

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Electronique et d'Informatique
Département Informatique

Master Systèmes Informatiques intelligents

Module : Conception et Complexité des Algorithmes

Rapport de projet 2 de TP

Réalisé par :

AIT AMARA Mohamed, 181831072170

BOUROUINA Rania, 181831052716

CHIBANE Ilies, 181831072041

HAMMAL Ayoub, 181831048403

Année universitaire : 2021 / 2022

Table des matières

1	Présentation du la solution	2
1.1	Modélisation de la solution	2
1.2	Algorithme de résolution	3
1.3	Algorithme de vérification	5
1.4	Illustration d'une instance du problème	6
2	Étude expérimentale	8
2.1	Complexité théorique de l'algorithme de résolution	8
2.2	Complexité théorique de l'algorithme de vérification	9
2.3	Expérimentation	10
2.3.1	Algorithme de résolution	10
2.3.2	Algorithme de génération de solution	11
2.3.3	Algorithme de vérification de solution	13

Chapitre 1

Présentation du la solution

1.1 Modélisation de la solution

Dans ce problème, chaque anneau porte un numéro séquentiel unique $a_i \in [1, n]$ qui représente sa taille tel que n est le nombre maximal d'anneaux (e.g. l'anneau avec le nombre 1 est plus petit que l'anneau avec le nombre 3).

De plus, on modélise chaque tour par un tableau T_j d'une taille égale au nombre maximum d'anneaux n . Si un niveau i de la tour j contient un anneau $a_{i'}$, $T[j, i] = a_{i'}$, sinon $T[j, i] = 0$. Le niveau le plus bas de la tour (la base de la tour) est placé à la dernière case du tableau ; $\forall j T[j, n]$ est le niveau le plus bas de la tour (voir Figure 1.1).

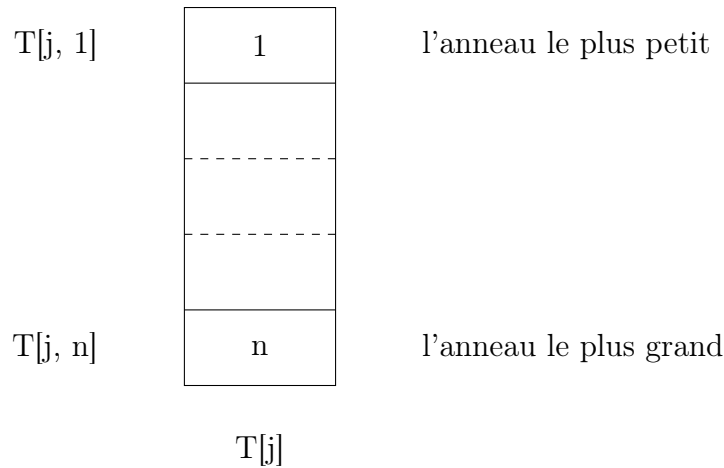


FIGURE 1.1 – Exemple d'une tour avec tous les anneaux

Par conséquent, le bord de jeu peut être représenté par une matrice colonne par colonne ou chaque colonne est en réalité une tour du jeu.

$$\mathbf{bord} = \begin{pmatrix} T[1, 1] & T[2, 1] & \dots \\ \dots & & \\ T[1, n] & T[2, n] & \dots \end{pmatrix}$$

Un exemple d'initialisation classique de trois tours avec trois anneaux placés sur la première tour :

$$\mathbf{bord} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}$$

Règle de changement d'état Pour passer d'un état de bord vers le suivant, on ne peut bouger qu'un seul anneau du haut d'une tour vers une autre tour, à condition que l'anneau supérieur de la tour destination a un nombre supérieur à l'anneau qu'on veut bouger.

Plus formellement, on peut déplacer le premier élément non nul d'une colonne s'il existe vers une autre colonne, s'il y a encore de la place et que le premier élément non nul de la colonne destination est supérieur à celui qu'on déplace.

1.2 Algorithme de résolution

Cet algorithme récursif permet de produire la séquence exacte d'actions pour résoudre le problème des tours de Hanoi.

On commence d'abord par déplacer les $n - 1$ disques de la tour de départ vers la tour intermédiaire par un appel récursif. Puis, le plus grand disque restant est transporté vers la tour d'arrivée. Ensuite, les $n - 1$ disques qui se trouvaient sur la tour intermédiaire sont déplacés vers la tour d'arrivée par le même processus récursif.

Algorithme 1 : Hanoi

Données : bord : matrice $[1, 3][1, n]$ d'entiers, depart, arrivee, intermediaire : 1..3,
nbdisques : entier

Résultat : bord : matrice $[1, 3][1, n]$ d'entiers

début

```
    si nbdisques = 1 alors
        // S'il reste un seul disque à déplacer, on le déplace directement
        deplacer(board, depart, arrivee)
    sinon
        Hanoi(bord, depart, intermediaire, arrivee, nbdisques - 1);
        deplacer(board, depart, arrivee);      // Déplacer le disque supérieur de la
        tour depart vers la tour arrivee
        Hanoi(bord, intermediaire, arrivee, depart, nbdisques - 1);
    fin si
```

fin

La fonction *deplacer* permet de déplacer le disque de niveau supérieur d'une tour vers un autre.

Fonction deplacer(Entrée/sortie : bord : matrice $[1, 3][1, n]$ d'entiers, Entrée : depart, arrivee : 1..3)

Variable :

i, j : entier;

début

```
    // Trouver le disque supérieur de la tour depart
    i ← 1;
    tant que i ≤ n et bord[depart][i] = 0 faire
        | i ← i + 1;
    fin tq
    // Trouver le disque supérieur de la tour arrivee
    j ← 1;
    tant que j ≤ n et bord[arrivee][j] = 0 faire
        | j ← j + 1;
    fin tq
    bord[arrivee][j - 1] ← bord[depart][i];
    bord[depart][i] ← 0;
```

fin

Calcul de la complexité La complexité est exprimée en terme de nombre d'opérations de déplacement effectuées. En l'occurrence, le nombre de déplacements est exprimé selon la suite

numérique suivante :

$$h(1) = 1$$

$$h(n) = 2 * h(n - 1) + 1$$

où n représente le nombre total de disque à déplacer.

En remplaçant $h(n - 1)$ par la formule réccurente, on obtient :

$$h(n) = 2 * (2 * h(n - 2) + 1) + 1$$

$$h(n) = 2 * (2 * (2 * h(n - 3) + 1) + 1) + 1$$

...

$$h(n) = 2^n - 1$$

Ce résultat peut être démontré par récurrence comme suit :

Cas de base ou pour $n = 1$:

$$h(1) = 2^1 - 1 = 1$$

Donc la formule est correcte pour $n = 1$.

Supposons que la proposition $h(i) = 2^i - 1$ est correct pour $\forall i \leq n$. Montrons qu'elle est aussi correcte pour $n + 1$:

$$h(n + 1) = 2 * h(n) + 1$$

D'après l'hypothèse :

$$h(n + 1) = 2 * (2^n - 1) + 1$$

$$h(n + 1) = 2^{n+1} - 1$$

Donc la proposition est correct $\forall n \in \mathbb{N}$.

Et ainsi la complexité est égale à $\mathcal{O}(2^n)$.

1.3 Algorithme de vérification

Le problème des tours de Hanoi à trois tours admet une unique solution, sous forme d'une suite de déplacements qui génèrent un séquençement d'états intermédiaires.

Pour vérifier la validité d'une solution quelconque, nous devons nous assurer que chaque déplacement est bien réglementaire (concerne le disque le plus haut et ne pose pas un disque sur un autre disque de taille plus petite) et que le dernier état engendré correspond effectivement à l'état but, c.à.d. que toutes les tours sont vides sauf la tour cible. De plus, la séquence de déplacement doit être exactement de longueur $2^n - 1$ tel que n représente le nombre de disques, car la solution est unique, donc égale à la solution calculée par l'algorithme exacte.

Nous nous assurons que l'algorithme de génération de solutions produit des séquences de déplacements qui respectent ces conditions susmentionnées. Par conséquent, nous devons vérifier que le dernier état qui doit correspondre à l'état but dans lequel les anneaux sont alignés sur la tour cible ou destination.

Fonction verification(Entrée : bord : matrice $[1, 3][1, n]$ d'entiers, arrivee : 1..3) : booléen

Variable :

début

```

    pour  $i \leftarrow 1$  à  $n$  faire
        si  $\text{bord}[\text{arrivee}][i] \neq i$  alors
            retourner faux;
        fin si
    fin pour
    retourner vrai;

```

fin

Calcul de complexité La complexité de l'algorithme de vérification est triviale, elle est égale à la complexité du parcours séquentiel des éléments d'un tableau (la tour d'arrivée).

La complexité temporelle est égale à $\mathcal{O}(n)$ tel que n est le nombre de disques.

Quant à la complexité spatiale, sachant que la fonction de vérification ne prend que le dernier état comme paramètre, elle est donc égale à la taille de la matrice d'entiers : $3 * n * n \cong \mathcal{O}(n^2)$.

1.4 Illustration d'une instance du problème

Nous étudions dans ce qui suit une instance du problème des tours de hanoi avec une disposition de trois tours et trois disques sur la tour initiale.

$$\mathbf{bord} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}$$

L'unique solution à cette disposition est une séquence suivante de 7 déplacements :

— On déplace le disque de taille 1 de la tour 1 vers la tour 3.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 1 \end{pmatrix}$$

— On déplace le disque de taille 2 de la tour 1 vers la tour 2.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 2 & 1 \end{pmatrix}$$

— On déplace le disque de taille 1 de la tour 3 vers la tour 2.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix}$$

— On déplace le disque de taille 3 de la tour 1 vers la tour 3.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 3 \end{pmatrix}$$

— On déplace le disque de taille 1 de la tour 2 vers la tour 1.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 3 \end{pmatrix}$$

— On déplace le disque de taille 2 de la tour 2 vers la tour 3.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 1 & 0 & 3 \end{pmatrix}$$

— On déplace le disque de taille 1 de la tour 1 vers la tour 3.

$$\mathbf{bord} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

Chapitre 2

Étude expérimentale

2.1 Complexité théorique de l'algorithme de résolution

Complexité temporelle La complexité de l'algorithme de résolution comme calculée précédemment est de l'ordre de $\mathcal{O}(2^n)$, et elle est précisément égale à $2^n - 1$ qui est une complexité exponentielle.

Le tableau suivant représente les temps d'exécution théorique en nanoseconde de l'algorithme de résolution selon la variation de la taille du problème (le nombre de disques) :

N	1	10	50	100	150	250	500	750	1000
t(ns)	2	1024	1,1259E+15	1,26765E+30	1,42725E+4	1,80925E+75	3,27339E+150	5,92239E+225	1,07151E+301

La figure suivante (voir Figure 2.1) représente l'évolution du temps d'exécution théorique selon le nombre de disques :

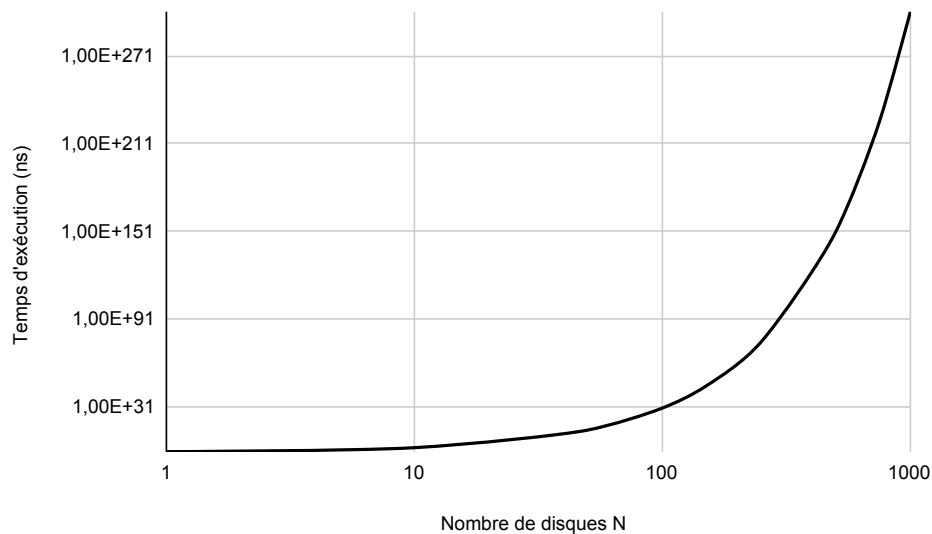


FIGURE 2.1 – Temps d'exécution théorique de l'algorithme de résolution

Depuis le graphe, nous observons que le temps d'exécution évolue de manière exponentielle selon le nombre de disques de départ. Il atteint rapidement des temps d'exécution incommensurables le rendant inexploitable.

Complexité spatiale L'algorithme exploite une matrice $3 \times n$ tel que n est le nombre de disques. Chaque disque est représenté par un entier de taille 4 octets. De plus, la taille de la représentation est statique au cours de l'exécution. Par conséquent, la complexité spatiale est égale à $4 * 3 * n$ octets donc de l'ordre de $\mathcal{O}(n)$.

Cependant, l'algorithme récursif fait au maximum n appels récursifs (le nombre maximum d'appels est le nombre maximum de disques pouvant être déplacés à la fois). L'adresse de retour étant stockée sur 8 octets, la taille maximal de la pile d'appel de fonctions exploitée est donc égale à $8 * n$ octets qui est de l'ordre de $\mathcal{O}(n)$.

La complexité spatiale totale est donc de l'ordre de $\mathcal{O}(n)$.

2.2 Complexité théorique de l'algorithme de vérification

Complexité temporelle La complexité temporelle de l'algorithme de vérification est linéaire et elle est égale à $n \cong \mathcal{O}(n)$, car l'algorithme parcourt la tour d'arrivée et vérifie si les disques sont bien rangés dans le bon ordre.

Le tableau suivant représente les temps d'exécution théorique en nanoseconde de l'algorithme de vérification selon la variation de la taille du problème (le nombre de disques) :

N	1	10	50	100	150	250	500	750	1000
t(ns)	1	10	50	100	150	250	500	750	1000

La figure suivante (voir Figure 2.2) représente l'évolution du temps d'exécution théorique selon le nombre de disques :

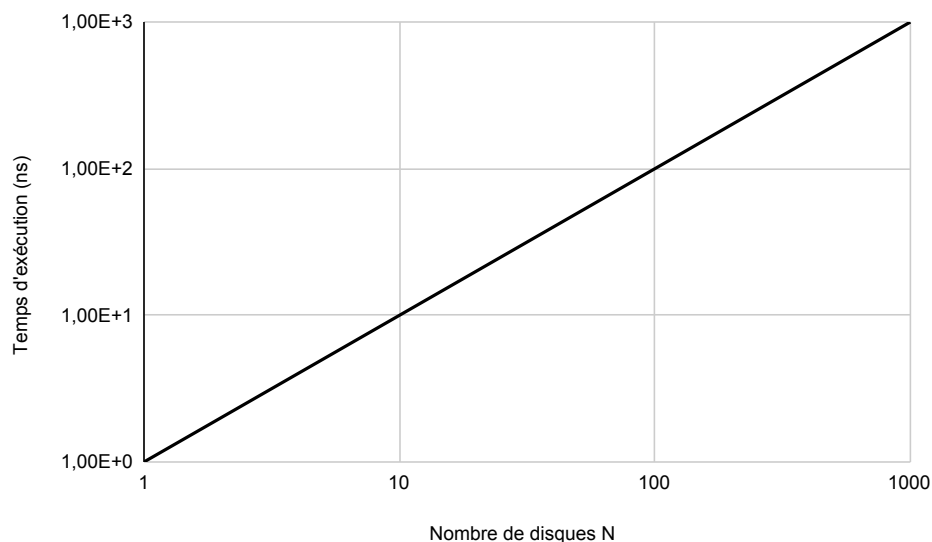


FIGURE 2.2 – Temps d'exécution théorique de l'algorithme de vérification

Depuis le graphe, nous observons que le temps d'exécution évolue avec une tendance linéaire avec l'augmentation du nombre de disques.

Complexité spatiale La complexité spatiale de l'algorithme de vérification est égale à la taille de la matrice plus l'indice de la tour cible qui est un entier, c.à.d. $3 * n + 4$ octets donc de l'ordre $\mathcal{O}(n)$.

2.3 Expérimentation

2.3.1 Algorithme de résolution

Le tableau suivant représente les temps d'exécution en nanoseconde de l'algorithme de résolution récursif selon la variation du nombre de disques.

N	3	5	7	10	15	20	25
t1(ns)	663	2029	7829	59701	2168020	90192200	3190280000
t2(ns)	595	2099	8018	60986	2648340	88261800	3188540000
t3(ns)	642	2118	11230	62336	2206080	89520400	3176230000
t4(ns)	597	2085	8118	61247	2211980	89318100	3179730000
t5(ns)	596	2021	7724	61253	2168540	88693300	3187330000
t6(ns)	882	2042	8044	61308	2275150	88450100	3175950000
t7(ns)	867	2067	7977	63335	2258050	88788800	3181730000
t8(ns)	615	2059	8085	63063	2301650	88529200	3178430000
t9(ns)	593	2077	7942	60909	2211830	88074100	3176490000
t10(ns)	619	2182	7955	61240	2221290	89301200	3182450000
moyenne(ns)	667	2078	8292	61538	2267093	88912920	3181716000

La figure suivante (voir Figure 2.3) représente l'évolution du temps d'exécution selon le nombre de disques.

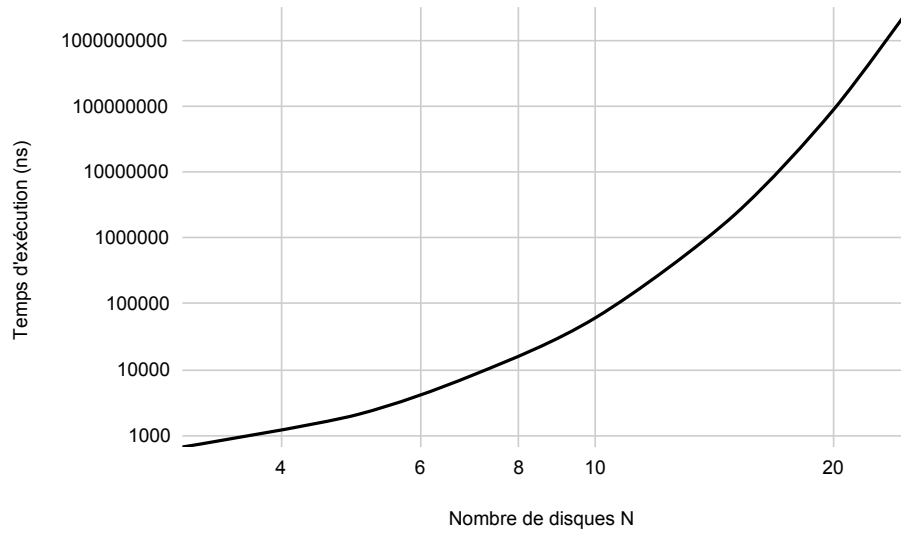


FIGURE 2.3 – Temps d'exécution de l'algorithme de résolution

Depuis la figure 2.3, nous remarquons bien que le temps d'exécution de l'algorithme de résolution est exponentiel, conforme à la complexité de $\mathcal{O}(2^n)$.

2.3.2 Algorithme de génération de solution

Cette algorithme fait au minimum $2^n - 1$ itérations, qui est équivalent au nombre d'étapes de la solution. Si l'algorithme génère à une étape un déplacement non autorisé, ce déplacement est rejeté et un autre est généré (si à son tour il n'est pas autorisé en génère un autre et ainsi de suite). Donc il est de complexité $\mathcal{O}(2^n)$.

Le tableau suivant représente les temps d'exécution en nanoseconde de l'algorithme de génération de solution pour le problème des tours de hanois selon la variation du nombre de disques.

N	3	5	7	10	15	20	25
t1(ns)	16484	25099	73290	304737	11255800	401042000	14478900000
t2(ns)	13480	23469	72592	310506	11503700	402620000	14470300000
t3(ns)	15804	22381	70229	321948	11607800	401610000	14440400000
t4(ns)	16334	24410	69875	312012	11348400	401075000	14418900000
t5(ns)	15780	23887	79824	329052	11008400	402276000	14439500000
t6(ns)	14275	21296	72074	321122	11444300	402046000	14443800000
t7(ns)	13965	30082	52879	309217	1,17E+07	405221000	14440000000
t8(ns)	14042	22535	62667	310238	1,14E+07	405030000	14471900000
t9(ns)	13832	22679	49077	311514	11376700	405483000	14419800000
t10(ns)	15764	22572	69374	311673	11215200	402717000	14449700000
moyenne(ns)	14976	23841	67188	314202	11386730	402912000	14447320000

La figure suivante (voir Figure 2.4) représente l'évolution du temps d'exécution selon le nombre de disques.

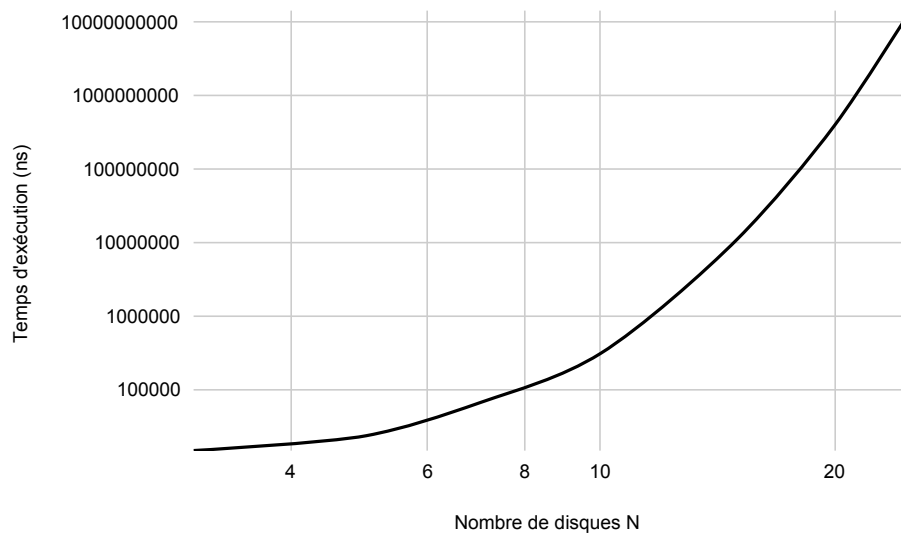


FIGURE 2.4 – Temps d'exécution de l'algorithme de génération de solution

Depuis la figure 2.4, nous remarquons bien que le temps d'exécution de l'algorithme de résolution est exponentiel, conforme à la complexité de $\mathcal{O}(2^n)$. Cependant, les temps d'exécution sont bien plus élevés que ceux de l'algorithme de résolution, car même si l'algorithme de génération fait $2^n - 1$ itération, certains déplacements sont rejetés.

2.3.3 Algorithme de vérification de solution

Le tableau suivant représente les temps d'exécution en nanoseconde de l'algorithme de vérification de solution pour le problème des tours de hanois selon la variation du nombre de disques.

N	3	5	7	10	15	20	25
t1(ns)	71	61	113	80	75	146	182
t2(ns)	61	68	111	68	77	94	61
t3(ns)	66	95	66	80	143	84	156
t4(ns)	68	68	82	66	101	86	143
t5(ns)	76	75	97	108	65	141	197
t6(ns)	57	67	81	110	64	104	179
t7(ns)	58	90	77	104	180	148	91
t8(ns)	74	73	70	78	137	147	102
t9(ns)	78	75	74	117	105	126	105
t10(ns)	60	66	68	80	68	78	164
moyenne(ns)	67	74	84	89	102	115	138

La figure suivante (voir Figure 2.5) représente l'évolution du temps d'exécution selon le nombre de disques.

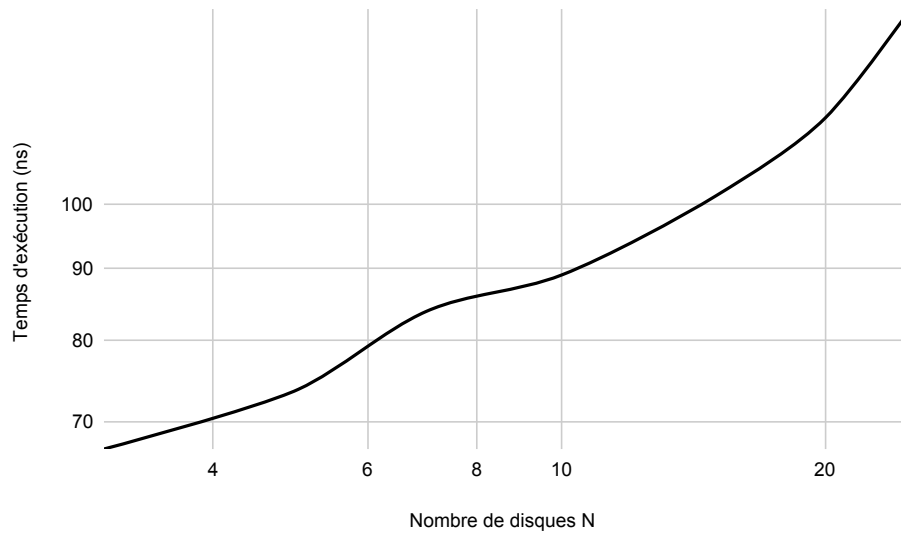


FIGURE 2.5 – Temps d'exécution de l'algorithme de vérification de solution

Depuis la figure 2.5, nous observons une trajectoire presque linéaire, qui est représentative de la complexité $\mathcal{O}(n)$. Les fluctuations sont dues aux nombres de disques relativement bas.