Algorithm analyze
from the time execution point of view.
Laboratory work nr. 1

Bîrcu Maxim

October 6, 2015

Student: Bîrcu Maxim
Instructor: Cojanu Irina

1 Source code in python language.

```
import time
import matplotlib.pyplot as plt

def fib1(n):
        if n<2:
                return n
        return fib1(n-1) + fib1(n-2)

def fib2(n):
        i = 1
        j = 0
        for k in range(1,n+1):
                j = i + j
                i = j - i
        return j

def fib3(n):
        i = 1
        j = 0
        k = 0
        h = 1
        while n>0:
                if (n % 2 == 1):
                        t = j*k
                        j = i * h + j * k + t
                        i = i * k + t
                t = h ** 2
                h = 2 * k * h + t
```

```
            k = k ** 2 + t
            n = n / 2
        return j

def getTime(function,argument):
        start = time.time()
        function(argument)
        end = time.time()
        return end - start

def Times(f, arr):
    allTimes = []
    for n in arr:
        allTimes.append(getTime(f, n))
    return allTimes

def plot(f, arr, title):
    plt.plot(arr, Times(f, arr))
    plt.title(title)
    plt.ylabel("times")
    plt.xlabel("Fn")
    plt.show()

plot(fib1, range(0, 30), title="Timeplot for 1st algorithm")
plot(fib2, range(0, 10000), title="Timeplot for 2nd algorithm")
plot(fib3, range(0, 20000,50), title="Timeplot for 3rd algorithm")
```

2 Execution time graphs for fibonacci algorithms.

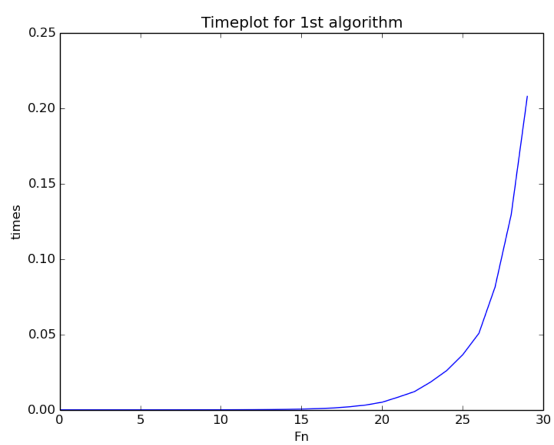

Figure 1: Graph for first algorithm fib1()

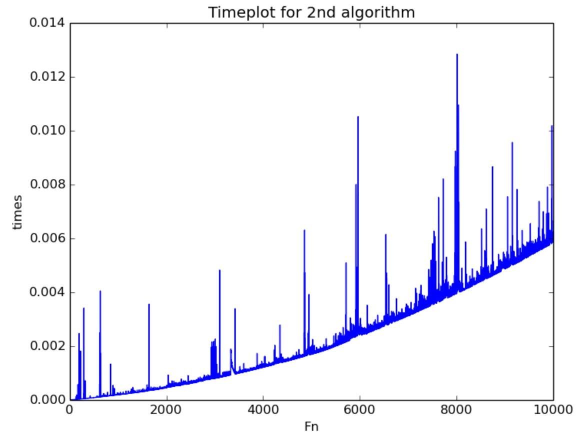

Figure 2: Graph for second algorithm fib2()

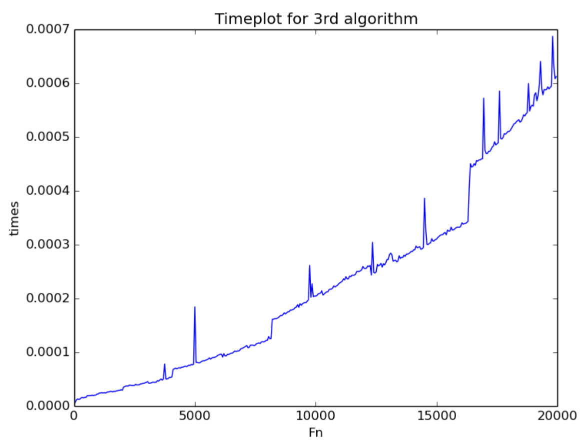

Figure 3: Graph for third algorithm fib3()

3 Algorithm analyze from the time execution point of view.

3.1 Complexity analysis for fib1 function

We have a recursive function which number of operations is double and therefore:

$$\mathrm{O}(\mathrm{Fib}_1(\mathrm{n}))=\mathrm{O}(2^{\mathrm{n}})$$

3.2 Complexity analysis for fib2 function

I observed that for for large enough input sizes the running time increases linearly, therefore we have an linearithmic function which complexity is:

$$O(Fib_2(n))=O(n)$$

3.3 Complexity analysis for fib3 function

Because the number of operations is divided to 2 every frame the complexity of the algorithm is:

$$O(Fib_3(n))=O(\log_2 n)$$

4 Conclusion: Sometimes we can solve a problem using a different methods and different algorithms , in this laboratory work for finding fibonacci sequence I've used 3 types of algorithms one recursive and 2 iterative and find out that recursion is an elegance method to solve the problem but from the execution time point of view , an iterative one is better