Fatorial

```
int fatorialRecursivo(int n) {
    if(n == 0) {
        return 1;
    }
    else {
        return n * fatorialRecursivo(n-1);
    }
}
```

$T(n) = 0$ para $n = 0$

$T(n) = 1*T(n-1) + 1$ para $n > 0$

$T(n) \in \Theta(n)$

Aprenderemos a resolver ao longo da aula.

```
int fatorialIterativo(int n) {
    int res = 1;
    for (int i=1; i<=n; i++){
        res = res * i;
    }
    return res;
}
```

operacoes(m) = m

operacoes(n) $\in \Theta(n)$

Fibonacci

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

$T(n) = 0$ para $n \leq 1$

$T(n) = T(n-1) + T(n-2) + 1$ para $n > 1$

$T(n) \in O(1{,}618034^n)$

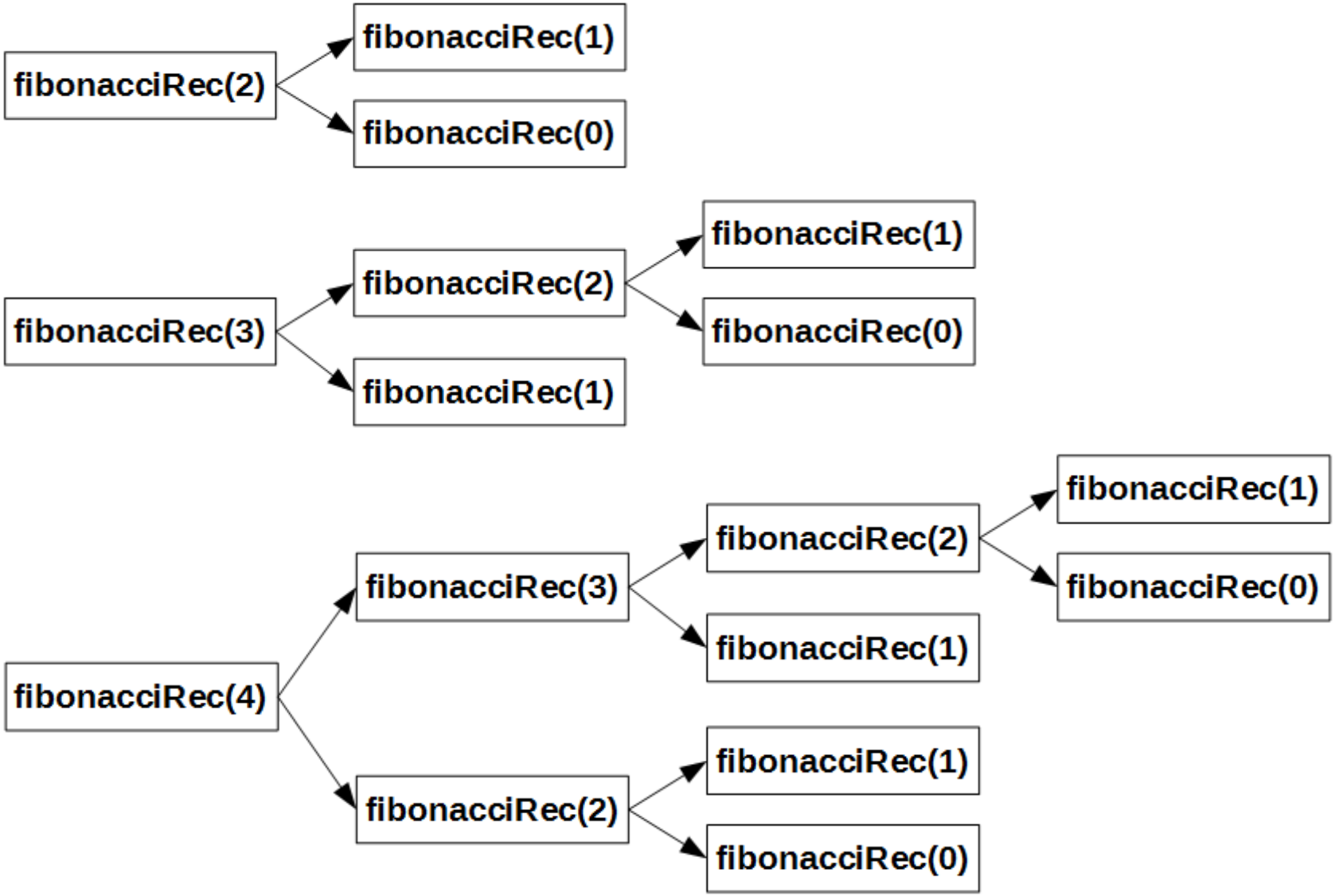
O número de operações realizadas é um número de Fibonacci.

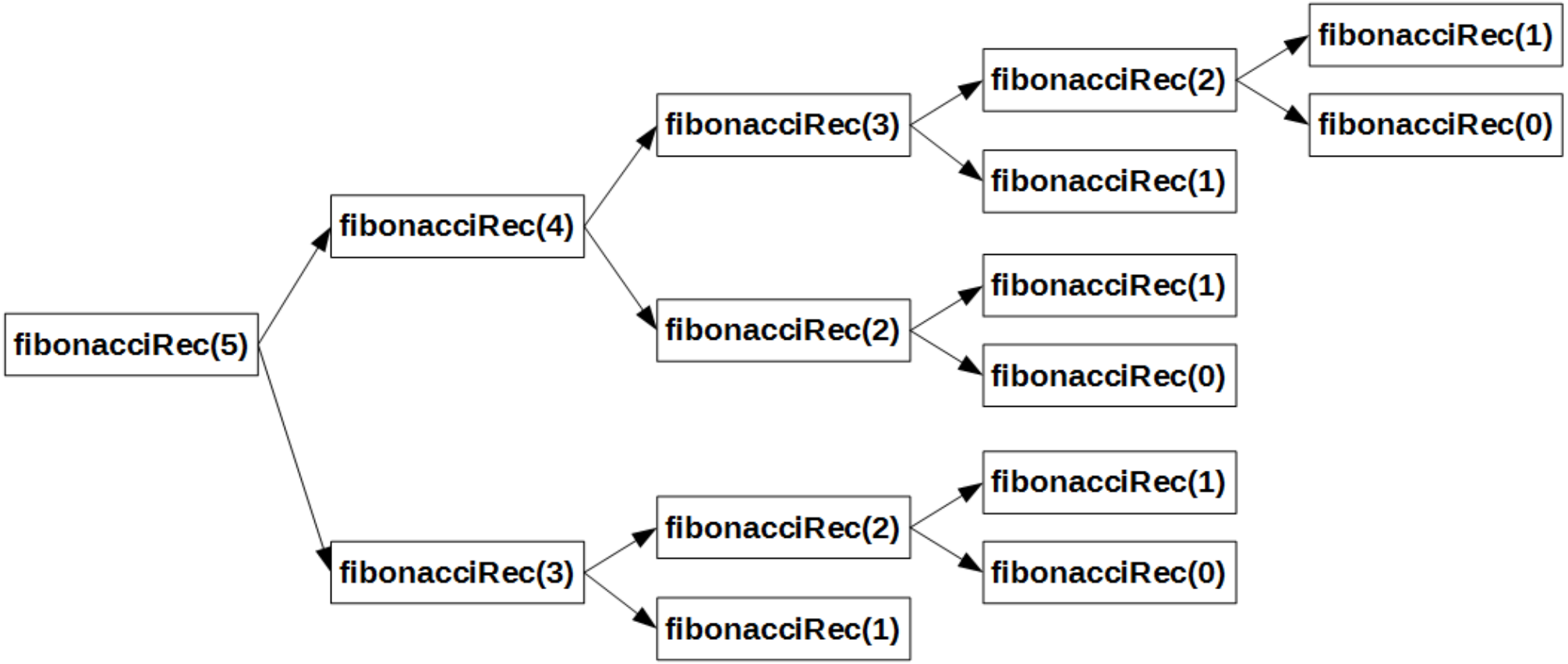
```
int fibonacciIter(int n) {
    if(n <= 1) return n;
    else {
        int[] res = new int[n+1];
        res[0] = 0;
        res[1] = 1;
        for (int i=2; i<=n; i++){
            res[i] = res[i-1] + res[i-2];
        }
        return res[n];
    }
}
```

operacoes(n) = n – 1

operacoes(n) $\in \Theta(n)$

Fibonacci – processamento




fibonacciRec(5)
fibonacciRec(4)
fibonacciRec(3)
fibonacciRec(2)
fibonacciRec(1)
fibonacciRec(0)
fibonacciRec(2)
fibonacciRec(1)
fibonacciRec(0)
fibonacciRec(3)
fibonacciRec(2)
fibonacciRec(1)
fibonacciRec(0)
fibonacciRec(1)

Busca Sequencial

```
int sequencialRec1(int valor, int[] vetor, int n) {
    if(n == 1) {
        if(vetor[0] == valor) return 0;
        else return -1;
    } else {
        int index = sequencialRec1(valor, vetor, n-1);
        if(index < 0) {
            if(vetor[n-1] == valor) {
                index = n-1;
            }
        }
        return index;
    }
}
```

Melhor caso

$T(n) = 1$ para $n = 1$

$T(n) = 1*T(n-1) + 0$ para $n > 1$

$\mathbf{T(n) \in \Theta(1)}$

Pior caso

$T(n) = 1$ para $n = 1$

$T(n) = 1*T(n-1) + 1$ para $n > 1$

$\mathbf{T(n) \in \Theta(n)}$

Podemos dizer que este algoritmo $\mathbf{\in O(n)}$

```java
int sequencialRec2(int valor, int[]
vetor, int n) {
    if(n == 1) {
        if(vetor[0]==valor) return 0;
        else return -1;
    } else {
        if(vetor[n-1]==valor) return n-1;
        else return sequencialRec2(valor,
vetor, n-1);
    }
}
```

Melhor caso

T(n) = 1 para n = 1

T(n) = 1 para n > 1

T(n) ϵ Θ(1)

Pior caso

T(n) = 1 para n = 1

T(n) = 1*T(n-1) + 1 para n > 1

T(n) ϵ Θ(n)

Podemos dizer que este algoritmo **ϵ O(n)**

```java
int sequencialIter(int valor, int[]
vetor) {
    for (int i=0;i<vetor.length;i++){
        if (vetor[i] == valor) return i;
    }
    return -1;
}
```

Melhor caso

operacoes(n) = 1

T(n) ϵ Θ(1)

Pior caso

operações(n) = n

T(n) ϵ Θ(n)

Busca Binaria

```
int binaria(int valor, int[] vetor, int esq, int dir) {
      int meio = (esq + dir)/2;
      if(esq <= dir) {
           if(valor > vetor[meio]) {
                esq = meio + 1;
                return binaria(valor, vetor, esq, dir);
           } else if(valor < vetor[meio]) {
                dir = meio - 1;
                return binaria(valor, vetor, esq, dir);
           } else {
                return meio;
           }
      } else {
           return -1;
      }
 }
```

Melhor caso:

T(n) = 0 para n = 0

T(n) = + 2 para n > 1

Pior caso:

T(n) = 0 para n = 0

T(n) = 1*T((n-1)/2) + 2 para n > 0

n = dir – esq + 1 => dir - esq = n -1

meio = (esq + dir) / 2

dir’ = meio -1

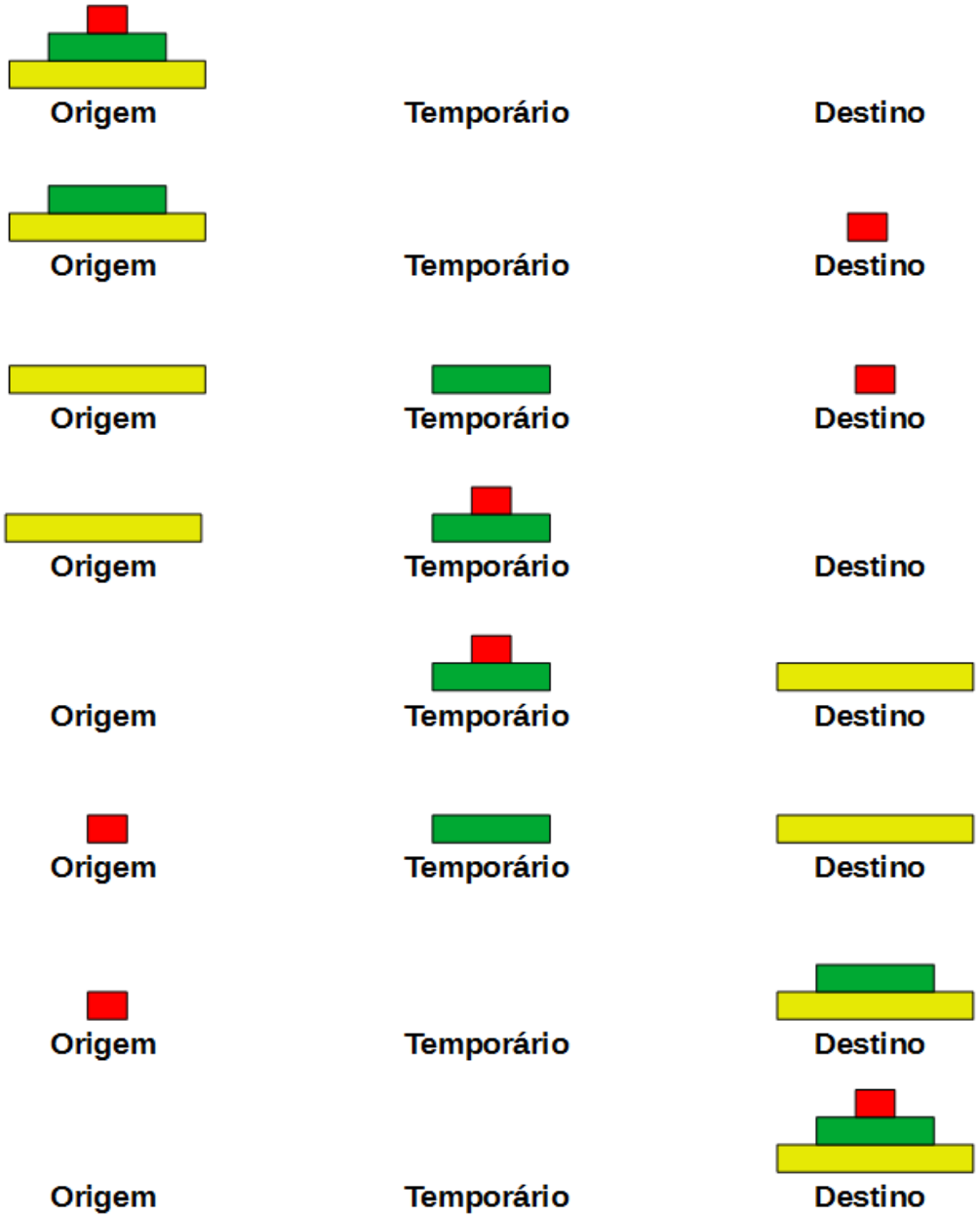
n’ = (esq + dir)/2 -1 – esq + 1

n’ = (esq + dir -2 –2*esq + 2)/2

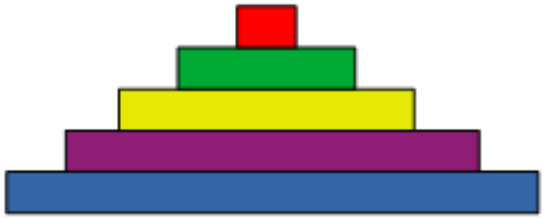
n’ = (dir – esq + 0)/2

n’ = (dir – esq)/2 = (n-1)/2

Torres de Hanói

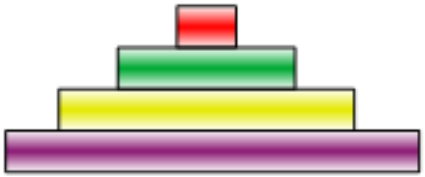


Torres de Hanói



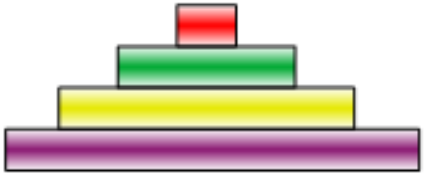
Origem Temporário Destino

Por hipótese de indução sabemos resolver o problema para n-1



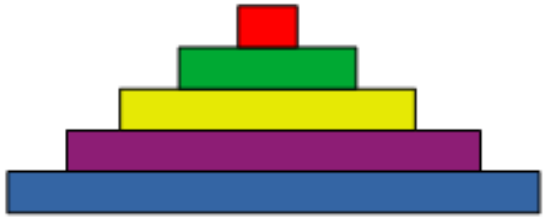
Origem Temporário Destino

Sabemos mover um único disco



Origem Temporário Destino

Por hipótese de indução sabemos resolver o problema para n-1



Origem Temporário Destino

Torres de Hanói

```
void Hanói(char ori, char dst, char aux, int n) {
    if(n == 1) {
        System.out.print("Move de "+ori+para "+dst);
    }
    else {
        Hanói(ori, aux, dst, n-1);
        Hanói(ori, dst, aux, 1);
        Hanói(aux, dst, ori, n-1);
    }
}
```

$T(n) = 1 \quad \text{para } n=1$

$T(n) = 2*T(n-1) + T(1) \quad \text{para } n>1$

$\mathbf{T(n) \in \Theta(2^n)}$

```
void Hanói(char ori, char dst, char aux, int n) {
    if(n == 1) {
        System.out.print("Move de "+ori+para "+dst);
    }
    else {
        Hanói(ori, aux, dst, n-1);
        System.out.print("Move de "+ori+para "+dst);
        Hanói(aux, dst, ori, n-1);
    }
}
```

$T(n) = 1 \quad \text{para } n=1$

$T(n) = 2*T(n-1) + 1 \quad \text{para } n>1$

$\mathbf{T(n) \in \Theta(2^n)}$

```
############ Hanói 1 ############
Move de O para D

############ Hanói 2 ############
Move de O para T
Move de O para D
Move de T para D

############ Hanói 3 ############
Move de O para D
Move de O para T
Move de D para T
Move de O para D
Move de T para O
Move de T para D
Move de O para D
```

```
############ Hanói 4 ############
Move de O para T
Move de O para D
Move de T para D
Move de O para T
Move de D para O
Move de D para T
Move de O para T
Move de O para D
Move de T para D
Move de T para O
Move de D para O
Move de T para D
Move de O para T
Move de O para D
Move de T para D
```