



UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ANÁLISE DE ALGORITMOS - DCC606

01

ALGORITMO

FIBONACCI

MARCIA GABRIELLE BONIFÁCIO DE OLIVEIRA - 2020011319
PAULO FERREIRA DA SILVA JÚNIOR - 2019034400





SEQUÊNCIA DE FIBONACCI

CONCEITO E DEFINIÇÃO

Ela recebe seu nome do matemático italiano Leonardo de Pisa, mais conhecido como Fibonacci, que a introduziu no mundo ocidental em seu livro Liber Abaci no início do século XIII.

Conceito:

A sequência de Fibonacci é uma série de números onde cada número é a soma dos dois anteriores. Inicia-se com 0 e 1, e é definida pela fórmula:

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

para $n > 1$, com $F(0) = 0$ e $F(1) = 1$.



ALGORITMO ITERATIVO:

Calcula a sequência de maneira linear, utilizando um loop para acumular os valores dos termos anteriores até o termo desejado.

FUNÇÃO DE CUSTO

$$T(n)=4n-8$$

COMPLEXIDADE

Tem complexidade de tempo $O(n)$, tornando-o mais eficiente que o recursivo.

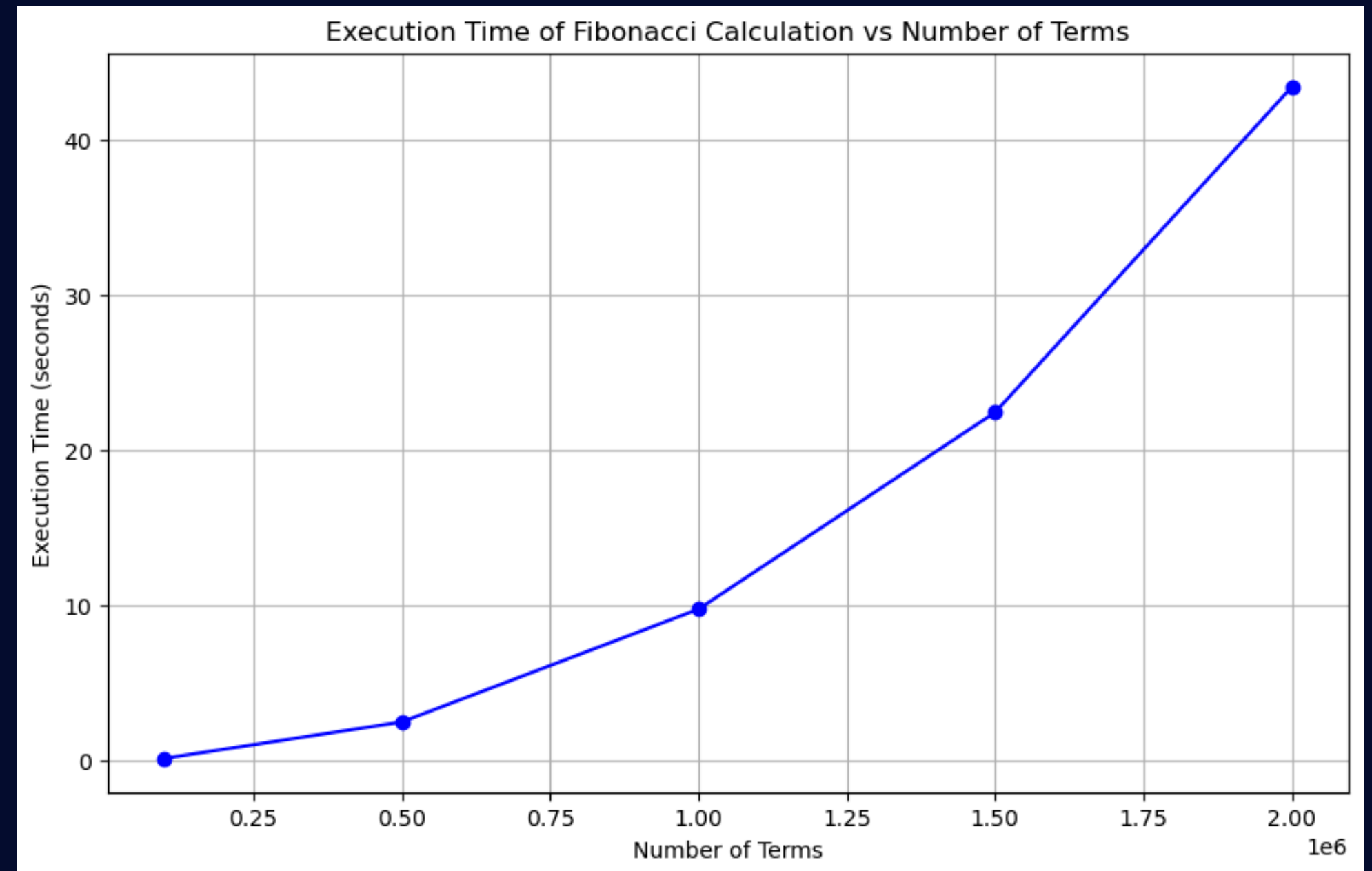


```
1 #include <stdio.h>
2 #include <gmp.h>
3 #include <time.h>
4
5 int main() {
6     FILE *fp = fopen("fibonacci_times.csv", "w"); // Arquivo para salvar os resultados
7     fprintf(fp, "Terms,ExecutionTime\n");
8
9     int terms[] = {100000, 500000, 1000000, 1500000, 2000000}; // Tamanhos de entrada específicos
10    int numTests = 5; // Número de entradas
11
12    for (int test = 0; test < numTests; test++) {
13        int n = terms[test];
14        mpz_t t1, t2, nextTerm, sum;
15        mpz_init(t1);
16        mpz_init(t2);
17        mpz_init(nextTerm);
18        mpz_init(sum);
19        mpz_set_ui(t1, 0);
20        mpz_set_ui(t2, 1);
21        mpz_add(nextTerm, t1, t2);
22        mpz_add(sum, t1, t2);
23
24        clock_t start, end;
25        double cpu_time_used;
26
27        start = clock();
28
29        for (int i = 3; i <= n; ++i) {
30            mpz_add(sum, sum, nextTerm);
31            mpz_set(t1, t2);
32            mpz_set(t2, nextTerm);
33            mpz_add(nextTerm, t1, t2);
34        }
35
36        end = clock();
37        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
38
39        fprintf(fp, "%d,%f\n", n, cpu_time_used); // Salva os resultados no arquivo
40
41        // Limpeza das variáveis GMP
42        mpz_clear(t1);
43        mpz_clear(t2);
44        mpz_clear(nextTerm);
45        mpz_clear(sum);
46    }
47
48    fclose(fp);
49    return 0;
50 }
51
```

EXECUÇÃO DO ALGORITMO

Utilizamos 5 diferentes entradas

```
fibonacci_times.csv > data
1  Terms,ExecutionTime
2  1000000,0.081000
3  5000000,2.427000
4  10000000,9.703000
5  15000000,22.401000
6  20000000,43.393000
7
```



EXPONENCIAÇÃO DE MATRIZES:

Uma abordagem mais avançada que utiliza multiplicação de matrizes para calcular a sequência em $O(\log n)$ tempo. Esta técnica é ideal para grandes valores de n , pois reduz significativamente o número de operações necessárias.

COMPLEXIDADE

$O(\log n)$

FUNÇÃO DE CUSTO

$$T(n) = c_1 + c_2 \cdot \log(n)$$

c_1 = Representa o custo das operações de inicialização, que é constante e independente de n .

c_2 = Representa o custo de cada multiplicação de matrizes, que é $O(1)$, multiplicado pelo número de multiplicações de matrizes necessárias, que é proporcional a $\log(n)$ devido à exponenciação rápida.



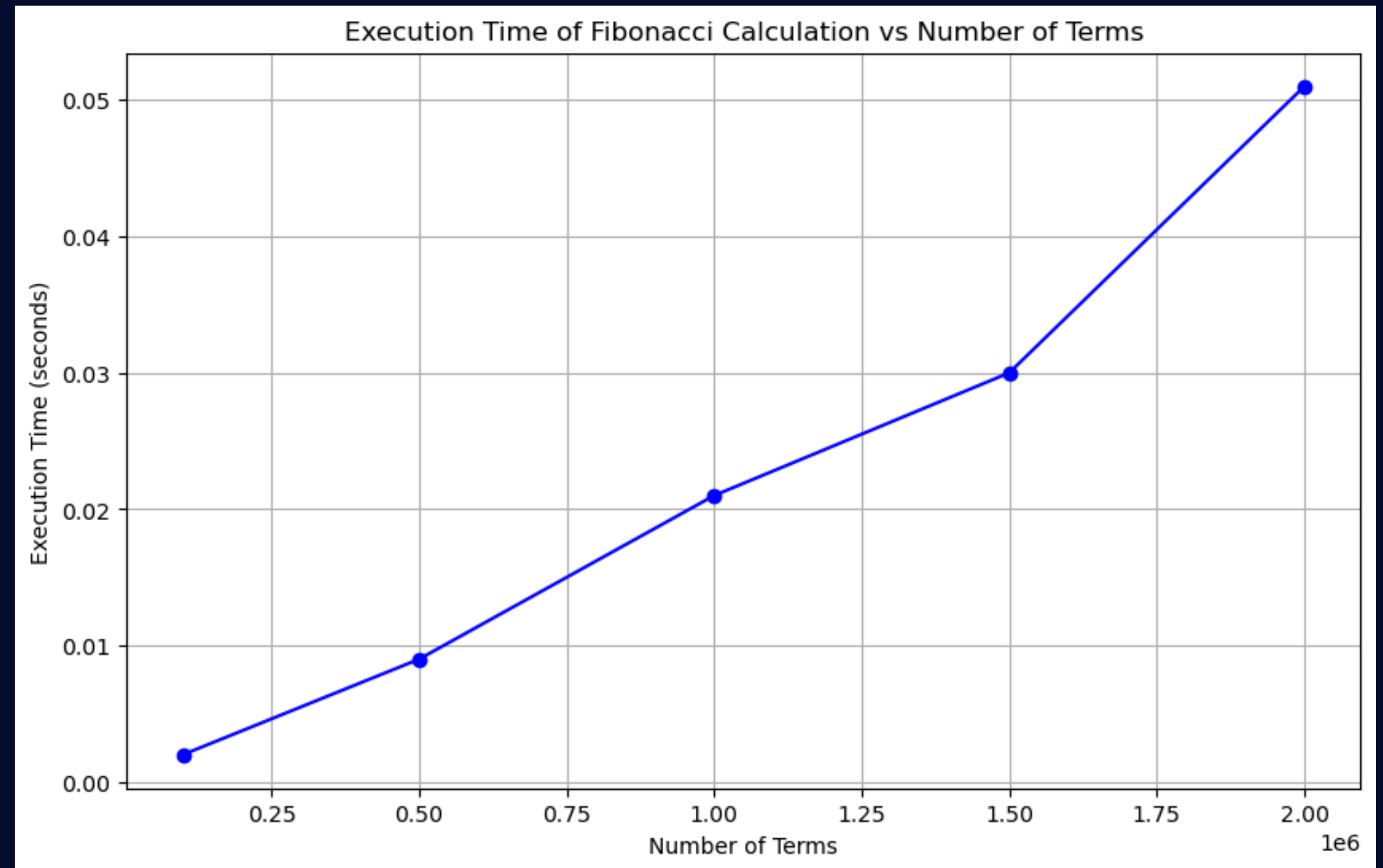
```
C fibonacci_melhorado.c
1  #include <stdio.h>
2  #include <gmp.h>
3  #include <time.h>
4
5  void multiply_matrices(mpz_t F[2][2], mpz_t M[2][2]) {
6      mpz_t a, b, c, d;
7      mpz_init(a); mpz_init(b); mpz_init(c); mpz_init(d);
8
9      mpz_mul(a, F[0][0], M[0][0]);
10     mpz_addmul(a, F[0][1], M[1][0]);
11
12     mpz_mul(b, F[0][0], M[0][1]);
13     mpz_addmul(b, F[0][1], M[1][1]);
14
15     mpz_mul(c, F[1][0], M[0][0]);
16     mpz_addmul(c, F[1][1], M[1][0]);
17
18     mpz_mul(d, F[1][0], M[0][1]);
19     mpz_addmul(d, F[1][1], M[1][1]);
20
21     mpz_set(F[0][0], a);
22     mpz_set(F[0][1], b);
23     mpz_set(F[1][0], c);
24     mpz_set(F[1][1], d);
25
26     mpz_clear(a); mpz_clear(b); mpz_clear(c); mpz_clear(d);
27 }
28
29 void power_matrix(mpz_t F[2][2], int n) {
30     if (n == 0 || n == 1)
31         return;
32
33     mpz_t M[2][2];
34     mpz_init(M[0][0]); mpz_init(M[0][1]);
35     mpz_init(M[1][0]); mpz_init(M[1][1]);
36
37     mpz_set_ui(M[0][0], 1); mpz_set_ui(M[0][1], 1);
38     mpz_set_ui(M[1][0], 1); mpz_set_ui(M[1][1], 0);
39
40     power_matrix(F, n / 2);
41     multiply_matrices(F, F);
42
43     if (n % 2 != 0)
44         multiply_matrices(F, M);
45
46     mpz_clear(M[0][0]); mpz_clear(M[0][1]);
47     mpz_clear(M[1][0]); mpz_clear(M[1][1]);
48 }
```

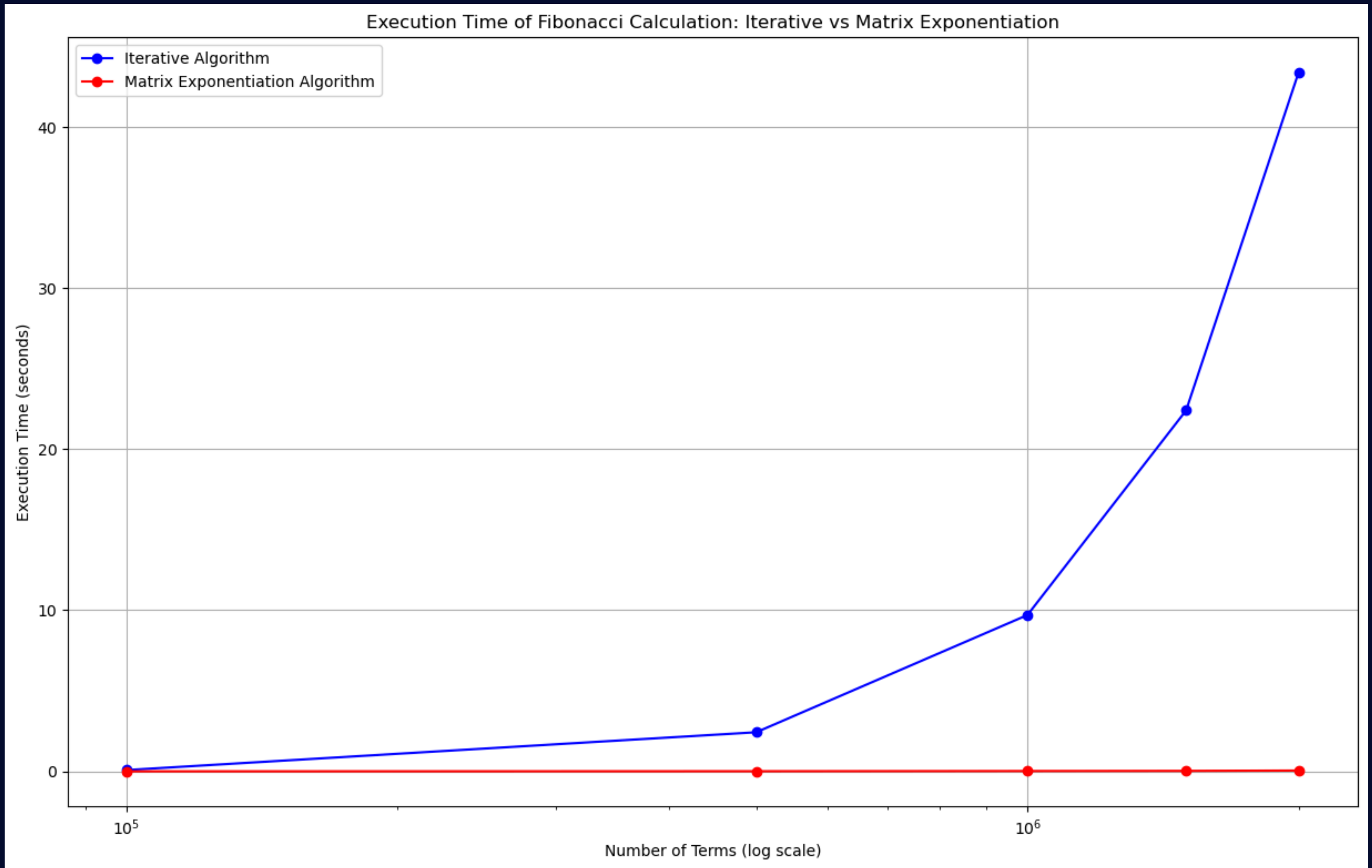
EXECUÇÃO DO ALGORITMO

Utilizamos 5 diferentes entradas

```
fibonacci_times_matrix.csv > data
```

```
1 Terms,ExecutionTime
2 100000,0.002000
3 500000,0.009000
4 1000000,0.021000
5 1500000,0.030000
6 2000000,0.051000
7
```







OBRIGADO(A)!

