Algorithm Analyze
Time of execution for different algorithm
Laboratory Work 1

Vasile Schidu

October 7, 2015

Date Performed:	October 1, 2014
Partners:	Vasile Schidu
Instructor:	Cojanu Irina

1 Source Code

```
import time
import matplotlib.pyplot as plt
from matplotlib import pyplot as PLT
from numpy import *

A = []
B = []
C = []

def Fib1(n):
        if n<2:
                return n
        return Fib1(n-1) + Fib1(n-2)   $O(2^n)$

def Fib2(n):
        i = 1   $O(1)$
        j = 0   $O(1)$
        for k in range(1,n+1): # O(n-1)
                j = i + j   $O(1)$
                i = j - i   $O(1)$
        return j

def Fib3(n):
        i = 1   $O(1)$
        j = 0   $O(1)$
        k = 0   $O(1)$
        h = 1   $O(1)$
        while n>0: $O(logn)$
                if (n % 2 == 1):
                        t = j*k   $O(1)$
                        j = i * h + j * k + t   $O(1)$
                        i = i * k + t   $O(1)$
                t = h ** 2   $O(1)$
                h = 2 * k * h + t   $O(1)$
                k = k ** 2 + t   $O(1)$
                n = n / 2   $O(1)$
        return j
```

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

n = 10000

for i in range(n):
        #A.append(Time(Fib1,i))
        B.append(Time(Fib2,i))
        C.append(Time(Fib3,i))

plt.plot(B)
plt.show()
plt.plot(C)
plt.show()
```

2 Complexity Analysis :

2.1 Complexity Analysis for Fib1 function

Because it is a recursive function and the number of calling it is doubling every time so the complexity is :

$$O(Fib_1(n))=O(2^{n})$$

2.2 Complexity Analysis for Fib2 function

We can easily see that this algorithm is liniar so it's complexity is :

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

2.3 Complexity Analysis for Fib3 function

I observed that in while loop n is divided by 2 every time ($n = n/2$), that's why the execution time is devided progressively, so the complexity is:

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

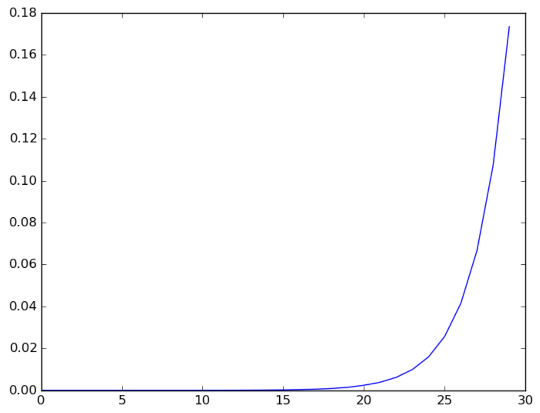

Figure 1: Graph for first function (Fib1)

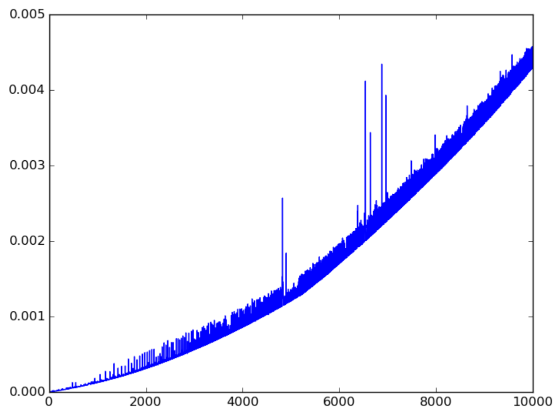

Figure 2: Graph for second function (Fib2)

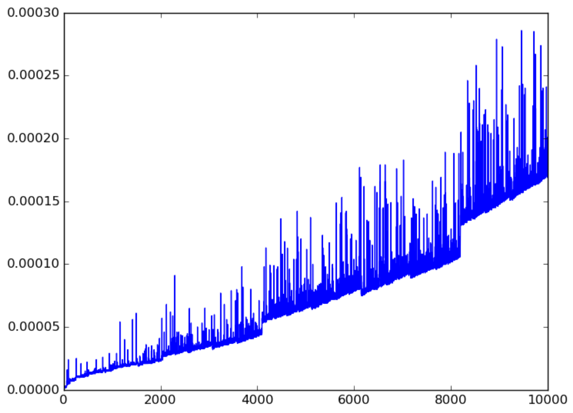

Figure 3: Graph for third function (Fib3)

3 Conclusion:

3.1 Definitions

Fib1 The first algorithm grows very fast, and for values greater than 40 it takes too much time to compute.This happens because the algorithm is recursive and complexity of recursive algorithm is very high.Even if recursion look "sexier", it's not a good idea to implement it for a huge number of iteration.

Fib2 The second algorithm is linear, so it's complexity is n, that coincide with input.

Fib3 The third algorithm is the fastest, this is beacause of logarithmic complexity and it computes easily very big values of n.