ATIVIDADE - ESTRUTURA DE DADOS

Complexidade de algoritmos (Fibonacci)

Aluno: *Roberth Fontes Ovelar*

1) Implemente versões iterativas e recursivas da função fib obtendo também o tempo de execução (note que a versão recursiva já está pronta).

Versão iterativa:

```c
#include <stdio.h>
#include <time.h>

float fibIterativo(int n){
    float vetFib[n];

    vetFib[0] = 0;
    vetFib[1] = 1;
    vetFib[2] = 2;
    for(int i = 3; i <= n; i++){
        vetFib[i] = 0.7 * (vetFib[i-1] + vetFib[i-2]);
    }
    return vetFib[n];
}

int main(){
    int n;
    clock_t t;

    scanf("%d", &n);
    t = clock();
    printf("%.2f\n", fibIterativo(n));
    printf("%f segundos\n", ((float) clock() - t)/((CLOCKS_PER_SEC)));

    return 0;
}
```

Versão recursiva:

```c
#include <stdio.h>
#include <time.h>

float fib(int n){
    if (n <= 2){
        return n;
    }
    return 0.7 * (fib(n - 1) + fib(n - 2));
}

int main(){
    int n;
    clock_t t;

    scanf("%d", &n);
    t = clock();
    printf("%.2f\n", fib(n));
    printf("%f segundos\n", ((float) clock() - t)/((CLOCKS_PER_SEC)));

    return 0;
}
```

2) Execute os dois medindo o tempo de execução para n = 10, 20, 30, 35, 40, 45 e 50.

Valores	Versão iterativa (segundos)	Versão recursiva (segundos)
10	0	0
20	0	0
30	0	0,002s
35	0	0,066s
40	0	0,646s
45	0	7,286s
50	0	81,373s

3) Faça um gráfico para cada um dos algoritmos tendo *n* nas ordenadas (eixo x) e *tempo de execução* nas abscissas (eixo y).

Gráfico da função iterativa:

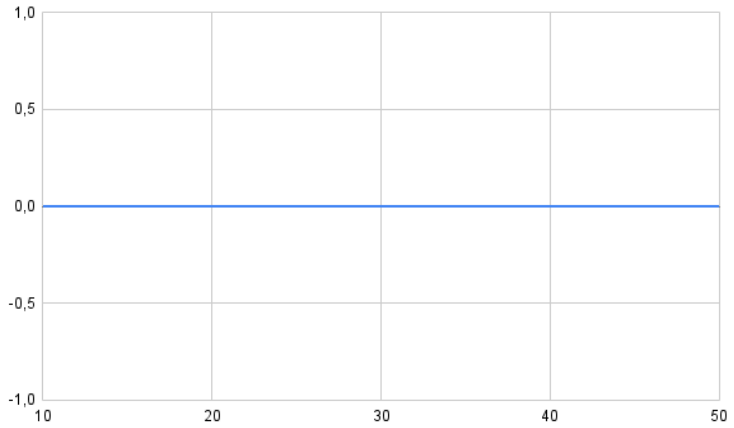
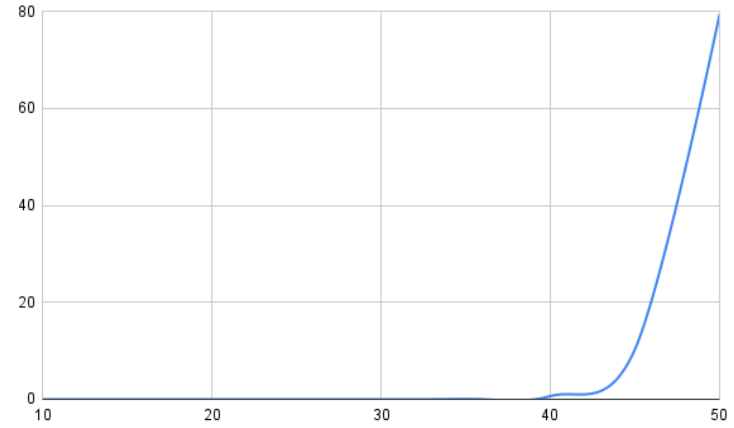


Gráfico da função recursiva:



4) Sem executar estime o tempo gasto aproximado por cada um dos algoritmos quando n = 200:

Estimo que a função iterativa irá gastar 0 segundos como em todos os casos testados anteriormente, enquanto a função recursiva talvez sobrecarregue a memória e não seja executada até o fim.