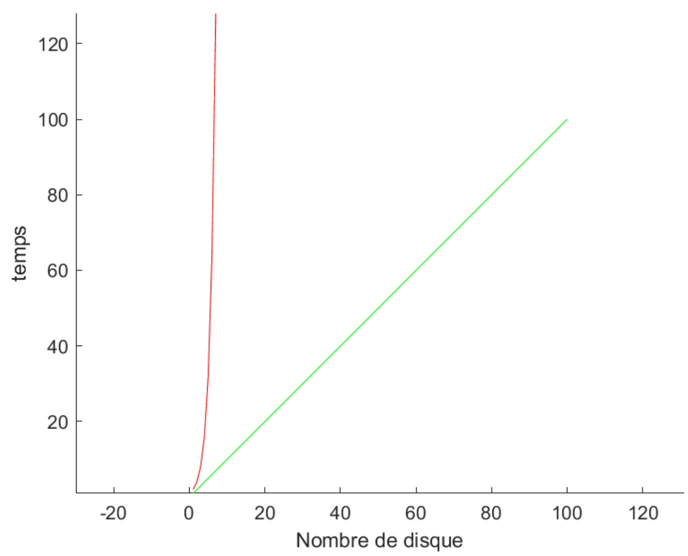

FIGURE 7 – Représentation graphique de la complexité temporelle et spatiale théorique de l'algorithme de résolution en version itérative.

0.9 simulation de la complexité temporelle et spatiale théorique de l'algorithme de vérification :

Représentation graphique :

l'algorithme de vérification va parcourir toute la pile qui continent l'état finale des disques, donc on peut dire que sa complexité est de O(n) -> complexité polynomiale

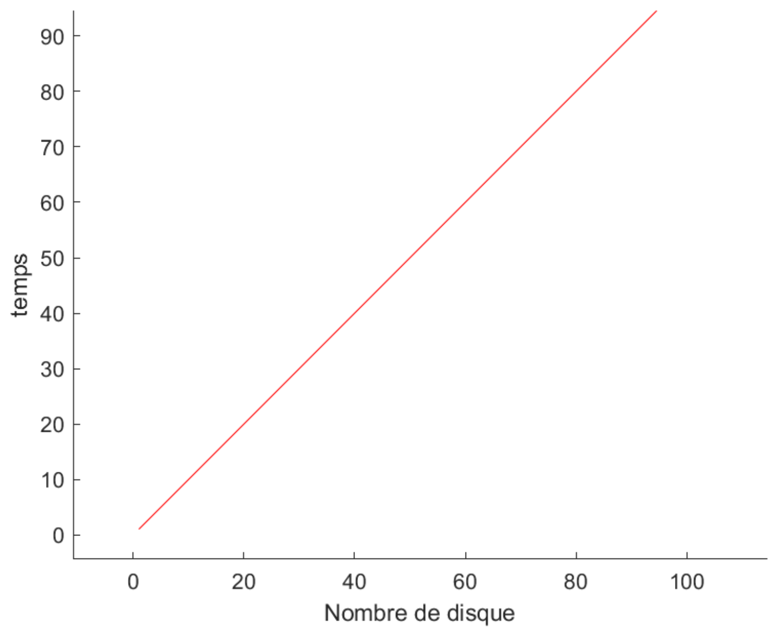

FIGURE 8 – Représentation graphique de la complexité temporelle et spatiale théorique de l'algorithme de vérification

0.10 Le meilleur et moyen et pire cas pour chaque algorithme :

Algorithme de résolution :

Dans notre cas on a toujours la même situation « des disques à déplacer », la seule chose qui varie c'est le nombre de disques, sinon c'est les mêmes déplacements qui seront répétés. Donc on peut dire qu'on a qu'un seul cas à étudier (On boucle toujours $2^n - 1 fois$).

Algorithme de vérification :

Le pire cas c'est lorsque la solution est juste (cad : la pile est ordonnée de plus grand au plus petit).
Le meilleur cas c'est lorsque le disque qui se trouve au sommet est plus grand que le suivant.

Conclusion Générale

Les résultats de notre analyse des résultats obtenus ainsi que le calcul théorique de la complexité temporelle des deux algorithmes de résolution (en deux versions) et de vérification, nous a permis de comparer l'efficacité des algorithmes de resolutions (version récursive et itérative) et établir lequel d'entre eux est le optimal et le plus rapide.

Les résultats obtenus prouvent que avec n'importe quelle solution soit récursive ou itérative, les résultats d'éxecution sont similaires car elles ont la meme complexité temporelle CT(n) = $O(2^n)$ qui s'accroit de façon exponentielle et donc même si la solution récursive est plus facile à comprendre et à écrire, elle n'est pas toujours optimale et diminue pas pour autant la complexité de tous les problèmes qu'on cherche à résoudre. De plus la version itérative n'utilise aucune mémorisation de la série des déplacement, et nécessite de se souvenir où on doit placer le disque.

Environnement expérimental

0.11 Caractéristiques des machines utilisés :

Caractéristiques de la machine	Système d'exploitation	CPU	RAM	Version compilateur C
BENKOUITEN Aymen	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz	8,00 GO	codeblocks-20.03mingw
KENAI Imad Eddine	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz	16,00 GO	codeblocks-8.1.0mingw
MALKI Omar Chouaab	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz	8,00 GO	codeblocks-20.03mingw
MEKKAOUI Mohamed	Système d'exploitation 64 bits, processeur x64. Windows 11 Home V-22H2.	Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz	16,00 GO	codeblocks-20.03mingw

TABLE 3 – Environnement expérimental

0.12 Répartition des tâches :

- Écriture du code source et le calcul de la complexité : KENAI + MALKI.
- Exécution des taches : MEKKAOUI.
- Représentation graphique : MEKKAOUI + KENAI.
- Analyse des résultats : BENKOUITEN.
- La rédaction du rapport et son organisation avec Latex : BENKOUITEN + MALKI.
- NB : Tous ce travail est fait durant un meet entre le team et toutes les réponses et les analyses effectuer sur les graphes sont bien été discuter avant la rédaction de rapport.

Annexe

0.13 Code source des algorithmes :

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

//element pile
struct Stack{
    unsigned size;
    int top;
    int *array;
}

struct Stack* init_pile(unsigned size){
    struct Stack* stack =
        (struct Stack*) malloc(sizeof(struct Stack));
    stack -> size = size;
    stack -> top = -1;
    stack -> array =
        (int*) malloc(stack -> size * sizeof(int));
    return stack;
}

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

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

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

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

//afficher deplacement
void afficher_deplacement(char fromPeg, char toPeg, int disk){
    printf("deplacement %d de  \'%c\' vers \'%c\'\n", disk, fromPeg, toPeg);
}

//deplacer disk entre deux piles
void deplacer_entre_deux_tours(struct Stack *source, struct Stack *destination, char s, char d){
    int pile1 = depiler(source);
    int pile2 = depiler(destination);

```

```
    // si source est vide
    if (pile1 == INT_MIN){
        empiler(source, pile2);
        // afficher_deplacement(d, s, pile2);
    }

    // si destination est vide
    else if (pile2 == INT_MIN){
        empiler(destination, pile1);
        // afficher_deplacement(s, d, pile1);
    }

    // deplacement vers pile destination n'est pas permis
    else if (pile1 > pile2){
        empiler(source, pile1);
        empiler(source, pile2);
        // afficher_deplacement(d, s, pile2);
    }

    // deplacement vers pile destination
    else{
        empiler(destination, pile2);
        empiler(destination, pile1);
        // afficher_deplacement(s, d, pile1);
    }
}

//algo de resolution (recursive) --------------
void hanoi_recu(int n, char D, char A, char I){
    if(n == 1) printf("Disque 1 de %c \n", D ,A);
    else{
        //D a A
        hanoi_recu(n-1, D, I, A);
        printf("Disque %d de %c a %c \n", n, D, A);
        //I a A
        hanoi_recu(n-1, I, A, D);
    }
}

//algo de resolution (iterative) --------------
void hanoi_iter(int num_of_disks, struct Stack *A, struct Stack *B, struct Stack *C){
    unsigned long long i,nbr_deplacements;
    char s = 'A', d = 'C', a = 'B'; //s:source   d:destination  , a: intermediere

    //si nombre de disque est paire
    if (num_of_disks % 2 == 0){
        char temp = d;
        d = a;
        a = temp;
    }
    nbr_deplacements = pow(2, num_of_disks) - 1;

    //empilement des disques dans la premiere pile
    for (i = num_of_disks; i >= 1; i--){
```

```c
        empiler(A, i);
    }
    for (i = 1; i <= nbr_deplacements; i++){
        if (i % 3 == 1)
        deplacer_entre_deux_tours(A, C, s, d);

        else if (i % 3 == 2)
        deplacer_entre_deux_tours(A, B, s, a);

        else if (i % 3 == 0)
        deplacer_entre_deux_tours(B, C, a, d);
    }
}

//fonction de verification --------------
int verification(struct Stack *C){
    int d1,d2;
    d1=depiler(C);
    while(!pile_vide(C)){
        d2=depiler(C);
        if(d1>d2) return 0;
        d1=d2;
        }
    return 1;
}

void time_complexity(int debut ,int fin ){
    struct Stack *A, *B, *C;
    time_t t1,t2;
    double t=0;

    for (int i = debut ; i <= fin ; i+=5){
        A = init_pile(i);
        B = init_pile(i);
        C = init_pile(i);

        t1=clock();
        hanoi_iter(i ,A ,B ,C);
        t2=clock();

        t= (float)(t2-t1)/CLOCKS_PER_SEC;

        printf("nombre de disques = %d   -->  %f secondes\n\n",i,t);

        printf(verification(C)? "algorithme verifiee" : "algorithme non verifiee");

        free(A);
        free(B);
        free(C);

    }

}


```

```c
//main fonction
int main()
{
    time_complexity(5,35);
    return 0;
}

```

Listing 4 – code source en C.

Bibliographie

[1] definition tour de hanoi. https://fr.wikipedia.org/wiki/Tours_de_Hano%C3%AF.

[2] Déf problème de tour de hanoi. https://www.pousseurdebois.fr/les-tours-de-hanoi/.

[3] Etat de déplacement. https://construire-des-savoirs.fr/?p=1036.

[4] histoire de tour de hanoi. https://jeux-casse-tete.com/blog/regles-de-jeux/regle-du-jeu-la-tour-de-hanoi.