University M’hamed Bougara Boumerdes

Institute of Electrical and Electronics Engineering (ex: INELEC)

Algorithms and Data structures laboratory

LAB N°1

2/26/2022

Superviser:

Dr. A. Zitouni

Students:

Madaoui Zakaria G3

Maallem Nassim G3

Objectives:

- Implementing the Hanoi tower problem
- Comparing the execution time of the recursive and iterative implementation when computing Fibonacci number of some natural number n.

Task 1: The Hanoi problem implementation

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

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod);

int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'B', 'C');
    return 0;
}

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 0) return; // Base condition
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod <<
                                " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

Running the above program for n=4 disks gave the following output.

```
› g++ hanoi.cpp -o hanoi
› ./hanoi
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
Move disk 4 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 2 from rod C to rod A
Move disk 1 from rod B to rod A
Move disk 3 from rod C to rod B
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
```

We can clearly see that the algorithm works correctly as all disk from rod A were transferred to rod B through rod C, according to the rules set at the beginning of the problem.

Task 2: Fibonacci number, recursive VS iterative

```cpp
#include <iostream>
#include <ctime>

using namespace std;

long feb_loop(int n);
long feb_recursive(int n);

clock_t start , end ;

int main(){
    long n = 15;
    long result;
    clock_t start;
    clock_t end;

    cout << "Type a number: \n";
    cin >> n;

    start  = clock(); //measure recursive implementation start time
    result = feb_recursive(n);
    end    = clock(); //measure recursive implementation end time
    cout << "feb_recursive of " << n << " is: " << result << "time taken is: "
         << ((float)(end - start) / CLOCKS_PER_SEC) << " seconds \n";

    start  = clock(); //measure iterative implementation start time
    result = feb_loop(n);
    end    = clock(); //measure iterative implementation end time
    cout << "feb_loop of " << n << " is: " << result << "| time taken is: "
         << ((float)(end - start) / CLOCKS_PER_SEC) << " seconds \n";
    return 0;
}

long feb_loop(int n){
        if(n <= 1) return n;
        long fib = 1;
        long prevFib = 1;
        for(int i=2; i<n; i++) {
            long temp = fib;
            fib+= prevFib;
            prevFib = temp;
        }
        return fib;
}

long feb_recursive(int n){
    if(n == 0 || n== 1) return n;
    return feb_recursive(n - 1) + feb_recursive(n-2);
}
```

Running this program which compares the the execution time of the recursive and iterative implementation of computing Fibonacci number for n = 45 produced the following output:

```
› ./feb
```

```
Type a number:
45
feb_recursive of 45 is: 1134903170| time taken is: 11.8424 seconds
feb_loop of 45 is: 1134903170| time taken is: 2e-06 seconds
```

Conclusion:

We can see a huge difference between the running time of the iterative and recursive algorithm. The recursive implementation was simpler and required less code, but it executed in 11.84 seconds. In contrast, the iterative implementation required few more lines of code but executed in almost no time ! (2e-6 seconds).

This experiment is also backed by the theory we covered in class. Analyzing the iterative implementation we can see that:

$$T1(n) = \begin{cases} 1 \; if \; n = 0 \; or \; 1 \\ n - 2 \; otherwize \end{cases} \xrightarrow{yields} O(n-2) \text{ for } n \gg 2$$

And analyzing the recursive implementation we have

$$T2(n) = \begin{cases} 1 \; if \; n = 0 \\ 2T(n-1) + 1 \; otherwize \end{cases} \xrightarrow{yields} O(2^n-1) \text{ for } n \gg 1$$

Where $O(2^n-1)$ grows exponentially and gets excessively large with large values of n, whereas the iterative implementation grows linearly O(n-2)