

Ordenação e Recursão



**Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
CEFET-RJ**

Ordenação por Intercalação

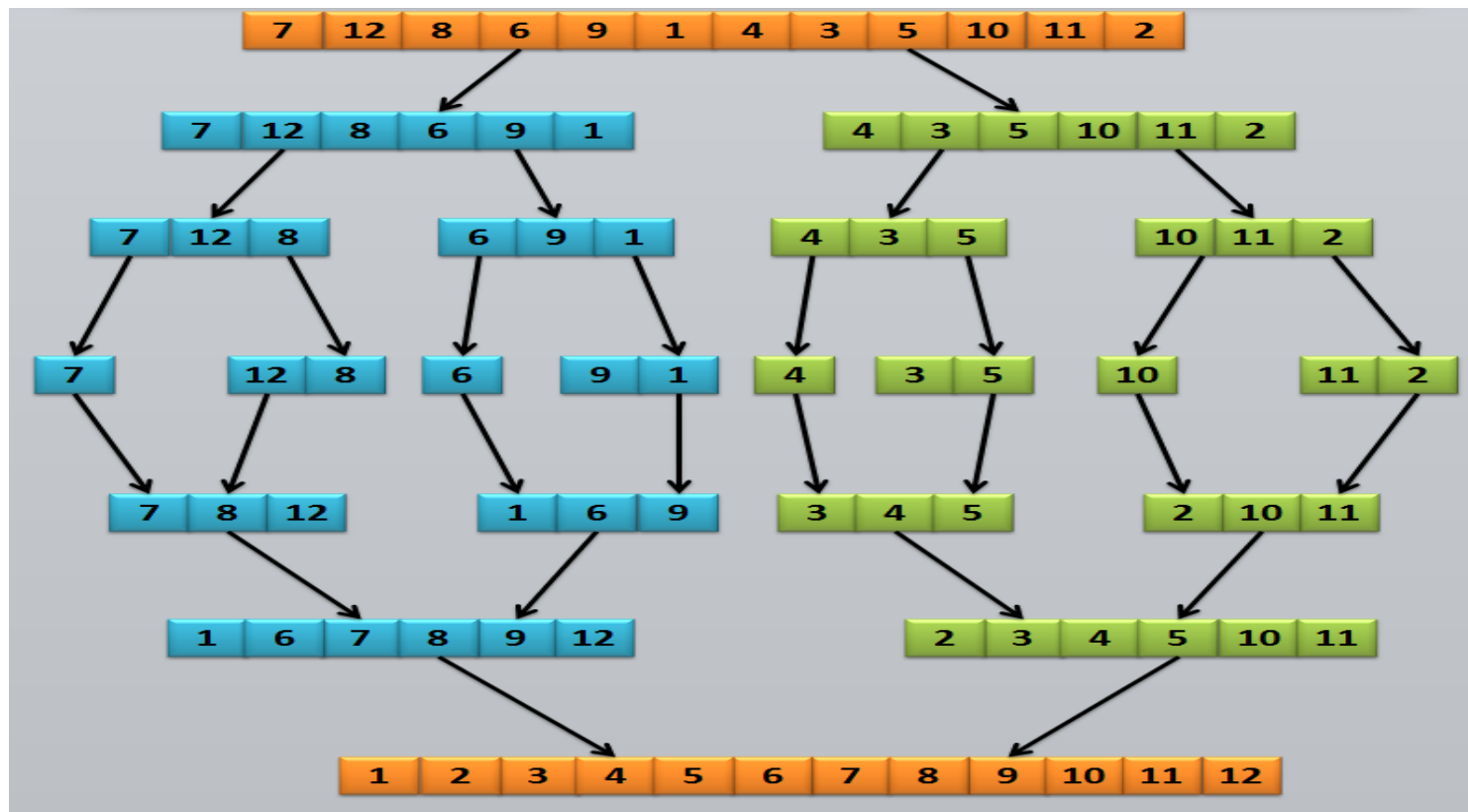
- A lista é dividida ao meio, recursivamente, até sobrar apenas um elemento.
- São ordenados e intercalados.
- Utiliza a técnica denominada Divisão e Conquista.



1. Dividir o problema original em subproblemas.
2. Conquista: resolver os subproblemas.
3. Combinar: intercalar as soluções obtidas e chegar a solução do problema original (global).

Merge Sort

- Seja uma lista A de n elementos. O algoritmo tem as seguintes fases:
 - Dividir: separa a lista A em duas sublistas aproximadamente pelo meio.
 - Conquistar: ordena cada sublista chamando Merge Sort **recursivamente**.
 - Combinar: intercala as sublistas ordenadas formando uma única lista.



Merge Sort - Algoritmo

mergesort (A, inicio, fim)

se (inicio < fim)

meio = (inicio + fim) div 2

mergesort(A, inicio, meio)

mergesort(A, meio + 1, fim)

função
recursiva

intercalar(A, inicio, fim, meio)

fim-se

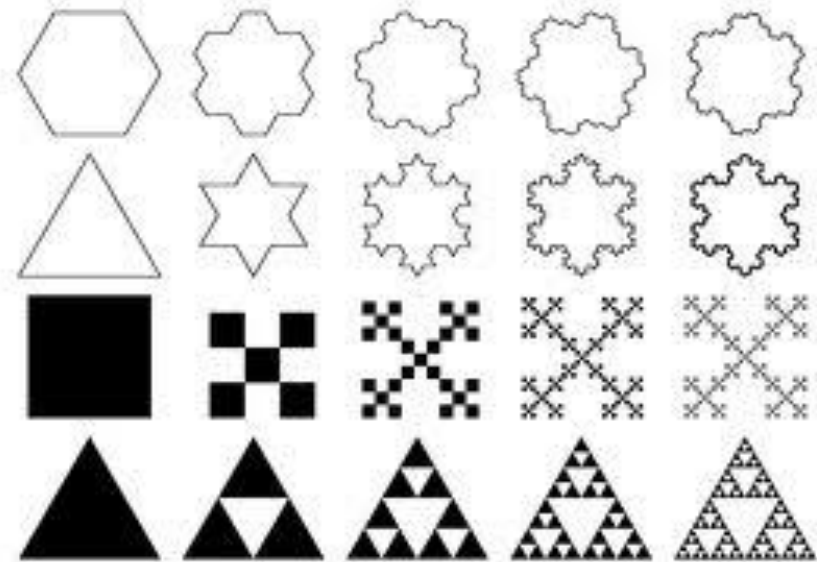
fim

Função Recursiva - função que chama a si mesma direta ou indiretamente (por meio de outra função).

Função Recursiva

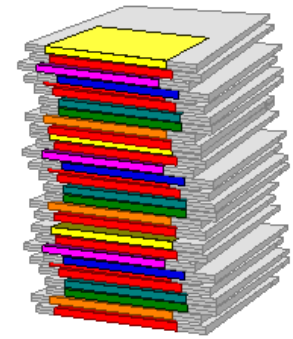
- Funções que chamam a si mesma (diretamente ou indiretamente).
 - Algumas vezes é útil ter funções que chamam a si mesmas.
- Resolver problemas de natureza recursiva ou para implementar estruturas de dados recursivas.

- ✓ Fractal
- ✓ Fibonacci
- ✓ Fatorial
- ✓ Ordenações
- ✓ Busca binária
- ✓ Árvores



Função Recursiva

- Quando uma função recursiva é executada, são alocados novos parâmetros e variáveis locais na pilha, para cada instância da função em questão.
 - É essencial ter referência aos parâmetros anteriores para poder retornar à função chamadora.



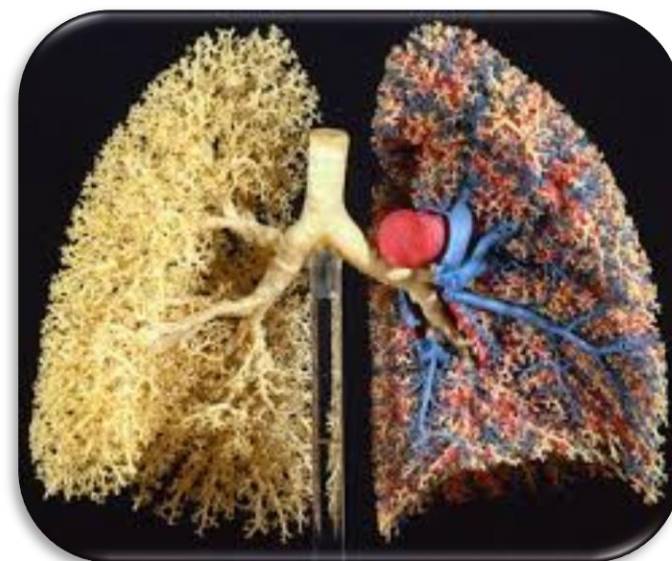
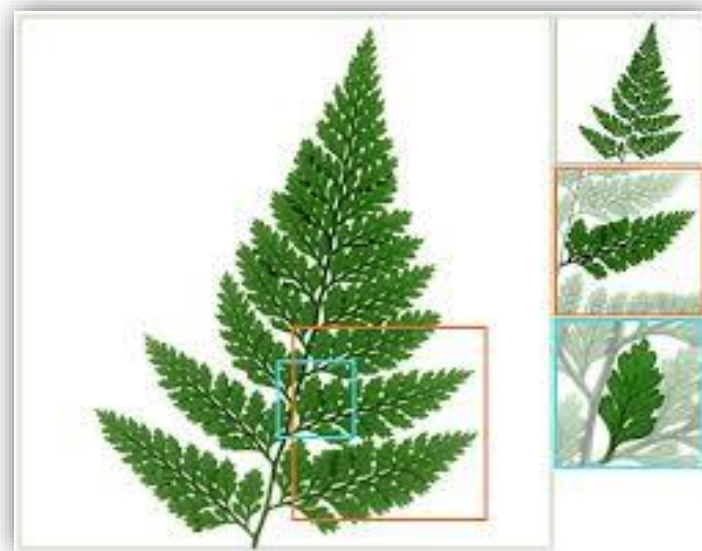
Função Recursiva – Partes Fundamentais

- **Ponto de Parada (condição de parada)**
 - ponto onde a função será encerrada
- **Regra Geral**
 - Método que reduz a resolução do problema através da função recursiva de casos menores (divisão e conquista).
 - Ao atingir o ponto de parada, os problemas são resolvidos e retornam o resultado obtido à função chamadora, sucessivamente, até atingir a função original.



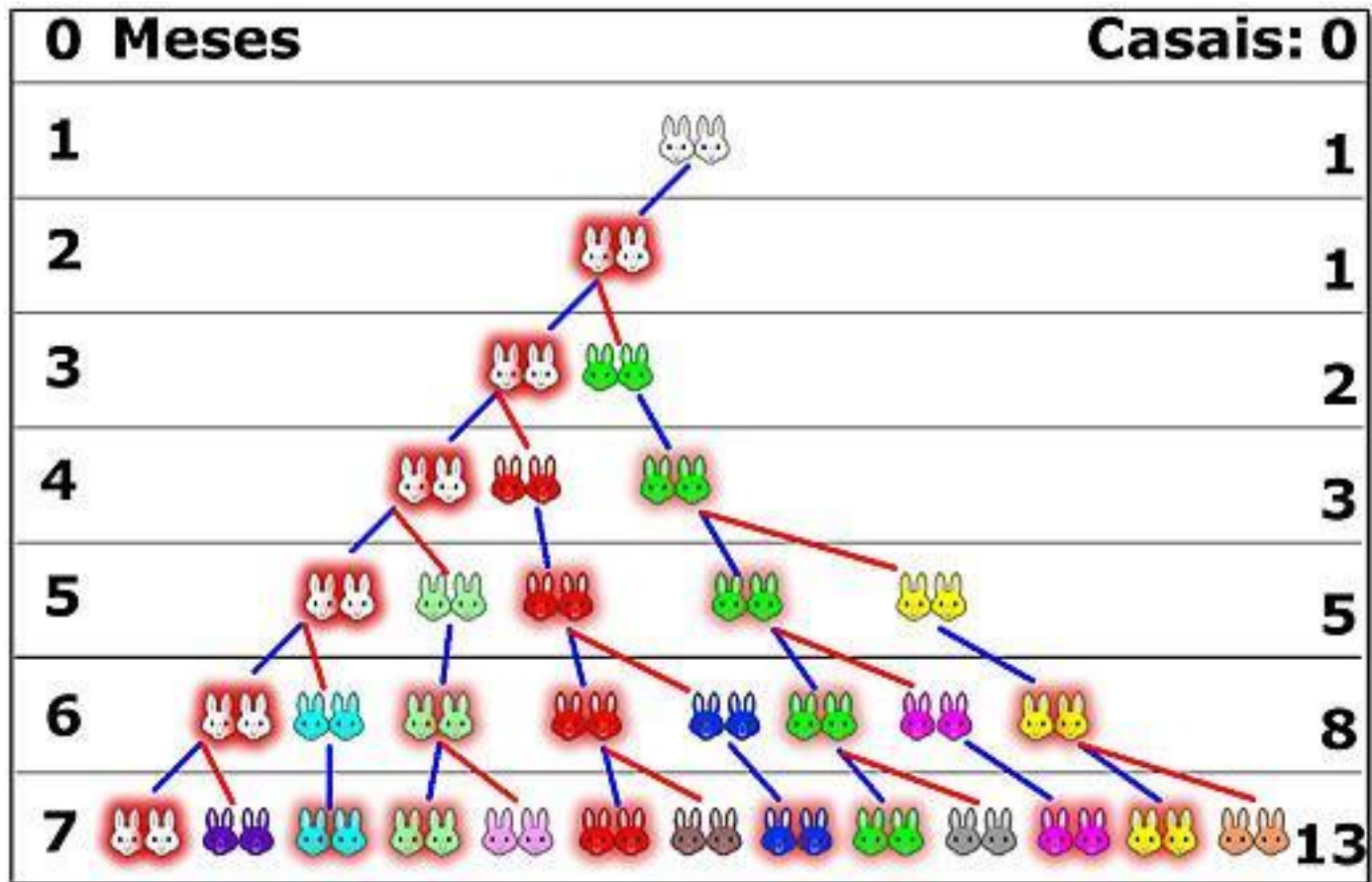
Sem ponto de parada - Loop infinito!

Recursividade na Natureza - Fractal



Sequência Fibonacci

- Tudo começou com um problema aparentemente banal:
 - Quantos pares de coelhos podem ser gerados de um par de coelhos em um ano?



Sequência Fibonacci



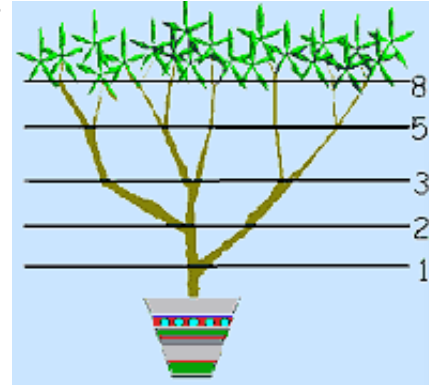
- O matemático italiano Leonardo Pisano (de Pisa), cujo apelido era "Fibonacci", ao resolver esse problema, transcreveu o que seria uma das sequências mais instigantes da matemática, que entrou para a história como a **sequência Fibonacci** (ou proporção áurea).
 - É uma série de números infinitos onde cada número é a soma dos dois anteriores.
 - A sequência inicia com 0 e 1.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Sequência Fibonacci

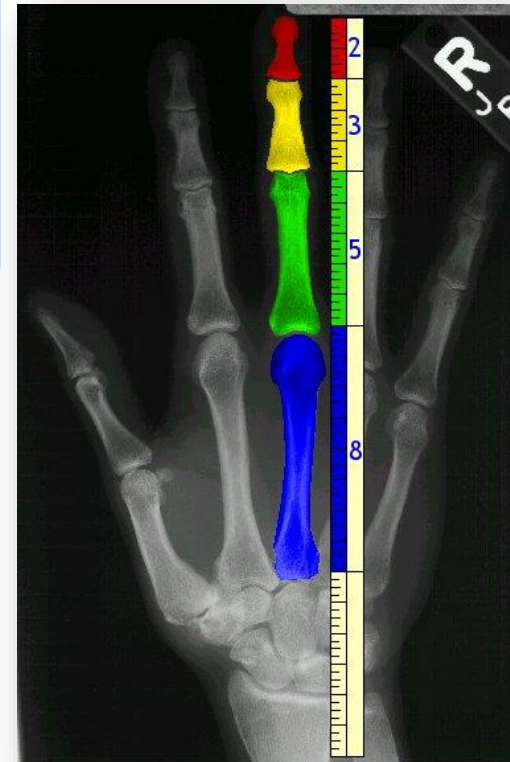
- Presente em diversos fenômenos da natureza:

- no comportamento da refração da luz
- átomos
- crescimento das plantas
- espirais das galáxias
- marfins de elefantes
- ondas no oceano, furacões etc



- Estudada e aplicada em diversas áreas:

- Biologia
- Arquitetura
- Computação
- Matemática
- Engenharia
- Artes...



Sequência Fibonacci – Solução Recursiva

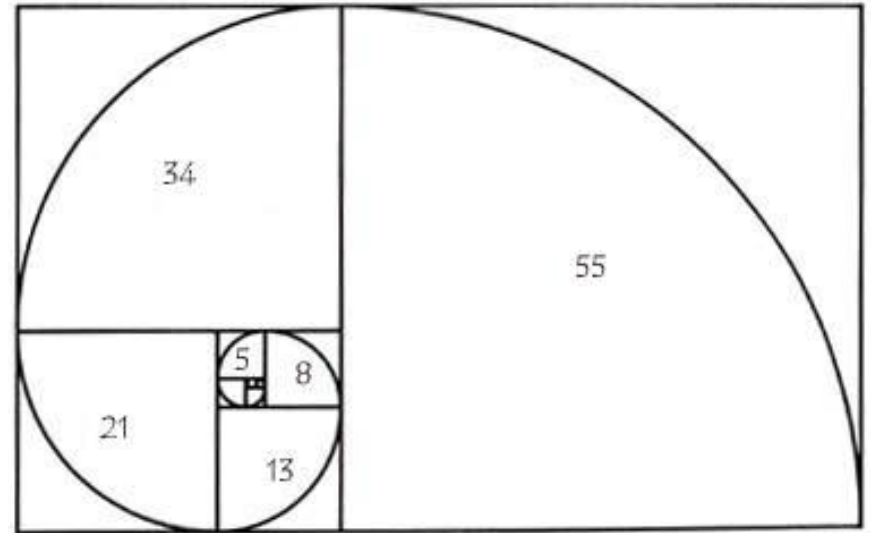
```
main()
{
    int n;

    cout << "Digite a quantidade de termos da sequencia: ";
    cin >> n;
    cout << "Sequencia de Fibonacci";

    for (int i = 0; i < n; i++)
        cout << fibonacci(i) << "\\t";
}
```

Sequência Fibonacci – Solução Recursiva

```
int fibonacci(int num)
{
    if (num < 2)
        return num;
    else
        return (fibonacci(num-1) + fibonacci(num-2));
}
```



Sequência Fibonacci – Partes Fundamentais

```
int fibonacci(int num)
```

```
{
```

```
    if (num < 2)
```

```
        return num;
```

```
    else
```

```
        return (fibonacci(num-1) + fibonacci(num-2));
```

```
}
```

Ponto de Parada

Regra Geral

Função Recursiva

- **Alguns algoritmos clássicos utilizam a recursão em sua solução.**
 - **Algoritmos de Ordenação (usam recursividade e o método de divisão e conquista)**
 - Merge Sort
 - Quick Sort
 - **Torre de Hanói**

MergeSort - main()

```
main()
{
    int A[5];
    for (int i = 0; i < 5; i++)
    {
        cout << "Entre com o " << i + 1 << "o. numero: ";
        cin >> A[i];
    }

    mergesort(A, 0, 4);

    for (int i = 0; i < 5; i++)
        cout << A[i] << "\t";
}
```


Implementação - mergesort()

```
void mergesort(int A[], int inicio, int fim)  
{  
    int meio;  
    if (inicio < fim)  
    {  
        meio = (inicio + fim)/2;  
        mergesort(A, inicio, meio);  
        mergesort(A, meio + 1, fim);  
        intercala(A, inicio, fim, meio);  
    }  
}
```

Implementação - intercala()

```
void intercala(int A[], int inicio, int fim, int meio)  
{  
    int i, aux [5], posLivre = inicio, inicioA1 = inicio, inicioA2 = meio + 1;  
  
    while (inicioA1 <= meio && inicioA2 <= fim)  
    {  
        if (A[inicioA1] > A[inicioA2]) {  
            aux[posLivre] = A[inicioA2];  
            inicioA2++;  
        } else {  
            aux[posLivre] = A[inicioA1];  
            inicioA1++;  
        }  
        posLivre++;  
    } // término do while
```

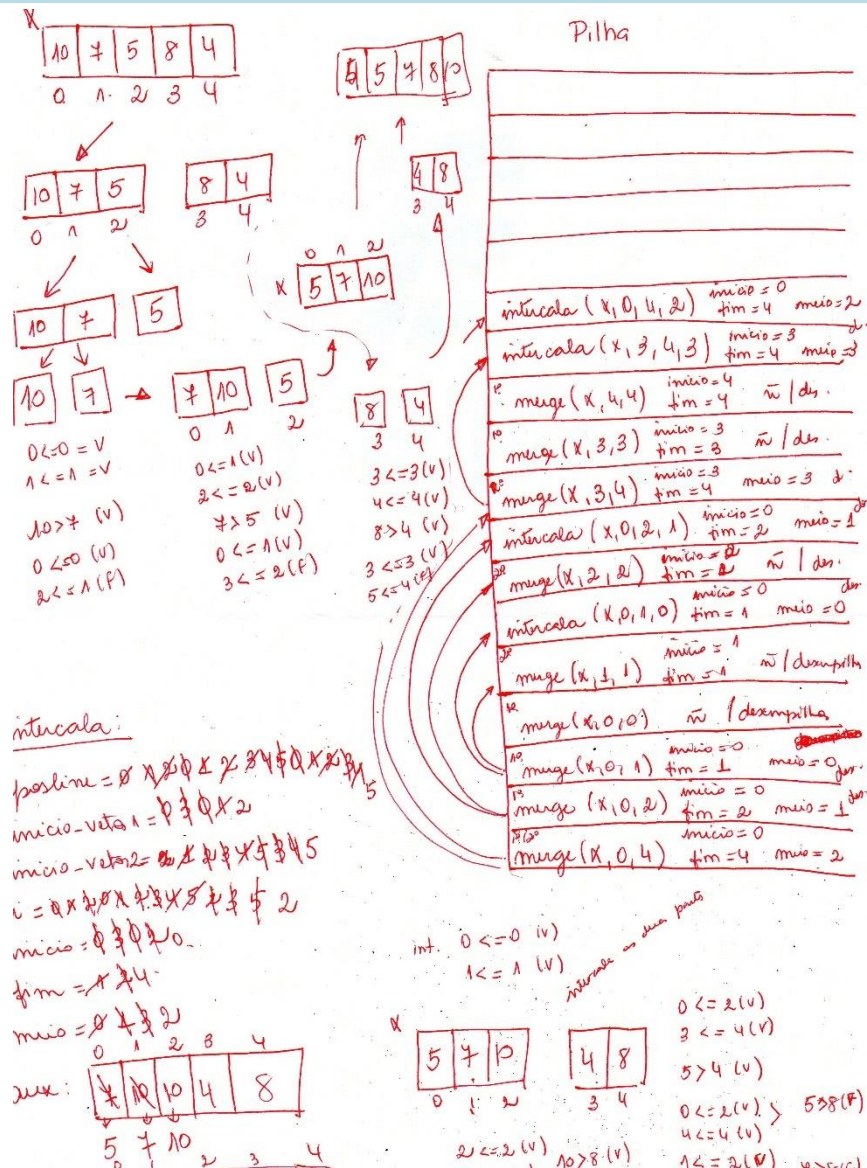
Implementação - cont. intercala()

```
for (i = inicioA1; i <= meio; i++) // elems. do 1o. array que não foram intercalados
{
    aux[posLivre] = A[i];
    posLivre++;
}
for (i = inicioA2; i <= fim; i++) // elems. do 2o. array que não foram intercalados
{
    aux[posLivre] = A[i];
    posLivre++;
}
for (i = inicio; i <= fim; i++) // atualiza array A conforme array auxiliar
    A[i] = aux[i];

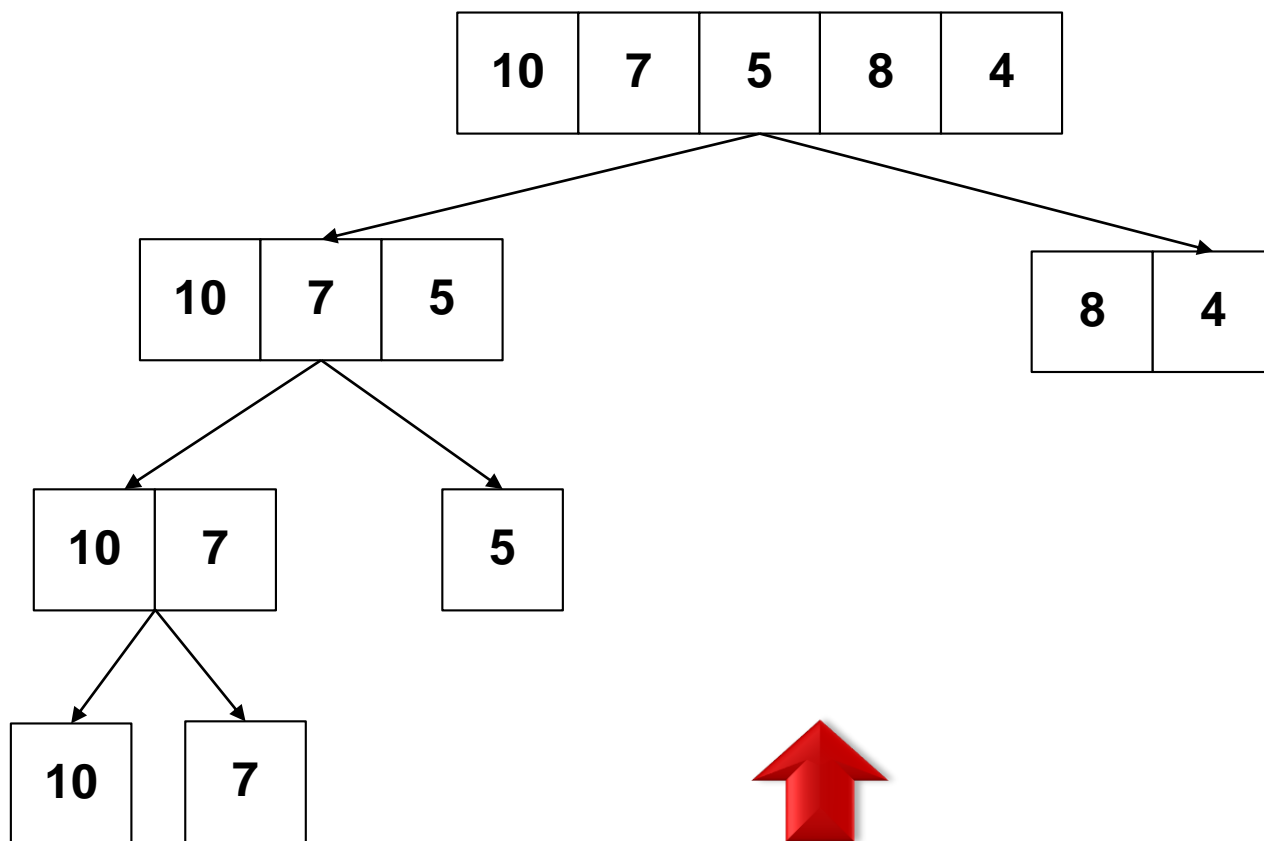
} // término da função intercala()
```

Teste de Qualidade do MergeSort (chinês)

Feito em aula



Merge Sort



Intercalar e Combinar



Torre de Hanói



- **Objetivo:**
 - Transferir todos os n discos de A (pino de origem) para B (pino de destino).
- **Regras:**
 - Mover um disco por vez.
 - Nunca colocar um disco maior em cima de um menor.

Função - Torre de Hanói

hanoi (n, A, B, C)

início

se (n > 0) então

hanoi(n-1, A, C, B)

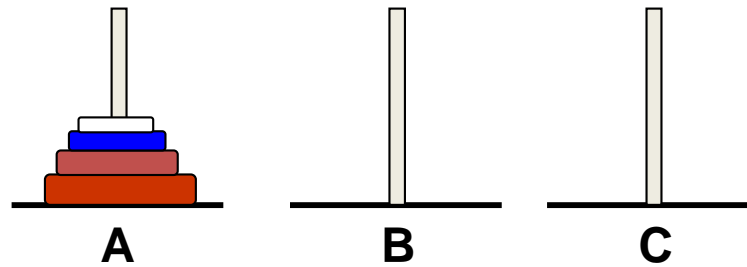
mover disco do topo de A para B

hanoi(n-1, C, B, A)

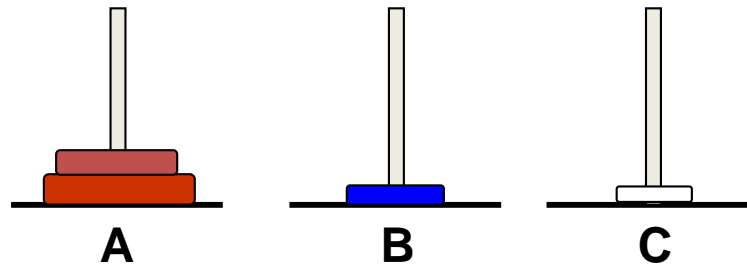
fim-se

fim

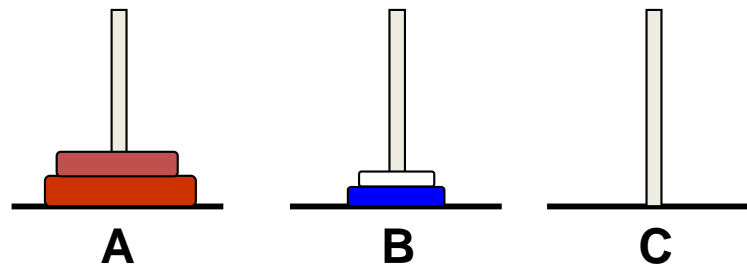
Torre de Hanoi



início

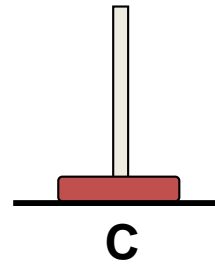
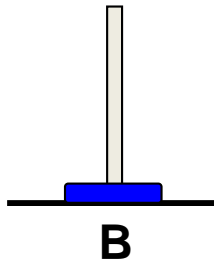
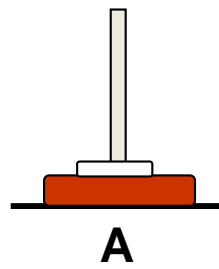


1º A – C
2º A – B

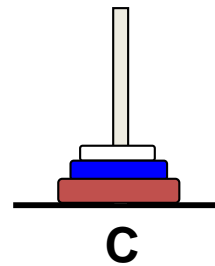
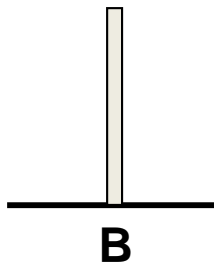
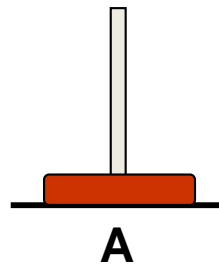


3º C – B

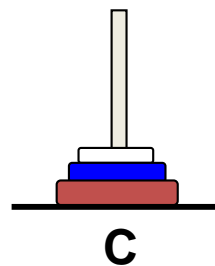
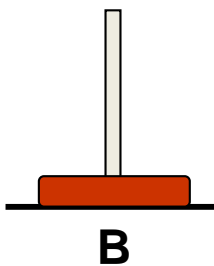
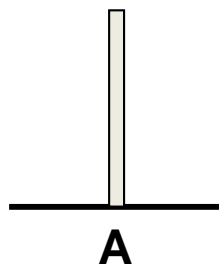
Torre de Hanoi



4° A – C
5° B – A

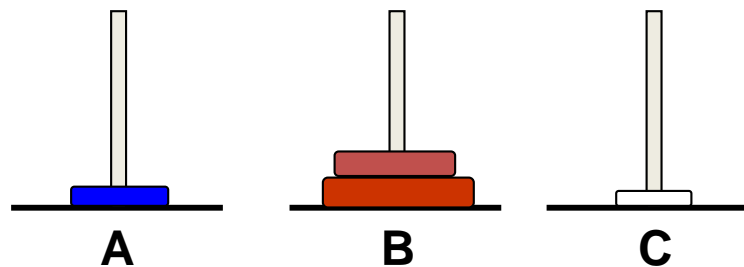
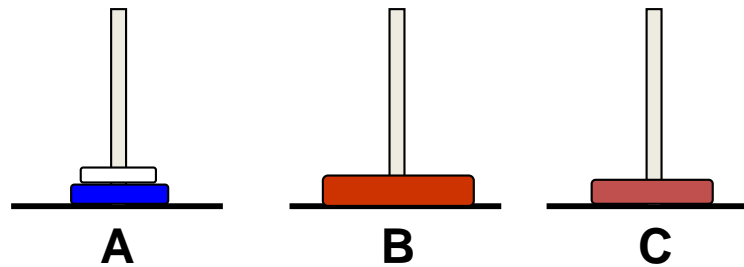
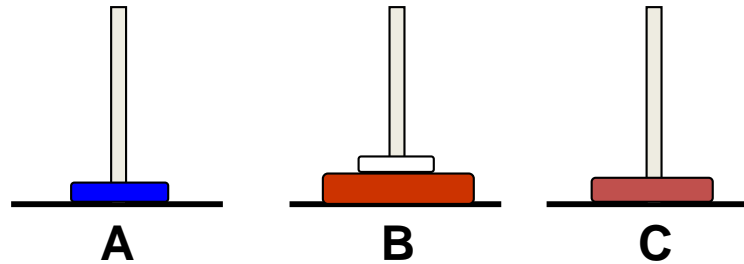


6° B – C
7° A – C

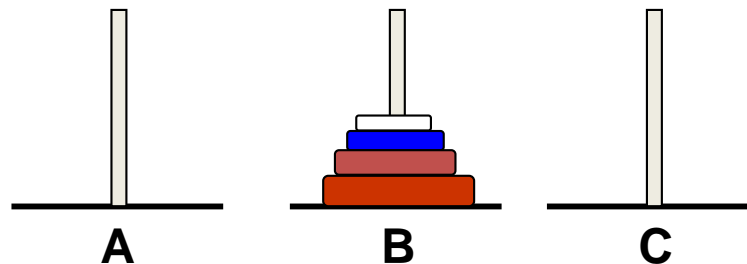


8° A – B

Torre de Hanoi



Torre de Hanoi



14° A – B
15° C – B

Quantidade mínima de movimentações:

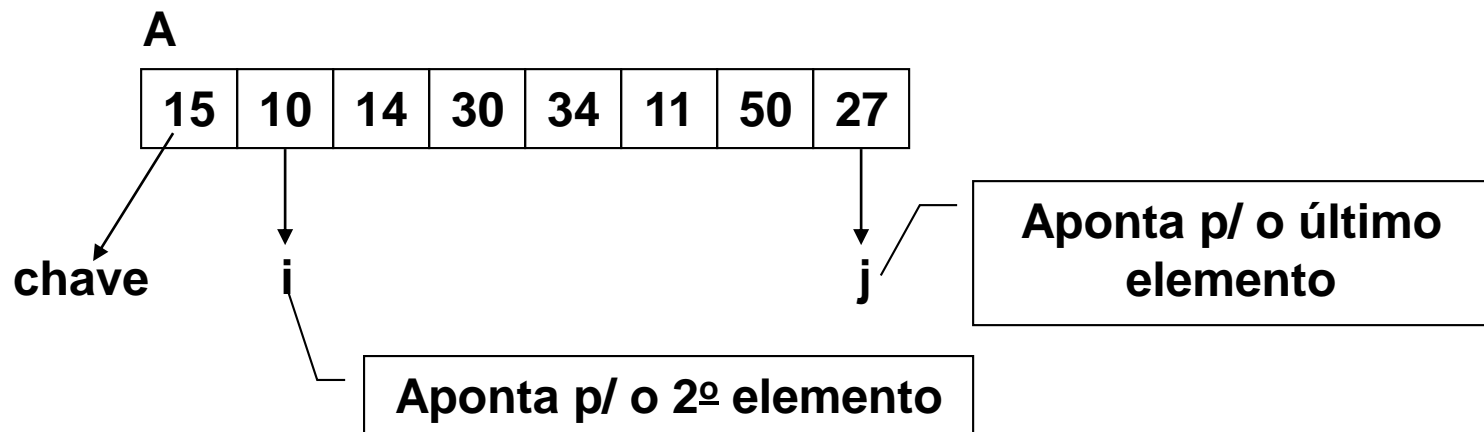
$$2^n - 1$$

Onde:

n = número de discos

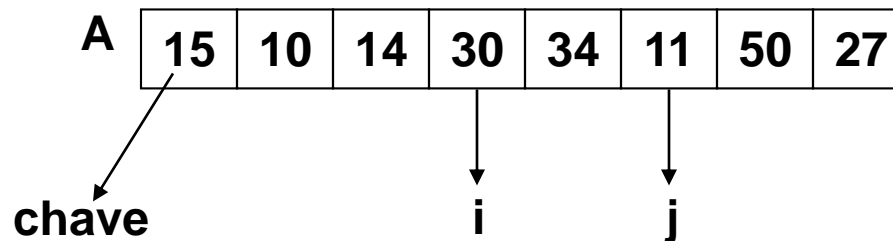
Quicksort

- Classifica uma lista, dividindo-a em partes menores, classificando essas partes e depois concatenando-as.
- Ideia principal \Rightarrow escolher uma chave e particionar a lista em relação a chave, de forma que todos os elementos de uma parte sejam menores que os outros elementos da outra parte.
- O algoritmo a seguir classifica uma lista A em ordem crescente.

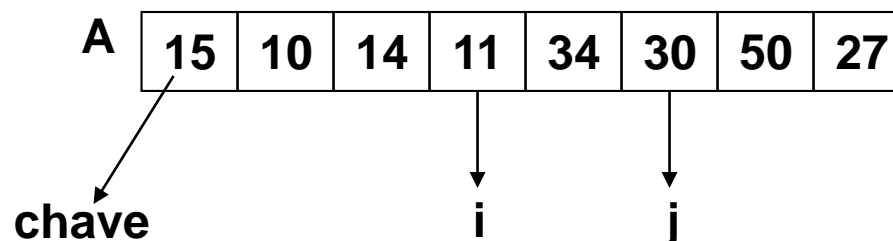


Exemplo – Quicksort

- **Determinar a posição correta dessa chave na lista A:**
 - Incrementa i sempre que $A[i] < \text{chave}$
 - Decrementa j sempre que $A[j] > \text{chave}$

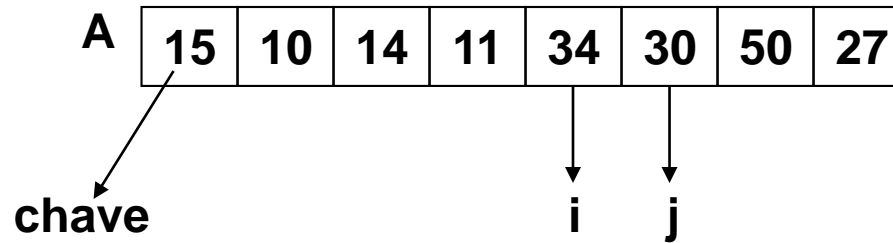


- Se $i < j \Rightarrow$ trocar $A[i]$ com $A[j]$



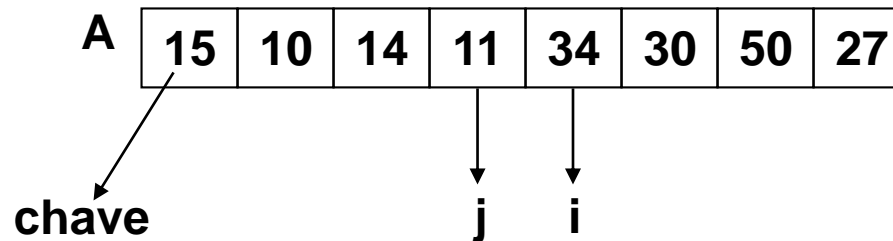
Exemplo – Quicksort

- enquanto $i < j \Rightarrow i++$



-Incrementa i sempre que $A[i] < \text{chave}$

-Decrementa j sempre que $A[j] > \text{chave}$



- Se $i < j \Rightarrow$ trocar $A[i]$ com $A[j]$ senão $\text{chave} \leftarrow j$

QuickSort - main()

```
main()
{
    int i, A[5];

    for(int i = 0; i < 5; i++)
    {
        cout << " Entre com o " << i + 1 << " o. numero: ";
        cin >> A[i];
    }

    quickSort(A, 0, 4);

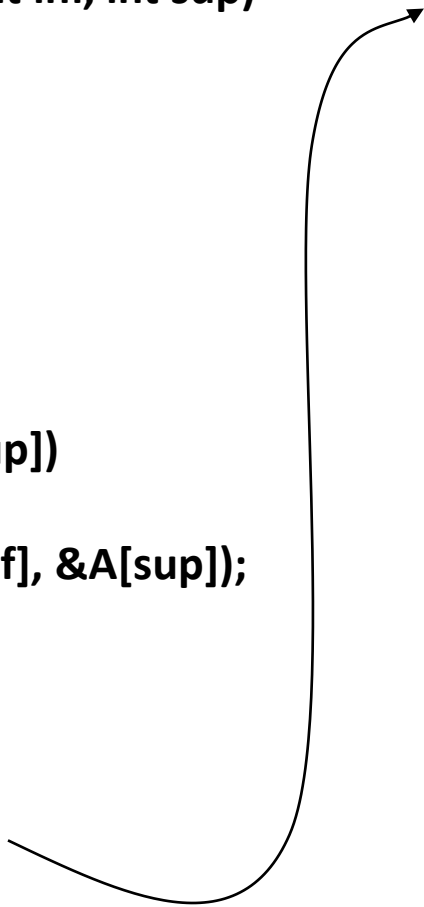
    for(int i = 0; i < 5; i++)
        cout << A[i] << "\\t";
}
```

Implementação QuickSort

```
void quickSort (int A[], int inf, int sup)
{
    int i, j, chave;

    if (sup - inf < 2)
    {
        if ((sup - inf) == 1)
        {
            if (A[inf] > A[sup])
            {
                troca (&A[inf], &A[sup]);
            }
        }
    }
    else
    {
        i = inf;
        j = sup;
        chave = A[inf];
```

```
        while (j > i)
        {
            i++;
            while (A[i] < chave)
            {
                i++;
            }
            while (A[j] > chave)
            {
                j--;
            }
            if (j > i)
            {
                troca (&A[i], &A[j]);
            }
        } //while
        troca (&A[inf], &A[j]);
        quickSort(A, inf, j - 1);
        quickSort(A, j + 1, sup);
    } //else
} //quickSort
```



Implementação Troca

```
void troca(int *x, int *y)
{
    int aux;

    aux = *x;
    *x = *y;
    *y = aux;
}
```

Resolução

```
inf = 0
sup = 7
i = 0 1 2 3 4
j = 7 6 5 4 3
chave = 15
```

```
1º Quicksort (0,2)
inf = 0
sup = 2
i = 0, 1, 2
j = 2, 1
chave = 11
```

0	1	2	3	4	5	6	7
15	10	14	11	34	30	50	27

↑
i

↑
j

0	1	2	3	4	5	6	7
11	10	14	15	34	30	50	27

0	1	2	3	4	5	6	7
10	11	14	15	34	30	50	27

2º Quicksort (0,0) → não faz

2º Quicksort (2,2) → não faz

Acaba recursão 2.

Resolução

1º Quicksort (4,7)

inf = 4

sup = 7

i = ~~4~~, ~~5~~, ~~6~~, 7

j = ~~7~~, 6

chave = 34

0	1	2	3	4	5	6	7
10	11	14	15	34	30	50	27

3º Quicksort (4,5)

inf = 4

sup = 5

0	1	2	3	4	5	6	7
10	11	14	15	34	30	27	50

3º Quicksort (7,5)

inf = 7

sup = 5

0	1	2	3	4	5	6	7
10	11	14	15	27	30	34	50

sai

Fractal

- Os fractais são objetos gerados por um processo recursivo, apresentando **autossimilaridade** e **complexidade infinita**.
 - **Autossimilaridade (semelhança)**: um fractal apresenta cópias aproximadas de si mesmo em seu interior. Um pequeno pedaço é similar ao todo. Visto em diferentes escalas a imagem de um fractal parece similar.
 - **Complexidade Infinita**: propriedade dos fractais que significa que é praticamente impossível representá-los completamente, pois a quantidade de detalhes é infinita. Sempre existirão reentrâncias e saliências cada vez menores.

Fractal

Autossimilaridade



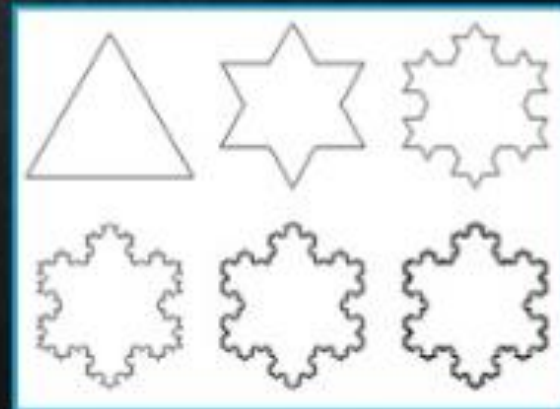
Um corte numa couve-flor exemplifica cópias sucessivas com autossemelhança.

Fractal

A Curva de Koch



A imagem ao lado ("*A Curva de Koch*") é um exemplo geométrico da construção de um fractal. Um mesmo procedimento é aplicado diversas vezes sobre um objeto simples, gerando uma imagem complexa. Cada pedaço da linha foi dividido em 4 pedaços menores idênticos ao pedaço original, cada um sendo 3 vezes menor que o tamanho original.



Exercícios

- 1. Implementar os algoritmos MergeSort, Quicksort e Hanoi.**
- 2. Fazer o chinês dos algoritmos MergeSort e QuickSort, tendo como entrada os valores: 25, 17, 32, 4, 8, 12.**
- 3. Quais são as diferenças entre o MergeSort e o QuickSort? Cite e explique.**
- 4. Apresentar as complexidades dos algoritmos Hanoi, MergeSort, e QuickSort (pior caso e caso médio).**

Referências

- **Moraes. *Estruturas de Dados e Algoritmos – uma abordagem didática*. Ed. Futura**
- **Markenzon e Szwarcfiter. *Estruturas de Dados e seus Algoritmos*. Ed. LTC**
- **Deitel. *Como Programar em C/C++*. Ed. Pearson**