

# Unidade 2

## Recursividade e Funções Recursivas



# Introdução

Continuando nosso estudo, nessa Unidade de Ensino da disciplina de Análise e Projeto de Algoritmos, vamos trabalhar com novos conceitos e tipos de funções diferentes no desenvolvimento de nossos algoritmos.

O Tema principal do conteúdo será a Recursividade. Será apresentado o conceito da Recursividade, as principais características que o compõem e, acima de tudo, como ele é utilizado dentro de um Algoritmo.

Caminhando com nosso estudo, também será construído exemplos em vários contextos que envolvam a Recursividade e as Funções Recursivas. Para um desses contextos vamos utilizar a Ordenação de Vetores, porém em uma técnica um pouco mais complexa das desenvolvidas na disciplina de Algoritmos e Estrutura de Dados. Para trabalhar com esse tipo de função será abordada a técnica *Merge Sort* de ordenação de elementos de um vetor.

Todo esse conteúdo será abordado considerando uma didática evolutiva, aliada a exemplos de aplicação dessas funções, estruturas e técnicas algorítmicas, sempre buscando elucidar de diferentes formas as dúvidas que surgirão no caminho.

No decorrer da unidade de ensino serão abordados eventualmente outros conteúdos que também serão de fundamental importância para o processo de desenvolvimento de um *software*.

## Objetivos

- Conhecer as ferramentas para análise e projeto de algoritmos;
- Conhecer e identificar uma função recursiva;
- Conhecer técnicas para trabalhar com ordenação de Vetores;
- Compreender e aplicar a recursividade em algoritmos.

## Conteúdo programático

Aula 01 – Utilizando a Recursividade na Ordenação de Vetores

Aula 02 – Funções Recursivas



Você poderá também **assistir às videoaulas** em seu celular! Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

## Aula 01 – Utilizando a Recursividade na Ordenação de Vetores

### Introdução

O objetivo dessa aula é abordar a técnica de Ordenação de Vetores *Merge Sort*. Diante disso, será explicada a técnica de ordenação, ou seja, todo o processo de separação, manipulação e ordenação dos elementos de um *Array*. A estrutura utilizada para desenvolvimento e demonstração dessa técnica será o *Array* Unidimensional conhecido também como vetor. Para melhor visibilidade, os conteúdos dos elementos serão valores numéricos inteiros aleatórios como pode ser visualizado na figura 1 a seguir. Os valores numéricos inteiros são todos os números positivos e ou negativos incluindo-se o número 0(zero).

**Figura 1** – *Array* Unidimensional de 10 posições

0	1	2	3	4	5	6	7	8	9
10	12	45	21	65	32	78	89	13	22

Fonte: o autor (2021)

O tamanho total do *Array* são 10(dez) elementos, isso quer dizer que podemos armazenar no máximo 10 elementos diferentes dentro do *Array* desde que sejam do mesmo tipo de dado. Os índices vão de 0 a 9. Sempre que o vetor tem uma quantidade de elementos, determina-se que o número do índice vai exatamente até um número

antecessor a quantidade máxima desse *Array*. Portanto, em um *Array* de 20 elementos, os índices vão de 0 a 19. Já em um *Array* de 15 elementos, os índices vão de 0 a 14 e assim sucessivamente.

Se você ainda não compreendeu muito bem a estrutura de um *Array* (vetor), os tipos de valores que eles podem armazenar ou as operações realizadas por eles, confira a indicação de leitura a seguir.

### Indicação de Leitura

Veja este livro sobre Algoritmos e Lógica de Programação, disponível no *link* abaixo, para maiores detalhes sobre algumas estruturas de dados, inclusive vetores e matrizes. Alguns outros conteúdos como: registros, novos tipos de variáveis e programação orientada a objetos constam nesta obra de Xavier.

Disponível em: [encurtador.com.br/ryAEZ](http://encurtador.com.br/ryAEZ), acesso em: 13 jul.2021.

XAVIER, Gley Fabiano Cardoso. **Lógica de programação**. São Paulo: Senac, 2018. 322 p.

## Merge Sort

Assim como as outras técnicas de ordenação de vetores, esta técnica tem o objetivo de colocar os elementos que estão no *Array* em ordem. Como pode ser observado na figura 1, utilizaremos valores numéricos como dados do nosso *Array*.

Segundo o autor Laureano (2012), em sua obra intitulada “*Estrutura de dados com algoritmos e C*”, a técnica *Merge Sort*, consiste na ordenação por comparação do tipo dividir-para-conquistar.

A ideia dessa técnica é dividir o *Array* em partes menores, ordenar essas partes, para depois juntar novamente todo o *Array* ordenado. Essa técnica utiliza a Recursividade para realizar a ordenação dos elementos, foco principal dessa aula: Utilizando a Recursividade na Ordenação de Vetores.

O processo para ordenação, a exemplo de outras técnicas como *Insertion Sort* e *Bubble Sort*, utiliza laços de repetições para varrer os elementos do *Array* e ordenar a estrutura até que finalmente esteja totalmente em ordem.

Para melhor entendimento da técnica de ordenação de vetores, veja a indicação de vídeo a seguir.

### Indicação de Vídeo

Veja este vídeo sobre a Ordenação de Vetores. No vídeo, por meio de um vetor humano, é realizada o teste de mesa de todo o processo de ordenação do *Array*. Assista quantas vezes achar necessário, faça anotações para que você compreenda o processo realizado e depois compare com a implementação via código que será desenvolvida.

Disponível em: [https://www.youtube.com/watch?v=XaqR3G\\_NVoo](https://www.youtube.com/watch?v=XaqR3G_NVoo). Acesso em: 17 jul. 2021.

O vídeo é bem interessante, pois apresenta passo a passo o que acontece na ordenação da estrutura. Lembrando que o teste de mesa sempre é importante para identificar a técnica de ordenação utilizada, tendo em vista que, o resultado final de todas as técnicas é sempre o mesmo. O que muda é a forma que a ordenação foi realizada.

### Técnica do *Merge*

Para abordarmos o que acontece na técnica de ordenação do *Merge*, vamos apresentar um *Array* com os seguintes elementos como é apresentado na figura 2.

**Figura 2** - Um *Array* de 8 posições

0	1	2	3	4	5	6	7
2	5	7	9	1	6	8	10

Fonte: o autor (2021)

Uma particularidade no algoritmo do *Merge* que temos que observar, é que ele já exige que o *Array* esteja ordenado, ou melhor que as partes do *Array* estejam ordenadas. Analisando o *Array* da Figura 2, você pode até pensar que ele não está ordenado completamente, porém como já apresentado, o algoritmo do *Merge* trabalha com a técnica do dividir para conquistar, portanto vamos dividir o *Array* e apresentá-lo novamente na figura 3.

**Figura 3** - Um *Array* de 8 posições

0	1	2	3	4	5	6	7
2	5	7	9	1	6	8	10

Fonte: o autor (2021)

Agora fica bem mais fácil visualizar a ordenação nas duas partes do *Array* separadamente. Na figura 3, temos a primeira metade do *Array* com os índices de 0 a 3 contendo os elementos {2,5,7,9}, e na segunda metade do *Array* com os índices de 4 a 7 contendo os elementos {1,6,8,10}. Note que as duas metades isoladas já estão ordenadas, facilitando assim o processo da ordenação total do *Array* (CORMEN, 2012).

Essa é a primeira parte do Algoritmo de ordenação do *Merge Sort*. A partir desse ponto, com “dois” *Arrays* isolados devidamente ordenados, os algoritmos fazem a ordenação final, que consiste no processo de colocar em um novo *Array*, todos os elementos dos dois *Arrays* iniciais já ordenados, ou seja, em ordem (CORMEN, 2012).

Continuando, será apresentado como se chega nessa ordenação. Em primeiro lugar a partir desse *Array*, o algoritmo cria mais “2” vetores auxiliares. Um vetor para trabalhar com os valores da esquerda, já ordenados: {2,5,7,9}, e outro para trabalhar com os valores da direita que também já estão ordenados: {1,6,8,10}. Os vetores podem ser nomeados como: **L** para esquerda e **R** para direita por exemplo.

Os vetores auxiliares são criados com  $n$  posições mais 1, assim sendo, vetor da direita ficaria a:  $R[N + 1]$  e vetor da esquerda:  $L[N + 1]$ .

O algoritmo vai precisar de 3 ponteiros. Os 2 primeiros ponteiros vão varrer os vetores da esquerda e direita e o último ponteiro para receber os elementos já em ordem.

O procedimento é simples: Se o primeiro elemento do vetor da esquerda for menor que o vetor da direita, um terceiro vetor irá receber esse elemento, caso contrário o mesmo vetor recebe o elemento da direita (CORMEN, 2012).

Na primeira videoaula, os detalhes da ordenação, bem como o processo do *Merge* no Algoritmo do *Merge Sort* é explicado. Acompanhe o vídeo para compreender o algoritmo e ficar por dentro da sua lógica.

## Videoaula 1

A primeira videoaula da aula 2, vai apresentar a lógica do algoritmo *Merge*. O professor vai explicar o código na prática, por meio do teste de mesa que faz o passo a passo de toda a ordenação do vetor. Assista e estude esse vídeo para compreender o processo de ordenação por meio da técnica *Merg Sort*.



### Videoaula 1

Utilize o QRcode para assistir!

A primeira videoaula da aula 2, vai apresentar a lógica do algoritmo *Merge*.



Prosseguindo com a ideia de ordenação, a figura 4 apresenta como ficará o *Array* após aplicada a técnica completa de ordenação do *Merge Sort*.

**Figura 4** - *Array* completamente ordenado

0	1	2	3	4	5	6	7
1	2	5	6	7	8	9	10

Fonte: o autor (2021)

Vamos trabalhar agora com um algoritmo por meio do Ambiente de Desenvolvimento Integrado NetBeans, demonstrando a construção e execução do *Merge Sort*.

Após a criação de um projeto no ambiente, vamos adicionar o *import* para trabalhar com *Arrays* e outro para criar um objeto da classe *Scanner*. Este processo é demonstrado na figura 5.

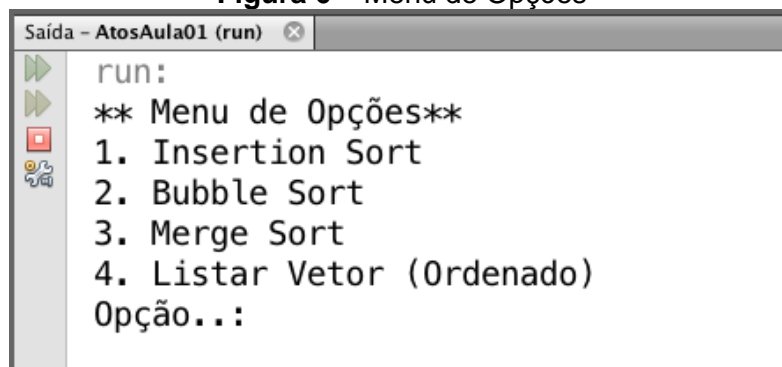
**Figura 5 – Imports**

```
8  import java.util.Arrays;  
9  import java.util.Scanner;
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Vamos implementar nosso código na própria classe Principal, onde se encontra o método *Main()*. Vamos definir um Menu de opções, no qual o usuário irá escolher entre: 1. *Insertion Sort*, 2. *Bubble Sort*, 3. *Merge Sort*, 4. Listar Vetor (Ordenado). Para qualquer outra opção, será apresentada a mensagem: “Opção Inválida”. A tela de execução é exibida na figura 6.

**Figura 6 – Menu de Opções**



A imagem mostra a janela de saída do NetBeans com o título "Saída - AtosAula01 (run)". O conteúdo da janela é o seguinte:

```
run:  
** Menu de Opções**  
1. Insertion Sort  
2. Bubble Sort  
3. Merge Sort  
4. Listar Vetor (Ordenado)  
Opção..:
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Para a ordenação no algoritmo do *Merge Sort*, será necessário 2 *Arrays*. Já para as outras técnicas trabalhadas no Menu 1 e 2, mais 2 *Arrays* serão necessários. A Figura 7 apresenta os quatro *Arrays* mencionados.



```
60     int[] vetDes = new int[10];
61     int[] vetOrd = new int[10];
62
63     int[] v = {5,2,7,9,6,8,10,1};
64     int[] w = new int[v.length];
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Agora vamos abordar um dos métodos responsáveis por realizar as verificações dos elementos do vetor e consequentemente sua ordenação. O primeiro método será o *mergeSort()*, que é chamado dentro da opção “3. Merge Sort”, do nosso Menu de Opções.

Dentro da implementação do método *main()*, quando o usuário escolher a opção 3, o algoritmo fará o seguinte: figura 8.

```

} else if (op == 3) {

    System.out.println(" ** Irá executar o método Merge Sort ** ");
    mergeSort(v, w, 0, v.length - 1);
    System.out.println(Arrays.toString(v));

} else if (op == 4) {

```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Como pode ser observado na figura 8, após o usuário optar pela opção 3, o algoritmo exibe uma mensagem informando o usuário que a ordenação será realizada por meio do método *Merge Sort*. Na sequência, o algoritmo chama o método *mergeSort()* e passa alguns parâmetros para que ele seja executado. O método será explicado neste material com maiores detalhes, assim como, a sua codificação, variáveis utilizadas etc. O método pode ser observado na figura 9.

**Figura 9 – Método Merge Sort**

```
122 private static void mergeSort(int[] v, int[] w, int ini, int fim) {  
123  
124     if(ini < fim){  
125  
126         int meio = (ini + fim) / 2;  
127         mergeSort(v, w, ini, meio);  
128         mergeSort(v, w, meio + 1, fim);  
129         intercalar(v, w, ini, meio, fim);  
130  
131     }  
132  
133 }
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Explicando a funcionalidade do método *Merge Sort*, inicialmente ele solicita 4 parâmetros obrigatórios. O primeiro parâmetro recebe o vetor contendo a ordenação em duas partes como foi exibido na Figura 7, o segundo parâmetro vai receber um vetor em branco que será utilizado no processo de ordenação. O terceiro e o quarto parâmetro recebem respectivamente os índices do início e fim do vetor. Nesse caso, o terceiro parâmetro vai receber sempre 0(zero), e o quarto parâmetro o tamanho do segundo vetor menos 1. Logo, para um vetor de 10 elementos, o primeiro parâmetro “ini”, vai receber 0 e o segundo parâmetro “fim”, recebe o valor 9.

Logo após, o método encontra o meio do *Array* de acordo com os elementos existentes, e chama novamente o método dentro dele 2 vezes. Esse processo será explicado na próxima videoaula.

## Videoaula 2

A próxima videoaula, vai abordar o método *mergeSort()* e como funciona a chamada recursiva existente dentro dele. O professor vai explicar o código na prática, construindo o método dentro do NetBeans e explicando sua lógica. Assista e estude esse vídeo para compreender o processo de ordenação por meio da técnica *Merg Sort*, a partir de um exemplo implementando na linguagem de programação Java.



## Videoaula 2

Utilize o QRcode para assistir!

A próxima videoaula, vai abordar o método `mergeSort()` e como funciona a chamada recursiva existente dentro dele.



Para a finalização da ordenação do algoritmo *Merge Sort*, só falta mais um procedimento. Esse procedimento é realizado pelo método sem retorno com o nome *intercalar()*. A chamada para esse método pode ser visualizada também na figura 9 já exibida anteriormente, porém vamos abordar esse algoritmo a partir de agora.

O método *intercalar()*, será responsável por ordenar os vetores da esquerda e da direita dentro do terceiro vetor. Os vetores auxiliares:  $R[N + 1]$  direita e  $L[N + 1]$ , esquerda serão comparados à medida que forem sendo chamados dentro do método *mergeSort()* e no método *intercalar()* inseridos no terceiro vetor. Após todo o procedimento realizado o *Array* é totalmente ordenado. O código completo do método *intercalar()* é exibido na figura 10.

**Figura 10 – Método Intercalar**

```
private static void intercalar(int[] v, int[] w, int ini, int meio, int fim) {  
  
    for(int k = ini; k <= fim; k++)  
        w[k] = v[k];  
  
    int i = ini;  
    int j = meio + 1;  
  
    for(int k = ini; k <= fim; k++){  
  
        if(i > meio){  
            v[k] = w[j++];  
        }  
        else  
        if(j > fim){  
            v[k] = w[i++];  
        }  
        else  
        if(w[i] < w[j])  
            v[k] = w[i++];  
        else  
            v[k] = w[j++];  
    }  
}
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

## Videoaula 3

A próxima videoaula, vai abordar a execução do algoritmo para a ordenação de vetores utilizando a técnica *mergeSort()* a partir da Recursividade. O professor vai explicar o código por meio do *debugging* no NetBeans. Assista e estude esse vídeo para complementar o seu aprendizado nos conteúdos dessa primeira Aula da Unidade 2 de Ensino.



### Videoaula 3

Utilize o QRcode para assistir!

A próxima videoaula, vai abordar a execução do algoritmo para a ordenação de vetores utilizando a técnica *mergeSort()* a partir da Recursividade.



A seguir, a figura 11 exibe os elementos do vetor já totalmente ordenados após a execução do programa.

**Figura 11** – Execução do Algoritmo

```
Saída - AtosAula01 (run) x
run:
** Menu de Opções**
1. Insertion Sort
2. Bubble Sort
3. Merge Sort
4. Listar Vetor (Ordenado)
Opção..: 3
  ** Irá executar o método Merge Sort **
[1, 2, 5, 6, 7, 8, 9, 10]
Deseja continuar? <S=SIM>..:
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Observe que na execução o usuário escolheu a opção 3 *Merge Sort*, a seguir o algoritmo exibe em um único vetor, os elementos dos dois vetores conjuntamente.

## Aula 02 – Funções Recursivas

### Introdução

Nessa aula, o tema será a Recursividade por meio de Funções Recursivas. Você já conseguiu compreender o que faz a recursividade na Aula 1 dessa unidade, pois utilizamos esse conceito no algoritmo de ordenação de vetores no *Merge Sort*. O que será abordado agora, serão além de outros exemplos de algoritmos utilizando a recursividade, será uma fundamentação desse processo.

A recursividade, como tantos outros conceitos existentes de algoritmos, está presente no cotidiano das pessoas, isso é, em situações rotineiras a que somos submetidos e que por desconhecimento do assunto nem nos damos conta.

Se verificarmos no dicionário, define-se “recursividade” como “Propriedade sintática pela qual um elemento pode aparecer um número infinito de vezes numa derivação, introduzido sempre pela mesma regra” (DICIO, 2021).

Para uma maior compreensão, vamos exemplificar por meio de alguns exemplos. O espelho de frente para o outro é um exemplo perfeito do efeito que a recursividade causa. Creio que todos já fizeram essa experiência. A imagem refletida de um espelho no outro gera um reflexo infinito. A Figura 1, exibe um exemplo desse reflexo recursivo.

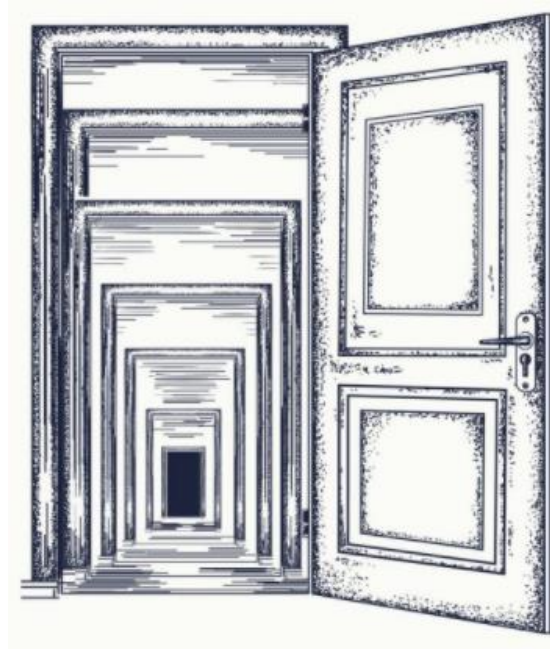
**Figura 1** – Processo de Desenvolvimento



Fonte: disponível em: <https://br.depositphotos.com/>. Acesso em: 20 jul. 2021.

A figura 1 apresenta a imagem de um relógio que quando você olha para o centro, nos mostra a ideia que existe outro relógio idêntico ao primeiro. E essa sensação vai se repetindo em forma de *looping*. A figura 2 exibe outra imagem que também apresenta o mesmo efeito de continuidade do mesmo objeto.

**Figura 2** – Processo de Desenvolvimento



Fonte: disponível em: <https://br.depositphotos.com/>. Acesso em: 20 jul. 2021.

Na programação, se esse efeito “infinito” não for parado em algum momento do algoritmo, o programa ficará preso e nunca vai sair desse *loop*. Os laços de repetições são comandos dentro da lógica de programação que podem ter o mesmo efeito senão forem construídos corretamente.

Dando continuidade ao nosso estudo, agora que você já compreendeu como o efeito recursivo acontece, vamos abordar como acontece a recursividade na programação. A linguagem de programação utilizada para demonstração desse exemplo será Java.

## Recursividade

A Recursividade dentro da programação, acontece quando dentro de uma função existe uma linha de código que chama novamente a mesma função. A figura 3 apresenta

um código simples de uma função que requer um valor “texto de entrada” e imprimir o valor dentro da função (CORMEN, 2012).

**Figura 3** – Estrutura de uma função recursiva

```
public static void imprimir(String ies){  
  
    System.out.println(ies);  
  
    imprimir("UniFil");  
}
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Na última linha da função ela chama novamente a si própria, passando o valor textual “UniFil” e isso causa um processo recursivo. Para ver o efeito que essa função vai causar, basta somente chamar a função **imprimir()** no método **main()**. A figura 4 exibe o método **main()** chamando a função recursiva **imprimir()**.

**Figura 4** – Função imprimir sendo chamada

```
6  [ ]  
7  
8  
9  
10  
11  
  
public static void main(String[] args) {  
  
    imprimir("UniFil");  
  
    System.out.println("Fim do Progrma!");  
}
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Observe que a no corpo da função **main()**, função principal da classe, a função **imprimir** é chamada passando por parâmetro o valor textual “UniFil”. Sendo assim, quando o programa é executado a função principal chama a função **imprimir** que por sua vez imprime um valor em tela e “chama novamente” a si própria. Os dados gerados em tela após a execução estão apresentados na figura 5.

**Figura 5** – Execução da função



Fonte: elaborado pelo autor via IDE NetBeans (2021)

Logo após a apresentação dos dados da figura 5, o programa apresenta uma mensagem de estouro de memória. Todo esse processo é explicado na próxima videoaula.

### Videoaula 1

Nessa videoaula, será abordado o exemplo simples de recursividade com o auxílio de um algoritmo na linguagem de programação Java. O professor vai implementar uma função recursiva para demonstrar qual é o comportamento esperado para esse tipo de codificação.

Acompanhe com atenção e não deixe de realizar no seu ambiente de desenvolvimento, todas as implementações, alterações no código e orientações de execução e o próprio teste do algoritmo.

O teste de mesa para essa função é imprescindível para que você compreenda em detalhes tudo o que está acontecendo no algoritmo.





### Videoaula 1

Utilize o QRcode para assistir!

Nessa videoaula, será abordado o exemplo simples de recursividade com o auxílio de um algoritmo na linguagem de programação Java.



## Fatorial de um Número

Outros exemplos de algoritmos conhecidos podem ser resolvidos utilizando as funções recursivas, um deles é o Fatorial de um número. O Fatorial de um número é calculado a partir da multiplicação de todos os seus antecessores até o número 1 (um), cuja fórmula para esse cálculo é:

$$n! = n \cdot (n - 1)$$

Desse modo, se você quiser saber quanto é o Fatorial do número 5, basta multiplicar  $(5 * 4 * 3 * 2 * 1) = 120$ . Existe outra maneira de fazer o cálculo que é inverter o início do algoritmo, ou seja, partir do número 1 e multiplicar pelos sucessores até o número que se deseja encontrar o Fatorial. Desse modo, teríamos a fórmula  $(1 * 2 * 3 * 4 * 5) = 120$ . O resultado é o mesmo (CORMEN, 2012).

A figura 6, apresenta o algoritmo para calcular o Fatorial de um número lido pelo usuário. Inicialmente será apresentado o cálculo do Fatorial sem a recursividade.

**Figura 6** – Algoritmo Fatorial sem a utilização da recursividade

```
17      int numfat = 0;
18      double fatorial = 0;
19
20
21      System.out.print("Fatorial de qual Número.. ");
22      numfat = leia.nextInt();
23
24      fatorial = fatorial(numfat);
25
26      System.out.println("O fatorial de " + numfat + " é: " + fatorial);
27
28
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Perceba que a partir de um valor fornecido pelo usuário, linha 22 da figura 6, esse valor é passado por parâmetro para a função **fatorial** na linha 24. A função retorna o cálculo do fatorial e armazena o resultado na variável **fatorial** também na linha 24.

A implementação da função fatorial é descrita na figura 7.

**Figura 7** – Função fatorial sem a utilização de recursividade

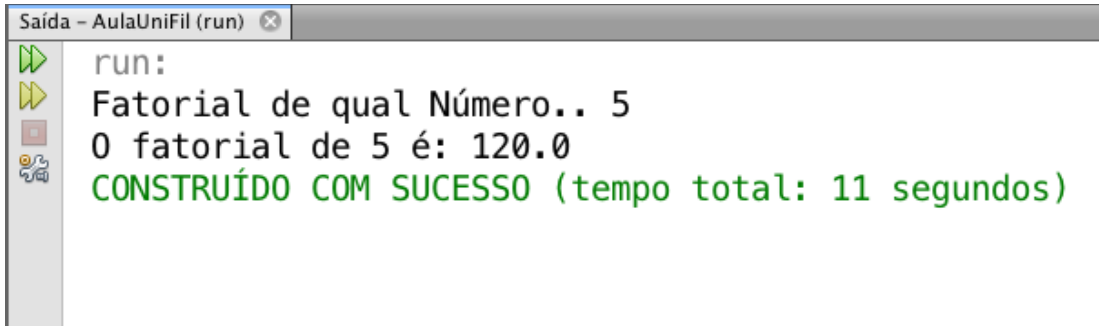
```
64      public static double fatorial(double pnum){
65
66          double fat = 1;
67
68          for(int i = 1; i <= pnum; i++){
69              fat = fat * i;
70          }
71
72          return fat;
73      }
74
75
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Explicando a Figura 7, a função recebe por parâmetro um valor numérico que é passado para o limite de execução do laço. O laço por sua vez inicia no número 1 e vai multiplicando até o valor passado por parâmetro. No nosso exemplo, se o valor 5 for passado por parâmetro, o cálculo da função **fatorial** será:  $(1 * 2 * 3 * 4 * 5) = 120$ . O resultado é armazenado na variável **fat**, que por sua vez é retornado pela função na linha 73.

A tela de execução desse método pode ser visualizada na figura 8.

**Figura 8** – Tela de Execução da função fatorial

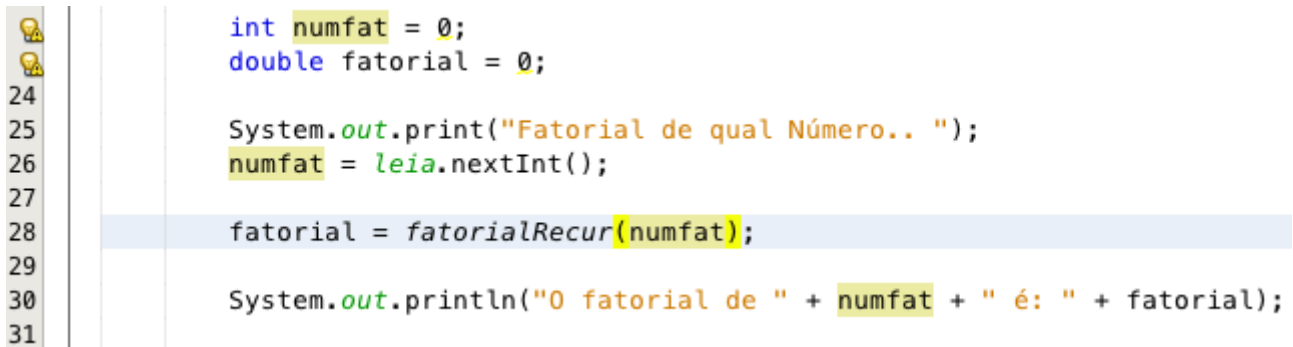


```
Saída - AulaUniFil (run) x
run:
Fatorial de qual Número.. 5
0 fatorial de 5 é: 120.0
CONSTRUÍDO COM SUCESSO (tempo total: 11 segundos)
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Nesse momento será abordado o algoritmo do fatorial considerando uma função recursiva para resolução do problema. Observe a figura 9 abaixo:

**Figura 9** – Algoritmo Fatorial com a utilização da recursividade



```
int numfat = 0;
double fatorial = 0;

System.out.print("Fatorial de qual Número.. ");
numfat = leia.nextInt();

fatorial = fatorialRecur(numfat);

System.out.println("0 fatorial de " + numfat + " é: " + fatorial);
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Perceba que na figura 9, a função chamada na linha 28, **fatorialRecur**, precisa de um número fornecido pelo usuário para fazer o cálculo do fatorial. Esse valor é passado por parâmetro para a função que realiza esse cálculo, como pode ser observado na figura 10.

**Figura 10** – Função fatorial com a utilização de recursividade

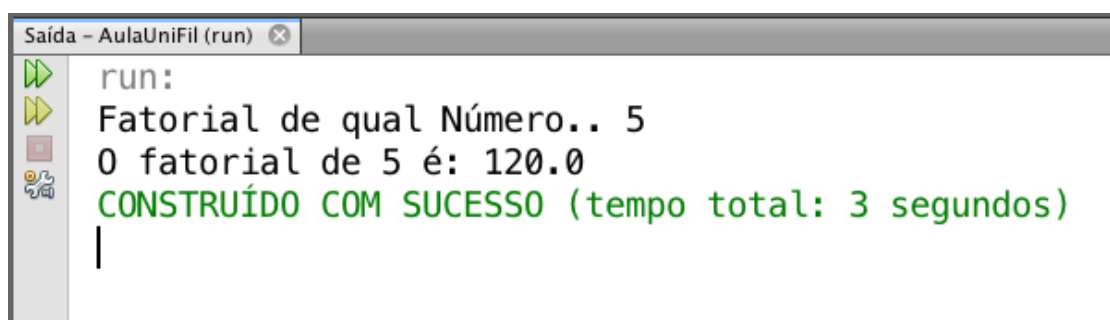
```
80 public static int fatorialRecur(int x){
81
82     if(x == 0){
83         return 1;
84     }
85
86     return x * fatorialRecur(x - 1);
87
88 }
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Explicando a figura 10, a função recebe por parâmetro um valor numérico. Se esse valor for igual a 0(zero) a função devolve o número 1, caso contrário, ela retorna chamando recursivamente a própria função, porém passando por parâmetro  $(x - 1)$ , ou seja, o mesmo valor passado inicialmente menos 1(um). No nosso exemplo, se o valor 5 for passado por parâmetro, o cálculo da função **fatorialRecur** exibe o mesmo resultado que a função sem recursividade.

A tela de execução desse método pode ser visualizada na figura 11.

**Figura 11** – Tela de Execução da função fatorialRecur



A imagem mostra a janela de saída do NetBeans com o título "Saída - AulaUniFil (run)". O conteúdo da janela é o seguinte:

```
run:
Fatorial de qual Número.. 5
0 fatorial de 5 é: 120.0
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

## Videoaula 2

Nessa segunda videoaula, será abordado os dois exemplos do cálculo do Fatorial de um número qualquer. O primeiro será sem a utilização da recursividade e o segundo utilizando a função recursiva. O professor vai implementar os códigos passo a passo e explicar o comportamento dos algoritmos.

Acompanhe com atenção e não deixe de realizar esses exemplos no seu ambiente de desenvolvimento. Execute o algoritmo passando diferentes valores para testes.

O teste de mesa para essa função continua sendo imprescindível para que você compreenda em detalhes tudo o que está acontecendo no algoritmo.



### Videoaula 2

Utilize o QRcode para assistir!

Nessa segunda videoaula, será abordado os dois exemplos do cálculo do Fatorial de um número qualquer.



No decorrer do seu estudo, você pode consultar os livros disponíveis nas Referências da disciplina para aprofundar ainda mais seu conhecimento. Veja a indicação de um material bem interessante que separamos para você.

## Indicação de Leitura

Nesse material desenvolvido pelo Instituto Federal do Espírito Santo – IFES em parceria com a Universidade Federal de Santa Catarina - UFSC, disponível no *link* abaixo, você vai encontrar na página 80, o cálculo do fatorial de um número utilizando funções recursivas. O material possui outros conteúdos na linguagem Java como: manipulação de arquivos, listas, pilhas, assim como, aspectos fundamentais do Java e informações sobre a Plataforma Java. Para maiores informações acesse:

Disponível em: [http://pronatec.ifpr.edu.br/wp-content/uploads/2012/07/Prog\\_de\\_Sistema.pdf](http://pronatec.ifpr.edu.br/wp-content/uploads/2012/07/Prog_de_Sistema.pdf). Acesso em: 20 jul. 2021.

Aqui você vai encontrar informações do funcionamento da recursividade, assim como, listas de exercícios para aprimorar seu conhecimento.

## Cálculo de Fibonacci

Um outro exemplo muito interessante de se trabalhar com recursividade é a sequência de Fibonacci. Esta sequência, composta por números, foi descrita pelo matemático italiano Leonardo de Pisa (1170-1250), conhecido como Fibonacci, no final do século 12. Ele percebeu uma regularidade matemática a partir do problema proposto que era a criação de coelhos (CORMEN, 2012).

A fórmula que Fibonacci está demonstrada na figura 12.

### Figura 12 – Sequência de Fibonacci

$F_n = F_{n-1} + F_{n-2}$
$3 + 2 = 5$
$5 + 3 = 8$
$8 + 5 = 13$
$13 + 8 = 21$
$21 + 13 = 34$
$34 + 21 = 55$

Fonte: o autor (2021)

Como podemos observar na figura 12, a regra para a formação do próximo termo é a soma dos dois termos antecessores.

Vamos demonstrar o algoritmo para a solução desse problema na figura 13.

**Figura 13 – Algoritmo de Fibonacci**

```
17 static long fibonacci(int n){
18
19     if(n < 2){
20         return n;
21     }else{
22
23         return fibonacci(n - 1) + fibonacci(n - 2);
24
25     }
26 }
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

A função ***fibo()***, solicita por parâmetro um número que será utilizado recursivamente em sua chamada no método ***main()***. A partir desse número, é produzida a sequência conforme apresentada na figura 12. Para chamar a função no método ***main()*** e apresentar a sequência dos termos da série de Fibonacci, foi implementado um laço que vai do primeiro termo até termo final que se deseja visualizar. Observe a figura 14.

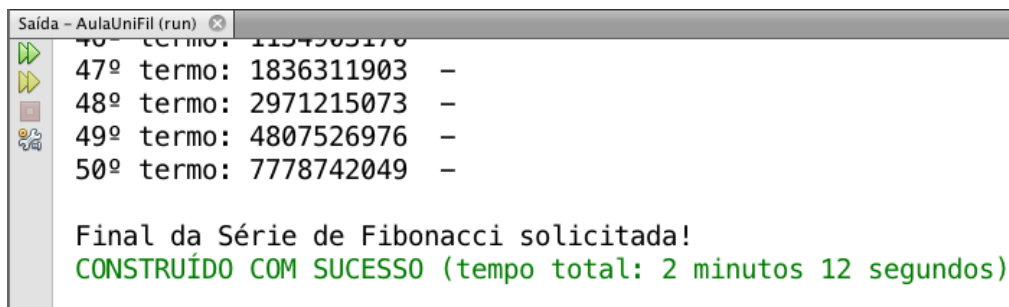
**Figura 14** – Execução da função de Fibonacci

```
7 public static void main(String[] args) {  
8  
9     for(int i = 0; i <= 49; i++){  
10  
11         System.out.print((i + 1) + "º termo: " + fibo(i) + " - ");  
12  
13     }  
14     System.out.println("");  
15     System.out.println("Final da Série de Fibonacci solicitada!");  
16 }
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

Repare que para ver até o 50º (quinquagésimo) termo da série de Fibonacci, foi passado no laço o intervalo de 0 a 49. O resultado é apresentado na figura 15.

**Figura 15** – Apresentação da Sequência de Fibonacci



```
Saída - AulaUniFil (run) x  
47º termo: 1836311903 -  
48º termo: 2971215073 -  
49º termo: 4807526976 -  
50º termo: 7778742049 -  
  
Final da Série de Fibonacci solicitada!  
CONSTRUÍDO COM SUCESSO (tempo total: 2 minutos 12 segundos)
```

Fonte: elaborado pelo autor via IDE NetBeans (2021)

### Videoaula 3

Nessa videoaula 3, será abordado o algoritmo para resolver a sequência de Fibonacci. Por meio de um exemplo prático, será demonstrado como se chega em qualquer termo dessa sequência, bem como, a utilização da recursividade para a resolução desse problema. O professor irá implementar um código em Java e abordará características e funcionalidades desse algoritmo.

Acompanhe com atenção e não deixe de realizar no seu ambiente de desenvolvimento, seja NetBeans, Eclipse ou IntelliJ, todas as implementações, alterações e valores de entrada que estão sendo realizadas no código de execução. Aproveite também para fazer o *debugging* do algoritmo e entender um pouco mais do seu funcionamento.





### Videoaula 3

Utilize o QRcode para assistir!

Acompanhe com atenção e não deixe de realizar no seu ambiente de desenvolvimento, seja NetBeans, Eclipse ou IntelliJ, todas as implementações, alterações e valores de entrada que estão sendo realizadas no código de execução.



Perceba que para a apresentação dos primeiros 50 termos da Sequência de Fibonacci, o algoritmo em Java levou aproximadamente 2 minutos e 12 segundos. Lembrando que esse tempo vai depender do processador e memória disponível do equipamento que você está utilizando, bem como, dos aplicativos que estão sendo executados em segundo plano.

## Encerramento da Unidade

Chegamos ao fim desta Unidade de Ensino, após caminhar por conteúdos importantes para o seu desenvolvimento acadêmico. A Unidade 2 apresentou a técnica de ordenação *Merge Sort* dentro da linguagem de programação Java. Foram detalhados processos, como a criação dos *Arrays* e a manipulação dos seus elementos para realização da ordenação de uma determinada estrutura de dados.

Após a assimilação dos conteúdos abordados nesta Unidade, você estará apto a prosseguir estudando um pouco mais sobre a Análise e Projeto de Algoritmos nas próximas Unidades de Ensino.

Tenha uma ótima leitura e não se esqueça de ler e estudar a Bibliografia e as videoaulas ministradas pelo professor.

Obrigado pela oportunidade de ser o mediador entre os conteúdos específicos abordados e o conhecimento adquirido por você por meio da didática e abordagem desta Unidade.

## Referências

- CORMEN, T. H. **Desmistificando algoritmos**. Rio de Janeiro: Campus, 2012. 926 p.
- DEITEL, H. M.; DEITEL, P. J. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010.
- FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.
- JANDL JUNIOR, P. **Java: guia do programador**. São Paulo: Novatec, 2007. 681 p.
- LAUREANO, M. **Estrutura de dados com algoritmos e C**. São Paulo: Brasport, 2012.
- PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados: com aplicações em Java**. São Paulo: Pearson, 2016.
- RECURSIVIDADE. *In*: **DICIO**. Dicionário Online de Português. Porto: 7Graus, 2021. Disponível em: <https://www.dicio.com.br/recursividade/>. Acesso em: 20 jul. 2021.
- SANTANA, A. L. **Técnicas de Programação: Curso Técnico em Informática**. Colatina: IFES, 2011.
- XAVIER, G. F. C. **Lógica de programação**. São Paulo: Senac, 2018. 322 p.



UNIFIL.BR