PRÁTICA 3 – Trabalho 2 Exercício 3: Torres de Hanói

Nome: Davi Gabriel Domingues

Número USP: 15447497

O trabalho consiste na implementação em C++ de uma função que implemente o famigerado algoritmo de resolução das Torres de Hanói, a partir do princípio de códigos computacionalmente recursivos. Tem – se como procedimento/função padrão para esta atividade a função recHanoi().

O objetivo é ilustrar a aplicação do algoritmo recursivo em si, além de se verificar as implicações nas análises da complexidade de tempo e de uso de memória provocados pelo programa durante a sua execução.

Sendo assim, foram elaboradas as seguintes respostas apropriadas para cada um dos itens:

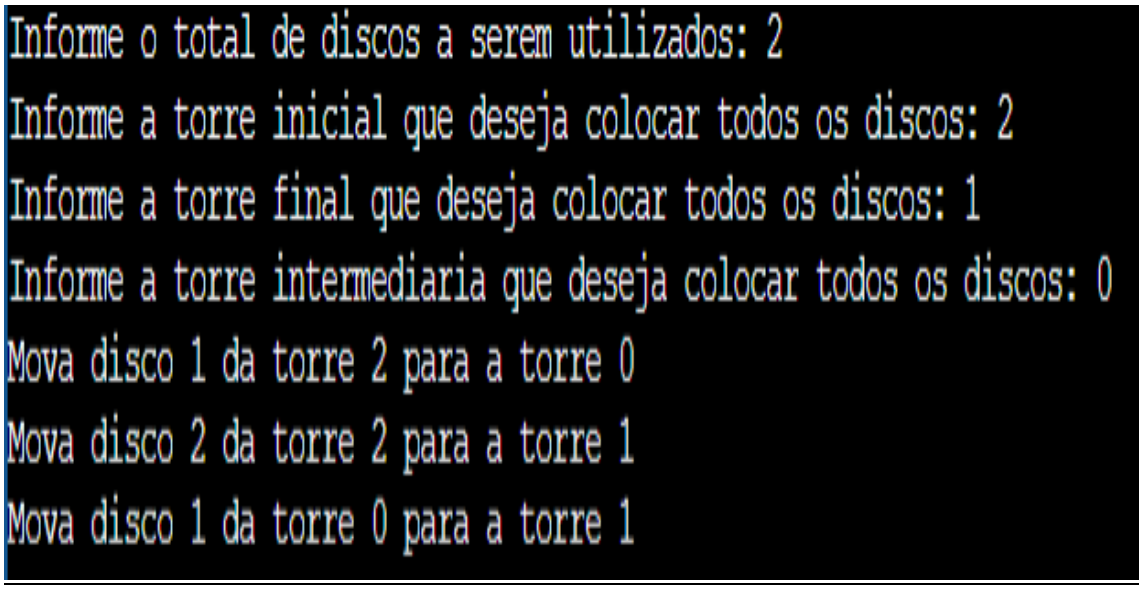
a) O código, implementado em C++, do programa para resolução do problema das Torres de Hanói baseado no procedimento *recHanoi*( ) se encontra a seguir:

```
#include <iostream>
using namespace std;

void recHanoi(int discos, int inicio, int fim, int temporario){
    if (discos > 0){
        recHanoi(discos - 1, inicio, temporario, fim);
        cout<<"Mova disco "<<discos<<" da torre "<<inicio<<" para a torre "<<fim<<endl;
        recHanoi(discos - 1, temporario, fim, inicio);
    }
}
```

```
int main(){
    int totalDiscos, torreInicial, torreFinal, torreTemporaria;

    cout<<"Informe o total de discos a serem utilizados: ";
    cin>>totalDiscos;

    cout<<"Informe a torre inicial que deseja colocar todos os discos: ";
    cin>>torreInicial;

    cout<<"Informe a torre final que deseja colocar todos os discos: ";
    cin>>torreFinal;

    cout<<"Informe a torre intermediaria que deseja colocar todos os discos: ";
    cin>>torreTemporaria;
    recHanoi(totalDiscos, torreInicial, torreFinal, torreTemporaria);

return 0;
}
```

b) Tem – se, a seguir, uma série de valores de n, que representam o número de discos dispostos inicialmente nas Torres de Hanói, pequenos, a fim de se checar o quão correta está a solução disposta em C++:

Para n = 2:



Para n = 3:

```
Informe o total de discos a serem utilizados: 3
Informe a torre inicial que deseja colocar todos os discos: 2
Informe a torre final que deseja colocar todos os discos: 1
Informe a torre intermediaria que deseja colocar todos os discos: 0
Mova disco 1 da torre 2 para a torre 1
Mova disco 2 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 0
Mova disco 3 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 2
Mova disco 2 da torre 0 para a torre 1
Mova disco 1 da torre 2 para a torre 1
```

Para n = 4:

```
Informe o total de discos a serem utilizados: 4
Informe a torre inicial que deseja colocar todos os discos: 2
Informe a torre final que deseja colocar todos os discos: 1
Informe a torre intermediaria que deseja colocar todos os discos: 0
Mova disco 1 da torre 2 para a torre 0
Mova disco 2 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 1
Mova disco 3 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 2
Mova disco 2 da torre 1 para a torre 0
Mova disco 1 da torre 2 para a torre 0
Mova disco 4 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 1
Mova disco 2 da torre 0 para a torre 2
Mova disco 1 da torre 1 para a torre 2
Mova disco 3 da torre 0 para a torre 1
Mova disco 1 da torre 2 para a torre 0
Mova disco 2 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 1
```

Para n = 5:

```
Informe o total de discos a serem utilizados: 5
Informe a torre inicial que deseja colocar todos os discos: 2
Informe a torre final que deseja colocar todos os discos: 1
Informe a torre intermediaria que deseja colocar todos os discos: 0
Mova disco 1 da torre 2 para a torre 1
Mova disco 2 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 0
Mova disco 3 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 2
Mova disco 2 da torre 0 para a torre 1
Mova disco 1 da torre 2 para a torre 1
Mova disco 4 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 0
Mova disco 2 da torre 1 para a torre 2
Mova disco 1 da torre 0 para a torre 2
Mova disco 3 da torre 1 para a torre 0
Mova disco 1 da torre 2 para a torre 1
Mova disco 2 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 0
Mova disco 5 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 2
Mova disco 2 da torre 0 para a torre 1
Mova disco 1 da torre 2 para a torre 1
Mova disco 3 da torre 0 para a torre 2
Mova disco 1 da torre 1 para a torre 0
Mova disco 2 da torre 1 para a torre 2
Mova disco 1 da torre 0 para a torre 2
Mova disco 4 da torre 0 para a torre 1
```

```
Mova disco 1 da torre 2 para a torre 1
Mova disco 2 da torre 2 para a torre 0
Mova disco 1 da torre 1 para a torre 0
Mova disco 3 da torre 2 para a torre 1
Mova disco 1 da torre 0 para a torre 2
Mova disco 2 da torre 0 para a torre 1
Mova disco 1 da torre 2 para a torre 1
```

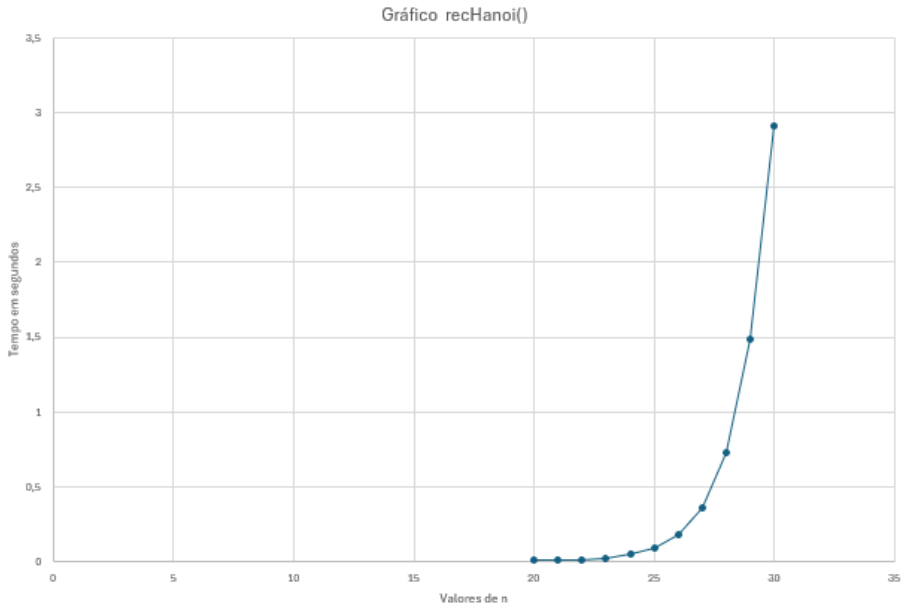
c) A análise experimental de tempo do programa se embasa no seguinte código:

```
#include <iostream>
#include <ctime>
using namespace std;

void recHanoi(int discos, int inicio, int fim, int temporario){
    if (discos > 0){
        recHanoi(discos - 1, inicio, temporario, fim);
        recHanoi(discos - 1, temporario, fim, inicio);
    }
}
```

```
int main(){
    clock_t time1, time2;
    int totalDiscos;
    double tempoFinal = 0.0;

    cout<<"Informe o total de discos a serem utilizados (ha somente 3 torres): ";
    cin>>totalDiscos;

    time1 = clock();
    recHanoi(totalDiscos, 0, 1, 2);
    time2 = clock();

    tempoFinal = difftime(time2, time1)/CLOCKS_PER_SEC;
    cout<<"\n\n"<<tempoFinal;

return 0;
}
```

Com isso, gera – se o gráfico de n x tempo de execução a seguir:

valores de n	tempo de execução
20	0,0052
21	0,0105
22	0,0119
23	0,0234
24	0,046
25	0,091
26	0,1822
27	0,3629
28	0,7278
29	1,4808
30	2,909



d) A análise experimental sobre o número de variáveis criadas pelo programa se dá pelo programa a seguir:

```
#include <iostream>
#include <ctime>
using namespace std;
int totalVariaveisPrograma = 0;

void recHanoi(int discos, int inicio, int fim, int temporario){
    totalVariaveisPrograma += 4;
    if (discos > 0){
        recHanoi(discos - 1, inicio, temporario, fim);
        recHanoi(discos - 1, temporario, fim, inicio);
    }
}
```

```
int main(){
    int totalDiscos, torreInicial, torreFinal, torreTemporaria;
    totalVariaveisPrograma += 4;

    cout<<"Informe o total de discos a serem utilizados: ";
    cin>>totalDiscos;

    cout<<"Informe a torre inicial que deseja colocar todos os discos: ";
    cin>>torreInicial;

    cout<<"Informe a torre final que deseja colocar todos os discos: ";
    cin>>torreFinal;

    cout<<"Informe a torre intermediaria que deseja colocar todos os discos: ";
    cin>>torreTemporaria;

    recHanoi(totalDiscos, torreInicial, torreFinal, torreTemporaria);
    cout<<totalVariaveisPrograma;

return 0;
}
```

Dessa forma, é visível que o número de variáveis criadas fornece um bom indicativo para o uso da memória. Observa – se que, o padrão da resolução das Torres de Hanói é definido previamente pela equação 2^(n) – 1, sem a análise computacional.

Como cada função é chamada recursivamente, então cada uma é referenciada, segundo o algoritmo construído, 2*2(n) vezes, ou seja, 2^(n + 1) vezes. Entretanto, na última vez em que a segunda referência à recHanoi() é feita, a recursão é interrompida e executada uma vez apenas e não duas, dado que o total de discos a serem reposicionados se tornou nulo, assim, o total de vezes, na verdade, é igual a 2^(n + 1) – 1.

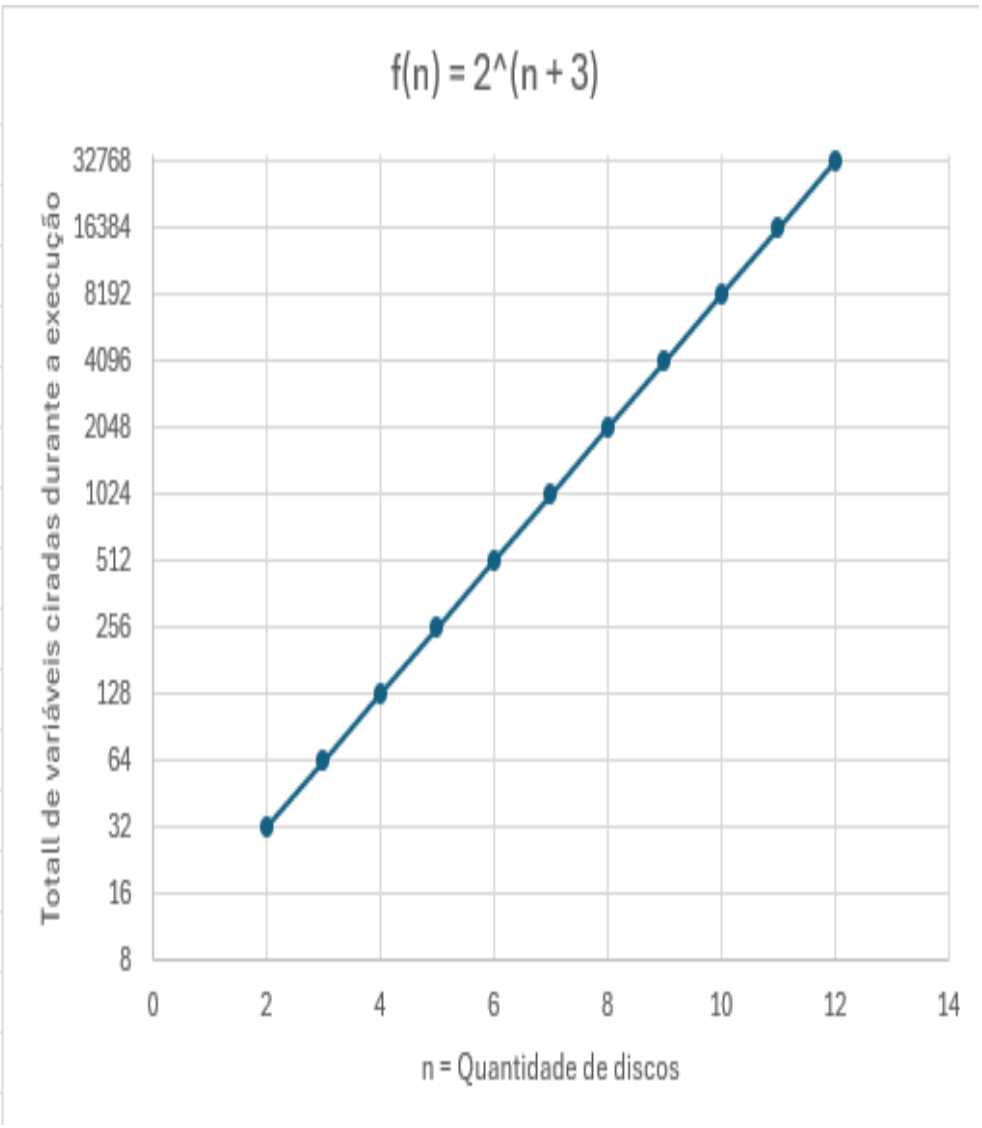
Considerando que a cada chamada foram criadas quatro variáveis, além de se considerar as quatro variáveis iniciais existentes na main(), tem – se que o total verdadeiro de variáveis criadas é:

4*(2^(n + 1) – 1) + 4 → 2^(n + 3) – 4 + 4 → 2^(n + 3) variáveis

Logo, a função que retrata o total de variáveis criadas é f(n) = 2^(n + 3).

Nesse panorama, tem – se este gráfico de n x número de variáveis criadas associado:

n	f(n) = 2^(n+3)
2	32
3	64
4	128
5	256
6	512
7	1024
8	2048
9	4096
10	8192
11	16384
12	32768

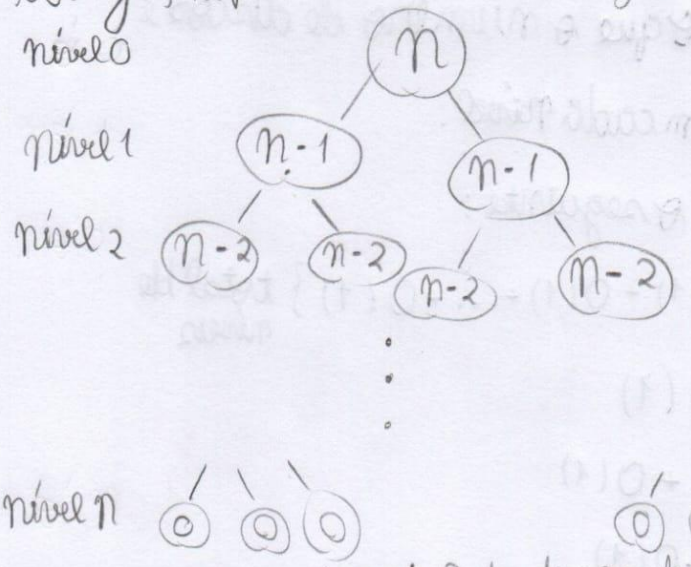


Obs: O gráfico gerado em Excel está na escala logarítmica de base 2, por isso a aparência de uma reta. Se não utilizarmos essa escala, tal função seria, na escala usual, uma curva ascendente, semelhante ao que se vê no item “c)”.

e) A análise dos resultados experimentais obtidos nos itens “c)” e “d)” permite uma comparação com os resultados teóricos da análise assintótica para tempo e memória, sendo tal relação a seguinte:

e) Análise assintótica para o tempo:

É possível retratar o tempo de execução do código, a partir da árvore a seguir:



Observação: n é o total de discos declarados na main().

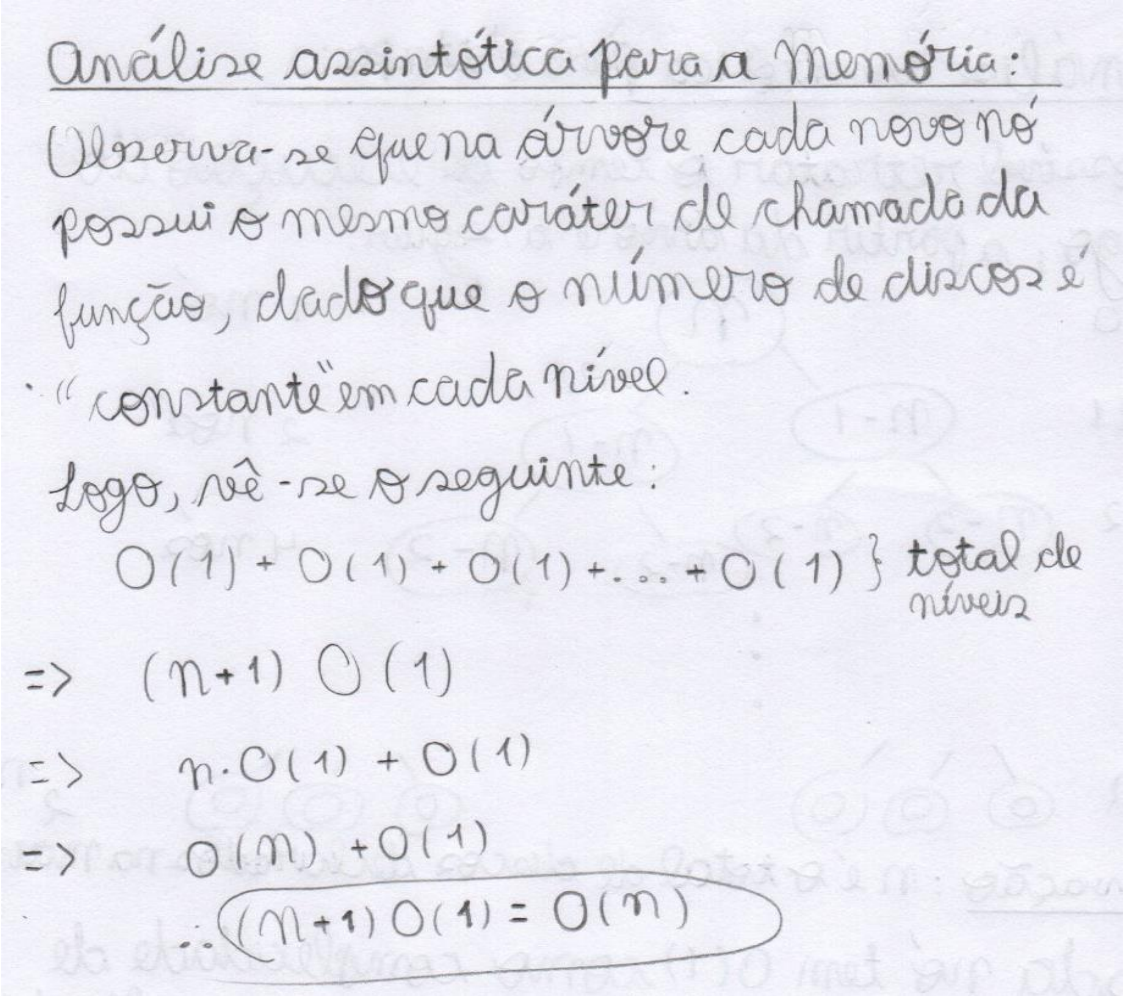
Se cada nó tem O(1) como complexidade de execução, então 2^{n+1} nós possui esta complexidade na notação "O":

$$2^{n+1} \cdot O(1) = O(2^{n+1})$$

$$2^{n+1} \cdot O(1) = O(2^n \cdot 2)$$

$$2^{n+1} \cdot O(1) = 2 \cdot O(2^n)$$

$$\therefore 2^{n+1} \cdot O(1) = O(2^n)$$



Dessa forma, percebe – se que a complexidade de tempo bate com a descrição fornecida em “c)”, respaldada pelos dados obtidos e analisados nos devidos valores de n. Entretanto, vê – se que no caso da memória, há uma disparidade evidente: a quantidade de variáveis criadas segue uma complexidade O(2^n), diferentemente do gasto de memória durante a execução da função recHanoi(), o qual é de complexidade O(n), como demonstrado acima.

Ou seja, há um gasto de memória linear, mas com um total de variáveis envolvidas que cresce exponencialmente, assim como o tempo de execução do algoritmo recursivo em questão.