**MINISTERUL EDUCAŢIEI**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică și Microelectronică**

**Tehnologia Informației**

# Raport

**Disciplina:** Analiza și proiectarea algoritmilor

**Lucrarea de laborator nr. 1**

**Tema:** *“* Analiza algoritmilor (Timpul de execuție al algoritmilor).*”*

**Student:** _____ **Raevschi Grigore TI-231**

**Coordonator:** _____ **Asistent univ. Coșer Cătălin**

Chișinău 2024

# Cuprins

## Scopul lucrării:

1. Analiza empirică a algoritmilor
2. Analiza teoretică a algoritmilor
3. Determinarea complexității temporale și asimptotice a algoritmilor

## Sarcina de bază:

1. Efectuați analiza empirică algoritmilor
2. Determinați relația ce determină complexitatea temporală pentru acești algoritmi
3. Determinați complexitatea asimptotică a algoritmilor
4. Faceți o concluzie asupra lucrării.

## Introducere

Secvența Fibonacci este un set de numere întregi pozitive în care valoarea din poziția N în secvență este calculată prin însumarea celor două numere precedente. Elementele din secvența Fibonacci pot fi calculate folosind următoarea ecuație matematică:

$$fib(n) = fib(n-1) + fib(n-2)$$

Așadar 0 fiind primul element și 1 fiind al doilea element, primele câteva valori ale secvenței ar fi următoarele: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144$

Cele trei algoritmi studiați în acest raport sunt un algoritm iterativ, un algoritm recursiv și un algoritm de programare dinamică. Am folosit C pentru a implementa

acești algoritmi, iar viteza lor în calcularea diferitelor valori din secvența Fibonacci a fost comparată și analizată.

## Big O Analysis

Complexitățile de timp și spațiu pentru toți cei trei algoritmi

| Version | Big O | Space Used |
|---|---|---|
| Iterative | O(n) | O(1) |
| Recursive | $O(2^n)$ | O(n) |
| Dynamic | O(n) | O(n) |

Pentru **implementarea iterativă** a codului, am ales să folosesc atribuiri de variabile pentru a stoca valorile celor două numere Fibonacci precedente.

De fiecare dată când apelăm **funcția recursivă** fib(n), dacă valoarea a N-a pe care o căutăm este altceva decât 0 sau 1, atunci funcția va trebui să facă două apeluri la fib(n – 1) și fib(n - 2). La rândul lor, fiecare dintre aceste două apeluri va trebui să facă alte două apeluri, astfel dublând apelul fib() și rezultând într-o complexitate de timp exponențială de O(2^n).

Algoritmul de implementare dinamică pentru calcularea numerelor Fibonacci optimizează metoda recursivă pentru a crește viteza. Nu trebuie să recalculezi toate numerele anterioare până la valoarea N elimină implicațiile exponențiale de viteză ale algoritmului recursiv. Deoarece acum stocăm toate valorile până la N, complexitatea spațială este O(n). Cu toate acestea, utilizarea valorilor de index pentru a accesa un array are un timp de rulare de O(1), care, împreună cu calculul individual nou pe valoarea N, rezultă într-o complexitate de timp polinomială de O(n).

# Implementarea în C

## Forma iterativă:

```
int fib_iterative(int n) {
    int first = 0;
    int second = 1;
    int val;

    for (int i = 0; i < n; i++) {
        if (i <= 1) {
            val = i;
        } else {
            val = first + second;
            first = second;
            second = val;
        }
    }
    return val;
}
```

## Implementarea recursivă:

```
int fib_recursive(int n) {
    if(n <= 1) {
        return n;
    }
    return fib_recursive(n-1) + fib_recursive(n-2);
}
```

## Implementarea dinamică:

```
#define SIZE 50000
static long fib_table[SIZE];


unsigned long fib_d(int n) {
    if(n <= 1) {
        return n;
    } else if (fib_table[n] != -1){
        return fib_table[n];
    }

    fib_table[n] = fib_d(n-1) + fib_d(n-2);
    return fib_table[n];
}

int fib_dynamic(int n) {

    for(int i = 0; i < SIZE; i++) {
        fib_table[i] = -1;
    }

    fib_table[0] = 0;
    fib_table[1] = 1;

    int val;
    for (int i = 0; i < n; i++) {
        val = fib_d(i);
    }

    return val;
}
```

# Execuția programului

```
Menu:
1. Set number of iterations (N)
2. Set type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic)
3. Set print option (0 for no print, 1 for print N val, 2 for print all calculations)
4. Run analysis
5. Exit
Enter your choice: 1
Enter number of iterations (N): 50
Menu:
1. Set number of iterations (N)
2. Set type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic)
3. Set print option (0 for no print, 1 for print N val, 2 for print all calculations)
4. Run analysis
5. Exit
Enter your choice: 2
Enter type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic): 2
Menu:
1. Set number of iterations (N)
2. Set type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic)
3. Set print option (0 for no print, 1 for print N val, 2 for print all calculations)
4. Run analysis
5. Exit
Enter your choice: 3
Enter print option (0 for no print, 1 for print N val, 2 for print all calculations): 2
Menu:
1. Set number of iterations (N)
2. Set type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic)
3. Set print option (0 for no print, 1 for print N val, 2 for print all calculations)
4. Run analysis
5. Exit
Enter your choice: 4
Execution time: 0.001000 seconds
```

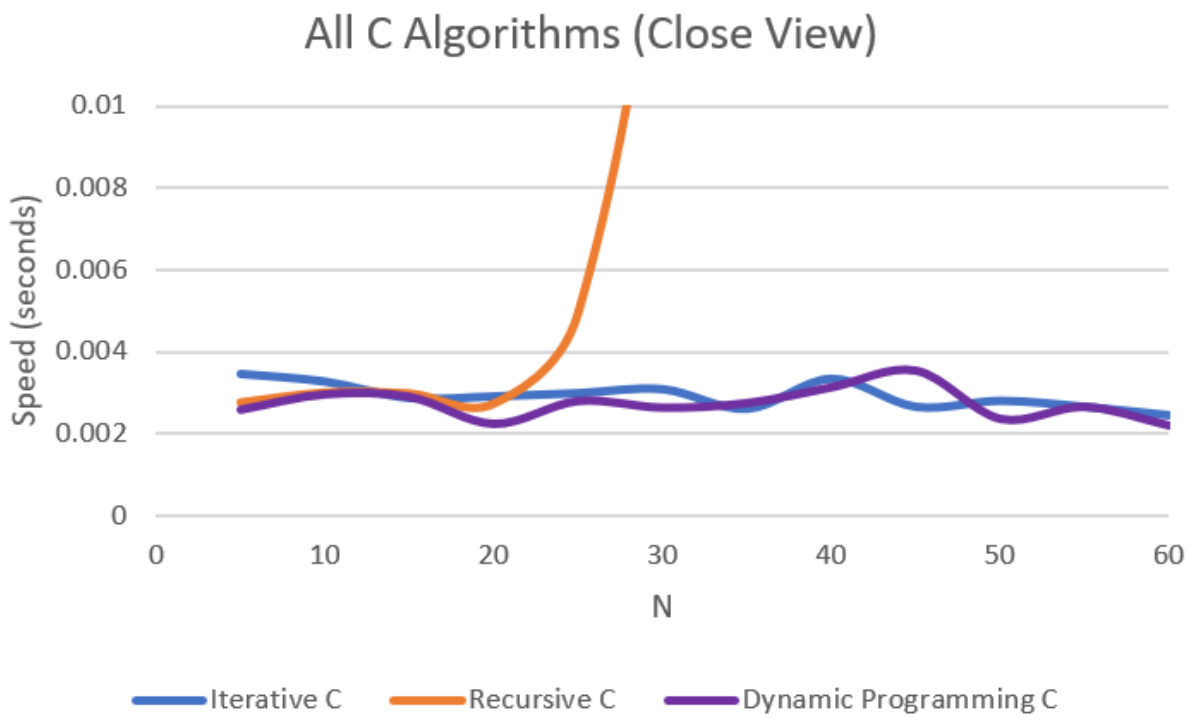
*Figura 1: Execuția programului*

# Analiza empirică

```
    Menu:
1. Set number of iterations (N)
2. Set type of algorithm (0 for iterative, 1 for recursive, 2 for dynamic)
3. Set print option (0 for no print, 1 for print N val, 2 for print all calculations)
4. Run analysis
5. Empiric Analysis
6. Exit
Enter your choice: 5
Empiric Analysis:
n = 10, Algorithm: Iterative
fib(10): 34
Execution time: 0.000001 seconds
n = 10, Algorithm: Recursive
fib(10): 34
Execution time: 0.000100 seconds
n = 10, Algorithm: Dynamic
fib(10): 34
Execution time: 0.000001 seconds

n = 40, Algorithm: Iterative
fib(40): 102334155
Execution time: 0.000001 seconds
n = 40, Algorithm: Recursive
fib(40): 102334155
Execution time: 0.410000 seconds
n = 40, Algorithm: Dynamic
fib(40): 102334155
Execution time: 0.000001 seconds
```

## Grafic de reprezentare a vitezei de execuție



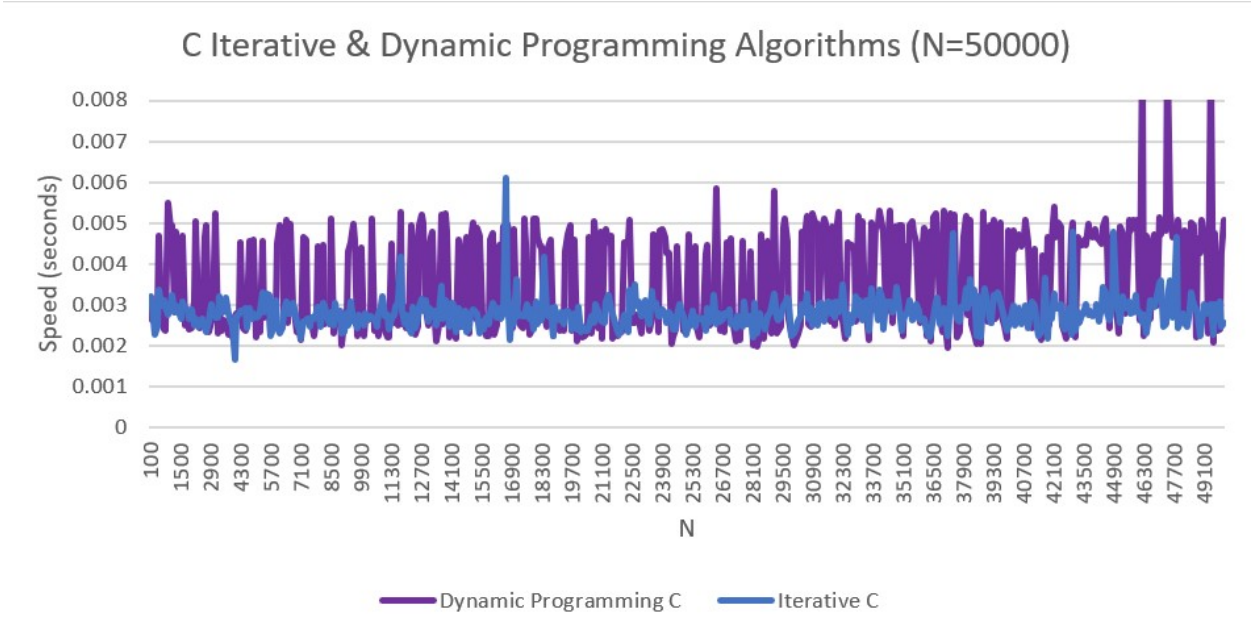

*Figură 2:Timpul de executare bazat la numărul de iterații*

Pentru programul în C, apelul recursiv a depășit timpul limită odată ce secvența Fibonacci a trecut de N = 45.

Versiunile Iterativă și Dinamică Următoarele grafice arată comparația între metodele iterativă și dinamică pentru fiecare program, rulate alături de metoda recursivă pentru un N = 60.

Se poate observa că variațiile pentru metodele iterativă și dinamică par să rămână în aceeași gamă una față de cealaltă, ceea ce are sens având în vedere că ambele au o complexitate de timp O(n). După ce am rulat scripturile de testare care au exclus metoda recursivă, am reușit să iterez până la un număr Fibonacci de N = 50000 în

câteva secunde. În urma acestei teste, putem vedea diferențele subtile între cele două algoritme pe măsură ce valoarea N crește.




*Figură 3: Execuția iterativă și dinamica*

# Concluzie

Efectuarea acestor teste și vizualizarea lor mi-au permis să văd mai bine diferența dintre algoritmii iterativi, recursivi și de programare dinamică în ceea ce privește implementarea, complexitatea de timp și complexitatea de spațiu. Pot înțelege acum mai pe deplin puterea Big O.

# Bibliografie


1. „Difference between Recursion and Iteration.” GeeksforGeeks, GeeksforGeeks, 11 Feb. 2023, https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/.
2. „FibSeriesResearch.”GitHub https://github.com/marianpadron/FibSeriesResearch/tree/main?tab=readme-ov-file