LAB 2

Algorithm 1 (A1) : Naive approach

1. Écrivez l'algorithme correspondant

1. Début
2. Print(("Algorithme 1 : Entrez un nombre pour tester s'il est premier : »)
3. Lire n ;
4. Cmp=0 ;
5. Pour chaque entier iii allant de 1 à n
6. If(n mod i=0)alors
7. cmp ++ ;
8. faire ;
9. Si cmp=2 alors :
10. Print("Oui, n est un nombre premier.") ;
11. Sinon
12. Print("Non, n n'est pas un nombre premier.")
13. Fin

2. Calculate the best and worst case theoretical complexity of this algorithm in Big Oh

Meilleur cas:

Le meilleur cas se produit lorsque n est égal à 1:

Si n=1, la condition $n <= 1$ est vraie, donc la boucle ne s'exécute pas.

Complexité asymptotique dans le meilleur cas : O(1)

Pire cas:

Se produit pour des grandes valeurs de n, où la boucle s'exécute complètement jusqu'à i=n.

1. **Début**
2. **Print(**("Algorithme 1 : Entrez un nombre pour tester s'il est premier : ») →1
3. Lire n ; →1
4. Cmp=0 ; →1
5. Pour chaque entier iii allant de 1 à n → n
6. If(n mod i=0)alors →2
7. cmp ++ ; →2
8. faire ;
9. Si cmp=2 alors : → 2

10. Print("Oui, n est un nombre premier.") ; →1
11. Sinon
12. Print("Non, n n'est pas un nombre premier."
13. **Fin**

f(n)=1+1+1+n(4)+3=4n+6
alors, la complexité dans le pire cas est O(n)

3. Écrire le programme correspondant.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n, cmp = 0;

    printf("Algorithme 1 : Entrez un nombre pour tester s'il est premier : ");
    scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            if (n % i == 0) cmp++;
        }
        if (cmp == 2) {
            printf("N = %d\nEst-ce un nombre premier ?\n Oui", n);
        } else {
            printf("N = %d\nEst-ce un nombre premier ?\n Non", n);
        }
    return 0;
}
```

n	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T(s)	0.003	0.005	0.010	0.017	0.034	0.069	0.132
Premier ?	oui	oui	oui	oui	oui	oui	oui

n	128000003	256000001	16000057	512000009	1024000009	2048000011
T	0.269	0.536	0.047	1.065	2.125	4.257
Premier ?	oui	oui	oui	oui	oui	oui

c- Que remarquons-nous à propos des données et des mesures obtenues ?

Nous remarquons que d'après les données et mesures obtenues, la croissance de la complexité temporelle est linéaire ~ O(n)

d- Comparer la complexité théorique et les mesures expérimentales.

Les prédictions théoriques sont-elles compatibles avec les mesures expérimentales ?

$$\boldsymbol{T(n) = f(n) * \Delta t}$$

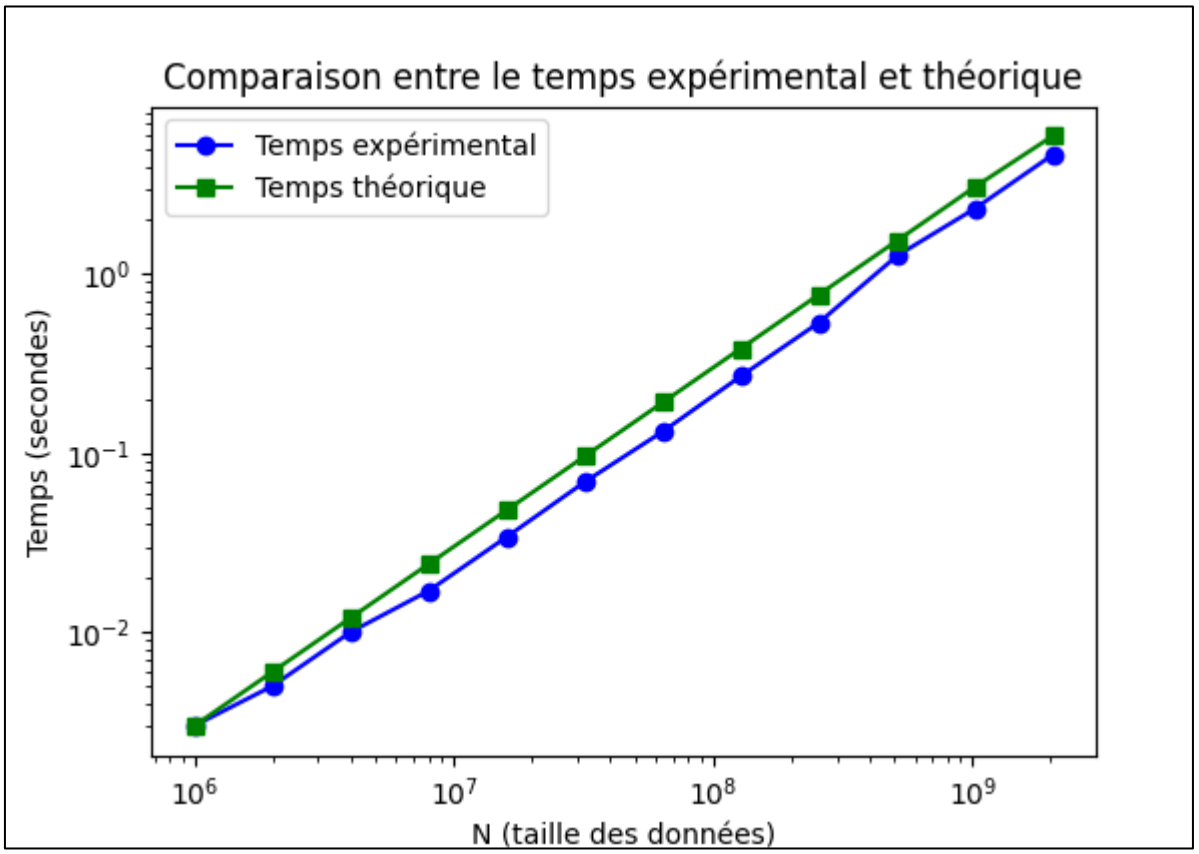
$$\Delta t = \frac{T(n)}{4n+6} = \frac{0.003}{4 * 1000003 + 6} = 7.5 * 10^{-10}$$

n	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T(s)	0.003	0.005	0.010	0.017	0.034	0.069	0.132
T theori	0.003	0.006	0.012	0.024	0.048	0.096	0.192

n	128000003	256000001	512000009	1024000009	2048000011
T	0.269	0.536	1.265	2.325	4.657
T theori	0.384	0.768	1.536	3.072	5.93

Les prédictions théoriques et les mesures expérimentales sont très proches.

e- Represent (in the same graph) with 2 curves the variations of the experimental and theoretical execution time T(N) for best case and 1 curve for experimental worst case.



Algorithm 2 (A2)

1) Écrivez l'algorithme correspondant

```
1.  n, cmpt, i integer;
2.  Begin
3.  cmp ← 0;
4.  i ← 1;
5.  print('entrez un nombre pour tester s'il est premier ')
6.  lire(n);
7.  Pour i de 1 a n/2 faire
8.  if ( n%i == 0 ) then
9.  cmp ← cmp + 1;
10. end if
11. i ← i + 1;
12. fait
13. if ( cmp == 1 ) then
14. print('Oui, n est un nombre premier.' );
15. else
16. Print("Non, n n'est pas un nombre premier.")
17. end if
18. End
```

Calculate the best and worst case theoretical complexity of this algorithm in Big Oh

Meilleur cas:

Le meilleur cas se produit lorsque n est égal à 1:

Si n=1, la condition $n <= 1$ est vraie, donc la boucle ne s'exécute pas.

Complexité asymptotique dans le meilleur cas : O(1)

Pire cas:

Se produit pour des grandes valeurs de n, où la boucle s'exécute complètement jusqu'à i=n/2.

```
1.  cmp ← 0; →1
2.  print('entrez un nombre pour tester s'il est premier ') →1
3.  lire(n); →1
4.  Pour i de 1 a n/2 faire →n/2
5.  if ( n%i == 0 ) then  →2
6.  cmp ← cmp + 1;  →2
7.  end if
8.  fait
```

```
9.  if ( cmp == 1 ) then →2
10. print('Oui, n est un nombre premier.' ); →1
11. else
12. Print("Non, n n'est pas un nombre premier.")
13. end if
```

f(n)=1+1+1+$\frac{n}{2}$ (4)+3=4$\frac{n}{2}$+6=2n+6

alors, la complexité dans le pire cas est O(n)

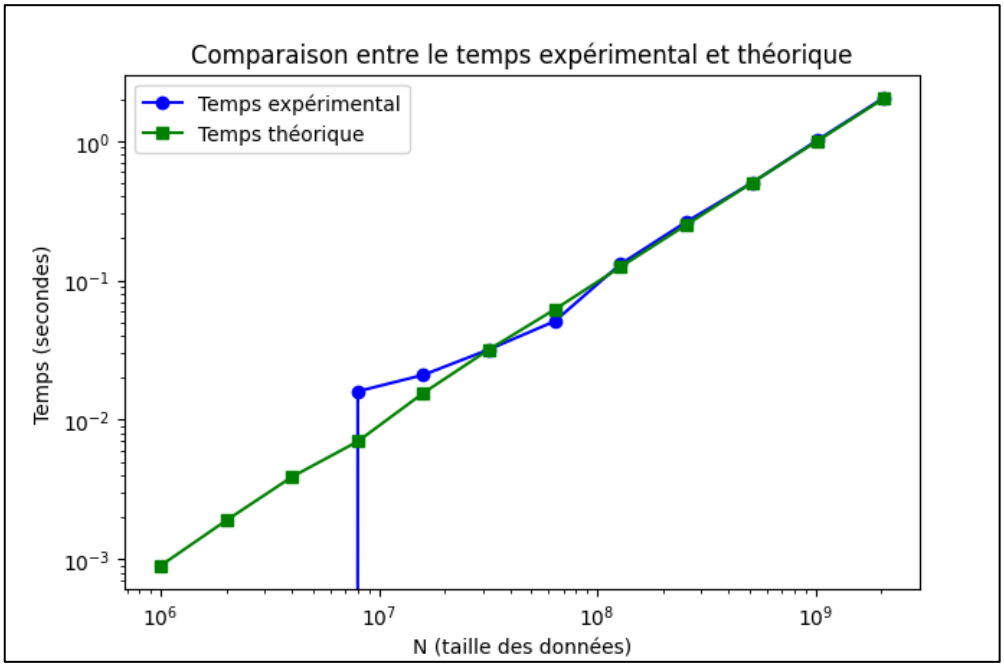
$$\boldsymbol{T(n) = f(n) * \Delta t}$$

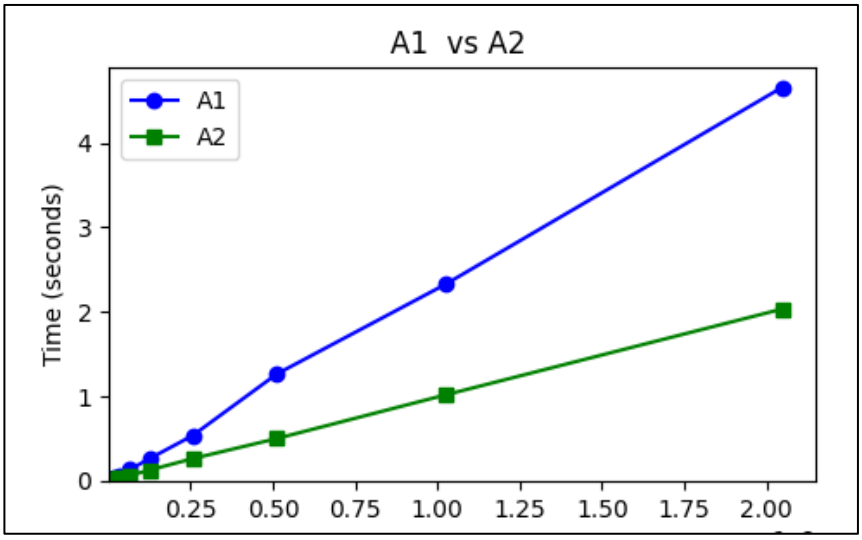
$$\boldsymbol{\Delta t = \frac{T(n)}{2n+6}} = \frac{0.032}{2 * 32000011 + 6} = \boldsymbol{4.9 * 10^{-10}}$$

n	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T(s)	0.000	0.000	0.000	0.016	0.021	0.032	0.051
T theori	0.0009	0.0019	0.0039	0.007	0.0156	0.032	0.062

n	128000003	256000001	512000009	1024000009	2048000011
T	0.131	0.263	0.501	1.017	2.034
T theori	0.125	0.25	0.5017	1.003	2.007

Les prédictions théoriques et les mesures expérimentales sont très proches.





La deuxième solution A2 prend environ la moitié du temps de la première solution A1, donc la deuxième solution est plus efficace

Algorithm 3 (A3)

Écrivez l'algorithme correspondant

1. n, cmpt, i integer;
2. Begin
3. cmp ← 0;
4. i ← 1;
5. print(‘entrez un nombre pour tester s’il est premier ’)
6. lire(n);
7. Pour i de 1 a $\sqrt{n}$ faire
8. if (n%i == 0) then
9. cmp ← cmp + 1;
10. end if
11. i ← i + 1;
12. fait
13. if (cmp == 1) then
14. print(‘Oui, n est un nombre premier.’);
15. else
16. Print("Non, n n'est pas un nombre premier.")
17. end if
18. End

Calculate the best and worst case theoretical complexity of this algorithm in Big Oh

Meilleur cas:

Le meilleur cas se produit lorsque n est égal à 1:

Si n=1, la condition $n <= 1$ est vraie, donc la boucle ne s'exécute pas.

Complexité asymptotique dans le meilleur cas : O(1)

Pire cas:

Se produit pour des grandes valeurs de n, où la boucle s'exécute complètement jusqu'à i=n/2.

```
1.  cmp ← 0; →1
2.   print('entrez un nombre pour tester s'il est premier ') →1
3.   lire(n); →1
4.   Pour i de 1 a √n faire →√n
5.   if ( n%i == 0 ) then  →2
6.   cmp ← cmp + 1;  →2
7.   end if
8.  fait
9.   if ( cmp == 1 ) then →2
10.  print('Oui, n est un nombre premier.' ); →1
11.  else
12. Print("Non, n n'est pas un nombre premier.")
13.  end if
```

f(n)=1+1+1+$\frac{n}{2}$ (4)+3=4$\sqrt{n}$+6=2$\sqrt{n}$+6

alors, la complexité dans le pire cas est O($\sqrt{n}$)

n	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T(s)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

n	128000003	256000001	512000009	1024000009	2048000011
T	0.0000	0.0000	0.0000	0.0000	0.0000

Algorithm 4 (A4)

Écrivez l'algorithme correspondant

n, cmpt, i integer;

1. Begin
2. cmp ← 0;
3. i ← 1;
4. print('entrez un nombre pour tester s'il est premier ')
5. lire(n);
6. If (n%2 =0) and (n !=2) then
7. print('Non, n n'est pas un nombre premier');
8. else
9. Pour i de 1 a $\sqrt{n}$ faire
10. if (n%i == 0) then
11. cmp ← cmp + 1;
12. end if
13. i ← i + 1;
14. fait
15. if (cmp == 1) then
16. print('Oui, n est un nombre premier.');
17. else
18. Print("Non, n n'est pas un nombre premier.")
19. end if
20. enif
21. End

Calculate the best and worst case theoretical complexity of this algorithm in Big Oh

Meilleur cas:

Le meilleur cas se produit lorsque n est égal à 1:

Si n=1, la condition $n <= 1$ est vraie, donc la boucle ne s'exécute pas.

Complexité asymptotique dans le meilleur cas : O(1)

Pire cas:

Se produit pour des grandes valeurs de n, où la boucle s'exécute complètement jusqu'à i=n/2.

14. cmp ← 0; →1
1. print('entrez un nombre pour tester s'il est premier ') →1
2. lire(n); →1

3. If (n%2 =0) and (n !=2) then → 2
4. print('Non, n n'est pas un nombre premier'); →1
5. else
6. end if
7. Pour i de 1 a $\sqrt{n}$ faire →$\sqrt{n}$
8. if (n%i == 0) then →2
9. cmp ← cmp + 1; →2
10. end if
11. fait
12. if (cmp == 1) then →2
13. print('Oui, n est un nombre premier.'); →1
14. else
15. Print("Non, n n'est pas un nombre premier.")
16. end if

f(n)=1+1+1+1+2+$\frac{n}{2}$ (4)+3=4$\sqrt{n}$+6=2$\sqrt{n}$+8

alors, la complexité dans le pire cas est O($\sqrt{n}$)

$$\boldsymbol{T(n) = f(n) * \Delta t}$$

$$\Delta \boldsymbol{t} = \frac{\boldsymbol{T(n)}}{\boldsymbol{2\sqrt{n} + 8}} = \frac{0.032}{2 * \sqrt{\ } + 8}$$

n	1000003	2000003	4000037	8000009	16000057	32000011	64000031
T(s)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
T theori							

n	128000003	256000001	512000009	1024000009	2048000011
T	0.0000	0.0000	0.0000	0.0000	0.0000
T theori					

L'ordre de complexité des Algorithmes 3 **et** 4 **est de** $\sqrt{N}$ **ce qui est bien plus rapide que les algorithmes vérifiant tous les diviseurs jusqu'à N, mais cela peut entraîner des temps d'exécution trop petits pour la machine donc 0 est affiché**

RESULTAT GLOBAL ET COMPARISON ENTRE LES ALGORITHMES :

Algorithme	A1	A2	A3	A4
complexité	O(n)	O(n)	$O(\sqrt{n})$	$O(\sqrt{n})$

- L'algorithme 2 est une petite amélioration de l'algorithme 1, l'ordre de complexité est le même O(N), les deux algorithmes évoluent linéairement mais l'algorithme 2 fait la moitié d'itération. On voit bien dans le tableau que le temps d'exécution de l'algorithme 2 est approximativement ½ du temps de l'algorithme 1
- L'algorithme 3 représente une amélioration meilleure, car la complexité théorique est réduite à racine carré de N.
- Le temps d'exécution de l'algorithme 3 est négligeable par rapport aux algorithmes 1 et 2, o(N) >>>> o ($\sqrt{N}$)
- L'algorithme 4 est encore une bonne amélioration car les nombres premiers sont des nombres impairs donc il n'est pas nécessaire de tester la divisibilité au nombre pairs Selon le tableau, le temps d'exécution est encore plus petit que l'algorithme 3