****

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Raport**

**pentru lucrarea de laborator Nr.1**

***la cursul de “Analiza și proectarea algoritmilor ”***

***Tema:Numerele Fibonacci.***

Efectuat: Studentul gr. TI-207 **Bunescu Gabriel**

Verificat:  *asist. Univ* **Bîtca Ernest**

**Chișinău – 20****21**

***Scopul lucrării:***

1. Analiza teoretică a algoritmilor.

2. Analiza empirică a algoritmilor.

3. Determinarea complexității temporale şi asimptotice a algoritmilor

Şirul lui Fibonacci este definit prin recurenţa următoare:



Termenul n al şirului poate fi obţinut direct prin definiție:

function fib1(n)

if n < 2 then return n

else return fib1(n-1) + fib1(n-2)

Această metodă nu este atît de reușită deoarece recalculează de mai multe ori aceleași valori.O altă metodă mai eficace care permite rezolvarea problemei este:

**function** *fib*2(*n*)  
     *i* = 1; *j* = 0  
     **for** *k* = 1 **to** *n* **do** *j* =*i* + *j*                                 *i* = *j* - *i* **return** *j*

Mai există un algoritm :

**function** *fib*3(*n*)  
     *i* = 1; *j* = 0; *k* = 0; *h* = 1  
     **while** *n* > 0 **do**          **if** *n* este impar **then** *t* = *jh  
                                           j* = *ih*+*jk*+*t  
                                           i* = *ik*+*t*          *t*  = *h*2  
          *h* = 2*kh*+*t*          *k* = *k*2+*t*          *n* = *n* **div** 2  
     **return** *j*

**Determinam complexitatea asimptomatica a acestor metode**

Recurența care defineşte şirul lui Fibonacci este:

tn = tn-1 + tn-2 n>=2

si t0 = 0 , t1 = 1.

Putem să rescriem aceasta recurență în altă formă:

tn- tn-1-tn-2 = 0, care are ecuația caracteristică:

x2-x-1=0, cu radacinile r1,2 = (1±√5)/2.

Soluția generală are forma:

Tn = c1r1n+c2r2n

Concluzionăm că timpul algoritmului care determină şirul lui Fibonacci prin metoda 1,creşte în mod exponențial în funcție de n.

O altă metodă propusă a acestei probleme:

function fib2(n)

i← 1; j+0

for k← 1 to n do j ←i+j

i← j-1

return j



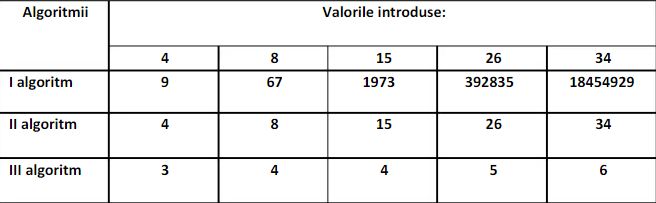
T(n)=C₁+C₂+(C3+C4+C5)n, scriem cit c₂+c1 prin k1, si (c3+c4+c5) prin k₂ =>

t(n)=k1+k₂n si mai departe ca t(n)=O(n). In aceasta metoda, timpul de

executie depinde liniar de n.

Metoda a 3 este o functie logaritmică: tn= O(log2 n)..deoarece avem oarecare costanta n ce se îmulteste cu log2 n și avem ciclul while se execută atîta timp cît n>0 dar n mereu se imparte la 2.

**Numarul de iteratii pentru fiecare algoritm :**



**Codul Sursa:**

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <bits/stdc++.h>

using namespace std;

long int c = 0,d=0,e=0;

long double fib1(int n);

long int fib2(int n);

long int fib3(int n);

int main()

{

int clock\_t ,start, finish;

int n;

cout << "Dati n:",

cin>>n; cout << "\n";

start = clock();

cout << "fib1 Rezultat:" <<fib1(n) << "\n";

finish = clock();

cout << "Numarul de iteratii:"<<c<< endl;

cout << fixed << setprecision(6) << "Timpul de executie pentru fib1=" << ((double)finish-start)/CLOCKS\_PER\_SEC << endl;

cout <<"\n";

start = clock();

cout <<"fib2 Rezultat:"<<fib2(n)<<"\n";

finish = clock();

cout << "Numarul de iteratii:"<<d<< endl;

cout <<fixed << setprecision(6) << "Timpul de executie pentru fib2=" << ((double)finish-start)/CLOCKS\_PER\_SEC << endl;

cout <<"\n";

start = clock();

cout << "fib3 Rezultat:" << fib3(n) << "\n";

finish = clock();

cout << "Numarul de iteratii:"<<e << endl;

cout <<fixed << setprecision(6) << "Timpul de executie pentru fib3 =" << ((double)finish-start)/CLOCKS\_PER\_SEC << endl;

system("pause");

}

long double fib1(int n)

{

c++;

if(n < 2)

{

return n;

}else

{

return fib1(n-1) + fib1(n-2);

}

}

long int fib2(int n )

{

long int i,j,k;

i = 1;

j = 0;

for (k = 0; k < n; k++)

{

j = i + j;

i = j - i;

d++;

}

return j;

}

long int fib3(int n)

{

long int i,j,k,h,t;

i = 1; k = 0;

j = 0; h = 1;

while(n>0)

{

e++;

if(n%2 !=0)

{

t = j \* h;

j = i \* h + j\* h + t;

i = i \* k +t;

}

t = h\*h;

h = 2\*k\*h+t;

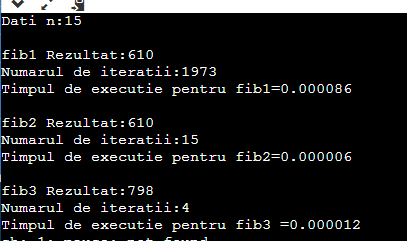
k = k\*k+t;

n = n/2;

}

return j;

}



***Concluzie:***

În această lucrare am înțeles că toate cele 3 metode sunt eficiente și au rezultat bun. Pentru numerele mici prima metodă este cea mai bună, dar pentru un număr mare nu este rentabil pentru că numarul de iteratii creşte în mod exponential. Pentru numerele în intervalul [20.. n] metoda a 3-a este cea mai eficace avînd un număr de iteratii mic,de aceea această metodă o consider ca fiind cea mai rezonabilă.