LAB 1

Part1 (Sum of the first N natural numbers)

1. Développer un algorithme itératif, noté Somme_1, qui permet le calcul de la somme, notée S, des n premiers nombres naturels :

$$S = \sum_{i}^{n} i = 1 + 2 + 3 + \cdots + n$$

L'entier naturel n doit être lu en entrée (n>=1). Utilisez la boucle while

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

int main() {
    int i, n, s;

    printf("Give the number n: ");
    scanf("%d", &n);
    s = 0;
    i = 1;
    while (i <= n) {
        s += i;
        i++;
    }
    printf("The sum is %d\n", s);
    return 0;
}
```

2. Calculez la complexité temporelle de cet algorithme.

printf("Give the number n: "); →1
scanf("%d", &n); →1
s = 0; →1
i = 1; →1
while (i <= n) { →n+1
s += i; →2n
i++; →2n
}
printf("The sum is %d\n", s); →1

f(n)=1+1+1+1+n+1+2n+2n+1 = 5n+6 →O(n)
L'algorithme a une complexité linéaire O(n).

$$\begin{aligned} T(n) &= f(n) * \Delta t \\ &= 5n + 6 * \Delta t \\ &= 5 * \Delta t * n + 6 * \Delta t \end{aligned}$$

3. Calculer la complexité spatiale de cet algorithme

l'algorithme utilise un espace mémoire fixe car l'espace requis par les variables (i,n,s) est égale =3
Le nombre d'instruction = 8
Le total=8+3=11 octet
Ce qui donne une complexité spatiale constante→ O(1).

4. Développer le programme itératif correspondant, noté PSum_1, avec le langage C.

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

int main2() {
    double i, n, s;
    printf("Give the number n: ");
    scanf("%lf", &n);
    s = 0;
    i = 1;
        while (i <= n) {
        s += i;
        i++;
    }

    printf("The sum is %lf\n", s);
    return 0;
}
```

Part II (Measuring Execution Time)

5. Reprenez le programme précédent et incluez les instructions qui permettent de mesurer les temps d'exécution T pour l'échantillon de valeurs de n donné dans le tableau ci-dessous (le nouveau programme est noté PSum_2).

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

int main3() {
    double i, n, s;
    clock_t T1, T2;
    double D;
    printf("Give the number n: ");
    scanf("%lf", &n);
    T1 = clock();
    s = 0;
    i = 1;
    while (i <= n) {
        s += i;
        i++;
    }
    T2 = clock();
    D = ((double)(T2 - T1)) / CLOCKS_PER_SEC;
    printf("Time taken: %f seconds\n", D);
    printf("The sum is %.0lf\n", s);
    return 0;
}
```

n	10^7	$2*10^7$	10^8	$2*10^8$	10^9	$2*10^9$	10^{10}	$2*10^{10}$	10^{11}	$2*10^{11}$
T Experimental	0.022	0.047	0.178	0.379	1.888	3.757	26.444	50.721	245.392	518.009
T Théorique	0.022	0.044	0.22	0.44	2.2	4.4	22	44	220	440

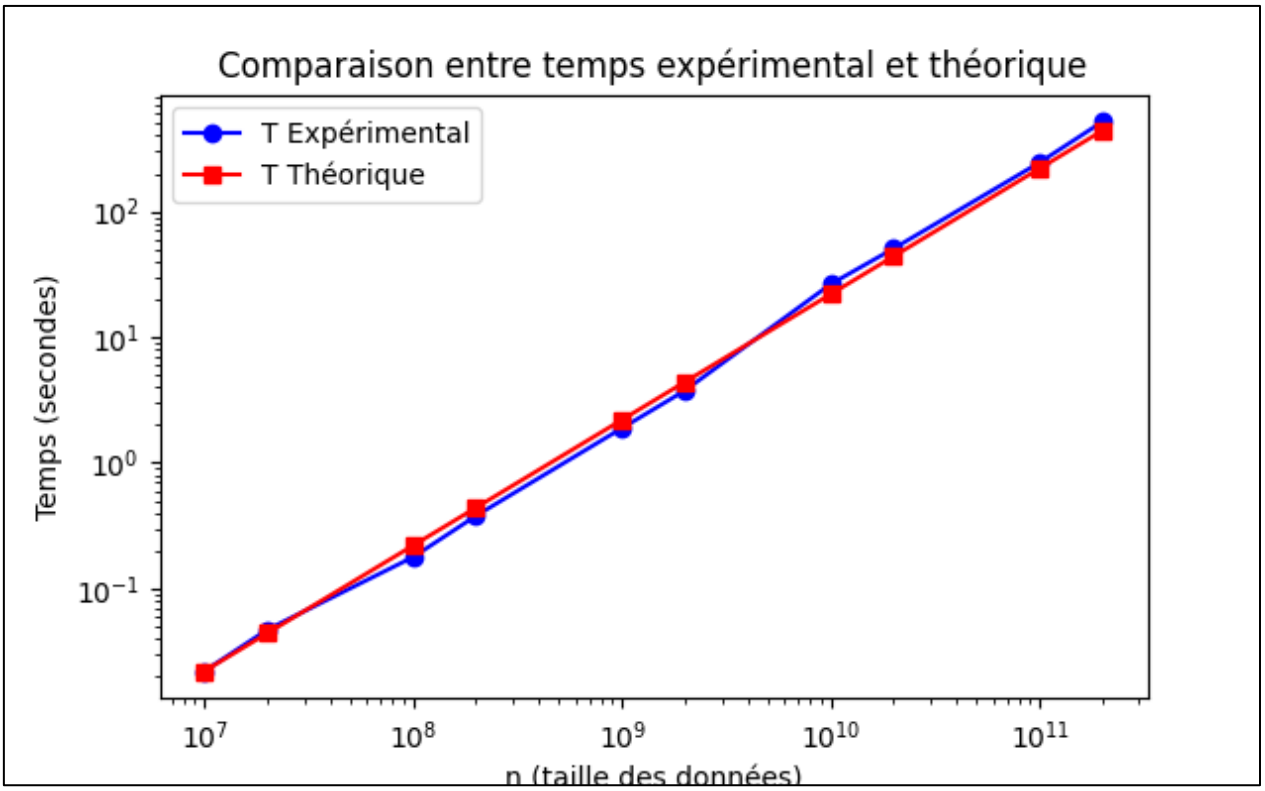
$$f(n) \times \Delta t = T(n)$$

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

$$\Delta t = \frac{T(n)}{5n + 6} = \frac{0.022}{5(10^7) + 6} = 4.4 * 10 - 10$$

6. Tracer les courbes d'évolution du temps d'exécution théorique et expérimental dans Excel.

7.



8. Comparez les courbes et interprétez les résultats obtenus.

La courbe expérimentale suit parfaitement la même tendance exponentielle que les valeurs théoriques. Cela confirme que la complexité temporelle de l'algorithme Somme_1 est bien en O(n)