University Center of Mila | **2nd Year LMD – Computer Science**

Institute of Science and Technology | Course : : Data Structures and Algorithms (A.S.D3)

Lab Session No. 02 – complexity

By calculating algorithmic complexity, our main goals are:

1. To compare the **efficiency of algorithms** that solve the same problem.
2. To verify whether an **algorithm solving a given problem** can provide an answer within a **reasonable time**.

The **first objective** of this lab session is to **practically compare algorithms** based on the **number of iterations performed** (Exercises 1 and 2) and their **execution time** (Exercise 3). The **second objective** is to determine whether a program can **respond within a reasonable time** or not (Exercise 3).

Exercise 1

Let the following two programs calculate the sum of the first *n* natural numbers.

Program 1 :

```
#include <iostream>
using namespace std;
int main() {
int i, n;
   long S = 0, Nbriteration = 0;

   cout << "Enter an integer n: ";
   cin >> n;

   for (i = 1; i <= n; i++) {
      S = S + i;
      Nbriteration++;
   }

   cout << "\nThe sum is: " << S;
   cout << "\nThe number of iterations is: " << Nbriteration << endl;

   return 0;
}
```

Program 2 :

```
#include <iostream>
using namespace std;

int main() {
   int i, j, n;
   long S = 0, Nbriteration = 0;

   cout << "Enter an integer n: ";
   cin >> n;

   for (i = 1; i <= n; i++) {
      for (j = 1; j <= i; j++) {
         S = S + 1;
         Nbriteration++;
      }
   }

   cout << "\nThe sum is: " << S;
   cout << "\nThe number of iterations is: " << Nbriteration << endl;

   return 0;
}
```

Work to do

1. Test both programs with different values of *n* by comparing the number of iterations performed by each program. What do you observe when *n = 100000* for program **P2**?

	N	5	20	50	10000	100000
Nbr Itérations	P1					
	P2					

2. Theoretically, the number of iterations for program **P1** is *N*, while the number of iterations for program **P2** is $N \times (N + 1) / 2$. Compare the results in the table with the theoretical results.
3. Propose a faster solution than the two previous ones.

Exercise 2:

There are two main techniques for searching an element in an array: **sequential search** and **dichotomic search**. In this exercise, we will compare the number of iterations performed by these two techniques when searching for an element in an array filled with 10,000 integers (0, 1, 2, 3, …, 9999).

Work to do

1. Define the two functions **rechseq** and **rechdich**.
   These functions should return the **number of iterations** performed during the search for an element.
2. Test the program for the following values of **x**: 5, 25, 300, 6000, 10000, and -1.

```
#include <iostream>
using namespace std;

// Function prototypes
```

```
int rechseq(int T[], int N, int x);   // Sequential search
int rechdich(int T[], int N, int x);  // Binary search

int main() {
   int T[10000];
   int N = 10000;
   int N1 = 0, N2 = 0, x;

   // Fill the array
   for (int i = 0; i < N; i++)
      T[i] = i;

   cout << "Enter an integer x: ";
   cin >> x;

   N1 = rechseq(T, N, x);
   N2 = rechdich(T, N, x);

   cout << "\nThe number of iterations using sequential search is: " << N1 << endl;
   cout << "The number of iterations using binary search is: " << N2 << endl;

   return 0;
}
```

Exercise 3 – Fibonacci Sequence

The **Fibonacci sequence** is defined as follows:

$$\mathbf{U_0=1,\ U_1=1,\ \text{and for } n \geq 2,\ U_n=U_{n-1}+U_{n-2}.}$$

Work to do

1. **Write two functions** that compute the *n*-th term of the Fibonacci sequence:
   - an **iterative version** (expected time complexity ≈ O(*n*)),
   - a **recursive version** (expected time complexity ≈ $O(2^n)$).
2. Using the functions provided by the **`<time.h>`** library, **measure and compare the execution time** of the two implementations for several values of *n* (for example : 10, 20, 30, 35, 40 …).
   Record your observations in a results table.

N		20	30	40	50	10000	1000000000
Execution time	f1						
	f2					/	/

3. **Based on the measured execution times**,

   1. Which version would you choose to compute the *n*-th Fibonacci term efficiently?
   2. Can the recursive function reasonably be used for large values of *n*? Explain why.