

Estruturas de Dados

Conteúdo

- Ordenação.
- Algoritmos de ordenação.
 - Bubble Sort.
 - Selection Sort.
 - Insertion Sort.
 - Shell Sort.

Ordenação

- Ordenação, classificação, sorting (inglês).

Dicionário Houaiss, 2009:

- **ordenar** ato ou efeito de ordenar
- **ordenar** dispor de forma organizada; arrumar, organizar

"A classificação é geralmente entendido como sendo o processo de rearranjar um conjunto dado de objetos em uma ordem específica. O propósito de ordenação é facilitar a pesquisa posterior dos membros do conjunto classificado. "

Niklaus Wirth: Algorithms and Data Structures

São muitos os exemplos de objetos classificados: nas listas telefônicas, nos índices dos livros, nos catálogos das bibliotecas, as palavras nos dicionários, nas listas de alunos nas escolas, na agenda do celular...

Ordenação em memória interna ou externa

Ordenação em memória interna (memória primária ou principal)

- a ordenação é totalmente feita na memória primária (utilizando vetores, listas enlaçadas, árvores binárias etc.);
- é utilizada quando a memória do computador (RAM) é suficiente para armazenar todos os elementos a serem ordenados;
- observar que a memória RAM é mais rápida, mas é volátil;
- os objetos poderiam ser ordenados *In Loco* ou poderia ser utilizada memória interna adicional.

Ordenação em memória externa

- é utilizada quando os elementos não cabem na memória primária: se encontram armazenados em HD, SSD ou outros dispositivos externos;
- um HD e dispositivos externos de armazenamento são, no estado atual da tecnologia, mais lentos que a memória interna;
- uma parte dos elementos a serem classificados poderiam ser armazenados temporariamente em memória primária ou principal (RAM) e blocos de itens seriam trocados com a memória externa (mecanismo de *swapping*).

Alguns métodos de ordenação

Neste material estudaremos quatro algoritmos ou métodos de ordenação:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort

No próximo material estudaremos recursividade e os métodos de ordenação:

- Quick sort
- Merge Sort

Mas existem outros métodos de ordenação, por exemplo:

- Heap Sort
- Ordenação com árvores binárias
- Métodos híbridos



Bubble Sort

(método da bolha, borbulha)

Ordenação Bubble Sort - algoritmo geral



Ciclo externo i , n repetições (tamanho do vetor), enquanto seja efetuada pelo menos uma troca:

Ciclo interno j :

- Começando sempre na posição $j=0$, comparar elementos sucessivos verificando se $\text{vet}[j] > \text{vet}[j+1]$. Neste caso efetuaremos a troca entre esses dois valores, de forma a "borbulhar" (empurrar) o maior valor para o final do vetor.
- Como o maior valor ficará no final do vetor, nas **iterações de j** chegaremos somente **até o (tamanho do vetor - 1) - i**

Fim do ciclo externo i

Ordenação Bubble Sort (figura com a primeira varredura)

0	1	2	3	4	5	6	7	8
23	17	5	90	12	44	38	84	77

↑ exchange

17	23	5	90	12	44	38	84	77
----	----	---	----	----	----	----	----	----

↑ exchange

17	5	23	90	12	44	38	84	77
----	---	----	----	----	----	----	----	----

↑ ok ↑ exchange

17	5	23	12	90	44	38	84	77
----	---	----	----	----	----	----	----	----

↑ exchange

17	5	23	12	44	90	38	84	77
----	---	----	----	----	----	----	----	----

exchange ↑

17	5	23	12	44	38	90	84	77
----	---	----	----	----	----	----	----	----

exchange ↑

17	5	23	12	44	38	84	90	77
----	---	----	----	----	----	----	----	----

exchange ↑

17	5	23	12	44	38	84	77	90
----	---	----	----	----	----	----	----	----

The largest value 90 is at the end of the list.

A figura mostra apenas a etapa inicial (ciclo externo $i=0$), começando na posição $j=0$ e chegando na penúltima posição, $j=7$. Na próxima etapa (ciclo $i=1$) começaremos na posição $j=0$, até a posição $j=6$. **Em cada etapa** serão efetuadas comparações e trocas e **o maior valor ficará na última posição analisada**.

Ordenação Bubble Sort (método da bolha)

<http://www.youtube.com/watch?v=lyZQPjUT5B4>

Demonstração

Bubble-sort with Hungarian ("Csángó") folk dance,
Sapientia University (Romania).

A eficiência do Bubble Sort é de $O(n^2)$ no pior caso e na média, ou seja, a ordenação poderá ser resolvida em tempo polinomial de grau 2 (função quadrática). No melhor caso a eficiência é de $O(n)$.

Não é um algoritmo tão eficiente como outros que tenham $O(\log_2 n)$.

Ordenação Bubble Sort (método da bolha) v. 01

```
public boolean bubbleSort (double vetor []) {  
    if (vetor == null) return false;  
    for (int i = 0; i < vetor.length - 1; i++) {  
        for (int j = 0; j < vetor.length - 1 - i; j++) {  
            if (vetor[j] > vetor[j+1]) {  
                double tmp = vetor[j];  
                vetor[j] = vetor[j+1];  
                vetor[j+1] = tmp;  
            }  
        }  
    }  
    return true;  
}
```



Esta lógica de troca poderia ficar em um método separado e chamar: **troca(vetor, j, j+1);**
Veja em bubbleSort_v05.

Caso não aconteçam trocas poderíamos sair do método: veja uma solução um pouco mais eficiente em slides a seguir.

Simplificando, se analisamos a quantidade de comparações necessárias, temos:
 $n + (n-1) + (n-2) + (n-3) + \dots$
aproximadamente $n * n$
o que caracteriza este método com complexidade média de **$O(n^2)$** , sendo n o tamanho do vetor (a qtde. de elementos).

Ordenação Bubble Sort (método da bolha) v. 02

```
public void bubbleSort_v02 (double vetor []) throws Exception {  
    if (vetor == null) throw new Exception();
```

```
    ...
```

```
}
```

Nesta versão do método não retornamos um valor lógico, mas lançaremos uma exceção caso encontremos algum problema. Quem chame este método poderá capturar a *exception*.

Ordenação Bubble Sort (método da bolha) v. 04

```
public boolean bubbleSort_v04 (double vetor []) {  
    if (vetor == null) return false;  
    for (int i = 0; i < vetor.length - 1; i++) {  
        int trocas = 0; //ou utilizar: boolean trocou = false;  
        for (int j = 0; j < vetor.length - 1 - i; j++) {  
            if (vetor[j] > vetor[j+1]) {  
                double tmp = vetor[j];  
                vetor[j] = vetor[j+1];  
                vetor[j+1] = tmp;  
                trocas++; // trocou = true;  
            }  
        }  
        if(trocas==0)break; // if ( !trocou ) break; // comando para sair do ciclo externo  
    }  
    return true;  
}
```

A **otimização** aqui mostrada permite abandonar o ciclo externo e o método quando a lista de valores já se encontra ordenada.

Ordenação Bubble Sort (método da bolha) v. 05

```
public boolean bubbleSort_v05 (double vetor []) {  
    if (vetor == null) return false;  
    boolean trocou;    int i = 0;  
    do {  
        trocou = false;  
        for (int j = 0; j < vetor.length - 1 - i; j++) {  
            if (vetor[j] > vetor[j+1]) {  
                troca(vetor, j, j+1);  
                trocou = true;  
            }  
        }  
        i++;  
    } while (trocou);  
    return true;  
}
```

A **otimização** aqui mostrada permite abandonar o ciclo externo e o método quando a lista de valores já se encontra ordenada.

Resumo do método Bubble Sort

Observe que serão necessárias $n*n$ comparações, o que caracteriza o método como de **$O(n^2)$** na média, é um algoritmo com complexidade polinomial (quadrática), mais ineficiente se comparado com outros métodos.

Por exemplo, nos casos melhores: para um vetor com um único elemento fora de lugar teremos $O(2*n)$, que equivale a $O(n)$, e para uma lista completamente organizada será de **$O(n)$** . Para casos piores: **$O(n^2)$** .

Valores iniciais	44	55	12	42	94	18	06	67
	06	55	12	42	94	18	44	67
	06	12	55	42	94	18	44	67
							44	67
							44	67
							94	67
							94	67
							67	94
							i=6	

Selection Sort

Selection Sort (seleção)

Ordenação Selection Sort (seleção do menor valor)



Algoritmo geral

Para cada posição **i** de 0 até a penúltima posição do vetor:

Ciclo interno j:

- Analisar cada elemento a partir de **j = i + 1** até o final do vetor e determinar qual é o **menor** valor desse trecho à direita.
- Trocar esse **menor** valor com o elemento que se encontrava na **posição i**.

Fim do ciclo externo i

Ordenação Selection Sort (seleção do menor valor)

Valores iniciais	44	55	12	42	94	18	06	67
i=0	06	55	12	42	94	18	44	67
i=1	06	12	55	42	94	18	44	67
i=2	06	12	18	42	94	55	44	67
i=3	06	12	18	42	94	55	44	67
i=4	06	12	18	42	44	55	94	67
i=5	06	12	18	42	44	55	94	67
i=6	06	12	18	42	44	55	67	94
	i=0	i=1	i=2	i=3	i=4	i=5	i=6	

Para i=0, observe nos "valores iniciais" que o menor valor é 06. Será trocado com 44.

Para i=1, observe que o menor nos valores restantes é 12. Será trocado com 55.

Para i=2, observe que o menor nos valores restantes é 18. Será trocado com 55.

Para i=3, o menor valor (dos restantes) é o 42. Não precisa fazer troca.

Para i=4, observe que o menor nos valores restantes é 44. Será trocado com 94.

Para i=5, o menor valor é o 55. Não precisa fazer troca.

Para i=6, observe que o menor nos valores restantes é 67. Será trocado com 94.

Ordenação Selection Sort



Demonstração do método em:

Selection Sort | Autor: GeeksforGeeks

<https://www.youtube.com/watch?v=xWBP4IzkoyM>

Ordenação Selection Sort (seleção)

```
public boolean selectionSort (double vetor []) {  
    if (vetor == null) return false;  
    for (int i = 0; i < vetor.length - 1; i++) {  
        // Determina a posição min do menor valor encontrado entre as  
        // posições i+1 e vetor.length-1. Desta forma, estamos selecionando o menor  
        // valor, para efetuar uma troca.  
        int min = i;  
        for (int j = i+1; j < vetor.length; j++) {  
            if ( vetor[j] < vetor[min] ) min = j;  
        }  
        // troca os valores:  
        double temp = vetor[i];  
        vetor[i] = vetor[min];  
        vetor[min] = temp;  
    }  
    return true;  
}
```

Resumo: o método de ordenação **Selection Sort** também possui eficiência média de **$O(n^2)$** , porque:

$$(n-1) + (n-2) \dots + 2 + 1 \in O(n^2)$$

Observe que a eficiência é sempre de $O(n^2)$, em qualquer caso (melhor, pior e médio), independente da ordenação inicial do vetor.

Selection Sort é, possivelmente, o método mais simples para compreender e o pior em desempenho.

Ordenação Selection Sort (seleção) - uma variação

```
public boolean selectionSort (double vetor []) {  
    if (vetor == null) return false;  
    for (int i = 0; i < vetor.length - 1; i++) {  
        int posMenor = i;  
        double menor = vetor[i];  
        for (int j = i+1; j < vetor.length; j++) {  
            if ( vetor[j] < menor ) {  
                posMenor = j;  
                menor = vetor[j];  
            }  
        }  
        //trocamos vetor[i] com vetor[posMenor]:  
        double temp = vetor[i];  
        vetor[i] = menor; //ou vetor[i] = vetor[posMenor];  
        vetor[posMenor] = temp;  
    }  
    return true;  
}
```

No projeto fornecido encontrará três versões do algoritmo Selection Sort: a v02 lança uma exception.

Vetor inicial:

9.5 6.3 1.2 78.3 0.5 4.3

i=0 9.5 6.3 1.2 78.3 0.5 4.3

i=1 6.3 9.5 1.2 78.3 0.5 4.3

Insertion Sort

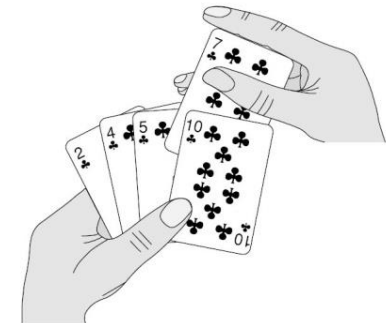
0.5 4.3

0.5

8.3

i=5 0.5 1.2 4.3 6.3 9.5

i=0 i=1 i=2 i=3 i=4



Insertion Sort (inserção)

Ordenação Insertion Sort



Algoritmo geral

Para cada posição **i** de 1 até a última posição do vetor:

- Guardar o valor (item) que está na posição **i** em uma variável temporária **x**.
- Com um ciclo interno **j** começando em **i**, fazer uma varredura com **j** diminuindo de direita para esquerda, enquanto **j** > 0 e **x** for menor que o elemento na posição **j-1**: deslocar o valor (item) em **j-1** para direita, de forma a **abrir** um espaço para inserção.
- Ao concluir o ciclo anterior, colocar o valor **x** na posição de inserção **j**.

Fim do ciclo externo **i**

Ordenação Insertion Sort (inserção na posição correta)

Insertion Sort (analizando a etapa $i=3, j=3$)

inicialmente:	9.5	6.3	1.2	5.4	0.5	4.3
$i=0$	9.5	6.3	1.2	5.4	0.5	4.3
$i=1$	6.3	9.5	1.2	5.4	0.5	4.3
$i=2$	1.2	6.3	9.5	5.4	0.5	4.3
$i=3$	1.2	5.4	6.3	9.5	0.5	4.3
$i=4$	0.5	1.2	5.4	6.3	9.5	4.3
$i=5$	0.5	1.2	4.3	5.4	6.3	9.5

$j = 3$

Ao analisar a posição 3, o valor 5.4 será guardado na variável temporária x . Retrocedendo para a esquerda, os valores 6.3 e 9.5 serão movidos para a direita (porque são maiores que $x=5.4$). Ao comparar $1.2 \leq x$ encontramos que a posição correta de inserção é $j=1$, e colocamos o valor x na posição j vazia, com $a[j] = x$; ou seja, $a[1] = 5.4$;

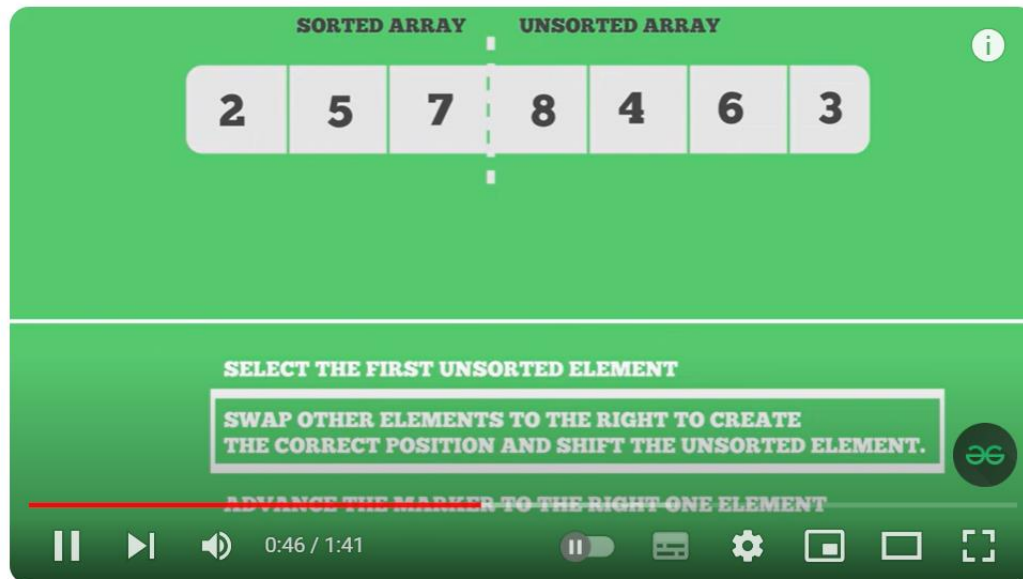
O princípio do algoritmo consiste em encontrar e colocar (**abrir espaço e inserir**) o elemento analisado na posição correta onde ele ficaria, por enquanto, dentro do vetor.

Observe que, para cada posição i analisada (ciclo externo), analisaremos apenas as posições anteriores (ciclo interno j).

A parada do ciclo interno j acontecerá quando for encontrada um valor $\leq x$ ou quando chegarmos no início do vetor ($j=0$).

Ordenação Insertion Sort

Em <https://www.youtube.com/watch?v=OGzPmgsl-pQ> de GeeksforGeeks, encontraremos uma demonstração.



Ordenação Insertion Sort (inserção)

```
public boolean insertionSort (double a []) {  
    if (a == null) return false;  
    int i, j; double x;  
    for ( i=1; i < a.length; i++ ) {  
        x = a[i]; j = i;  
        while ( j > 0 && x < a[j-1] ) {  
            a[j] = a[j-1]; // move para direita => abre espaço  
            j--; // para apontar ao item anterior  
        }  
        a[j] = x; // coloca x na posição j  
    }  
    return true;  
}
```

Para cada posição i (ciclo externo, que começa em 1 até a última posição), analisaremos apenas as posições anteriores com o ciclo interno j .

A parada do ciclo interno j acontecerá quando for encontrada a posição correta de inserção do item analisado ou quando chegarmos no início do vetor (se $j=0$).

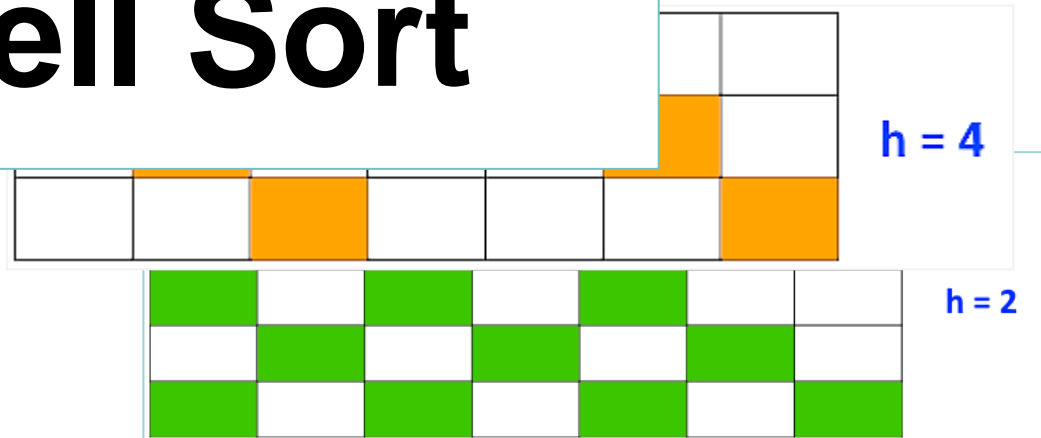
Insertion Sort

Este método de ordenação também possui complexidade quadrática de $O(n^2)$.

Este algoritmo tem uma lógica um pouco mais complexa que os anteriores.

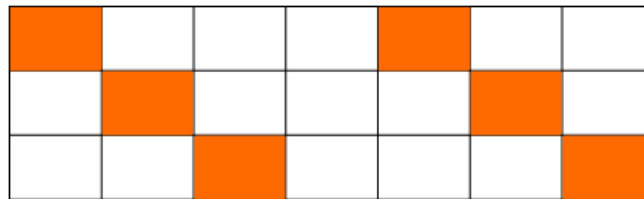
Existem algumas otimizações para este método, mas continuará sendo de $O(n^2)$ para os casos piores. Para o melhor caso (lista ordenada onde $x < a[j-1]$ falha), teremos $O(n)$.

Shell Sort



Método de ordenação Shell Sort

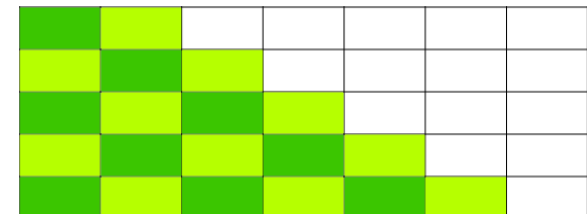
- Proposto por Donald Shell (1924-2015, EUA) em 1959, é **uma extensão (mais complexa) do algoritmo de ordenação por inserção**. No método de inserção são comparados elementos consecutivos do vetor (adjacentes, de distância 1). No método de Shell serão comparados elementos que se encontram a uma distância h (elementos distantes), e se modificará essa distância, seguindo alguma sequência, até chegar em $h=1$ (como acontece no método de inserção).
- Existem muitos estudos sobre a seleção correta da sequência de distâncias h , o que influenciará diretamente na eficiência deste algoritmo.
- Ainda que seja difícil demonstrar sua complexidade, se afirma que este método de ordenação está na média entre $O(n^{1,5})$ e $O(n^{1,25})$, que é algo melhor que $O(n^2)$.



$h = 4$



$h = 2$



$h = 1$

(mét. inserção)

Métodos de ordenação de complexidade $O(n^2)$ estudados

O três algoritmos estudados apresentam no caso médio uma complexidade em tempo de $O(n^2)$, que é uma função quadrática, ou seja, um polinômio de grau 2.

O **Insertion Sort** e o **Bubble Sort** atingem $O(n)$ em seus melhores casos (para vetores quase ordenados) e para outros casos serão de $O(n^2)$.

O **Selection Sort** é de $O(n^2)$ em qualquer situação (melhor, médio e pior caso), ou seja, independente da ordenação prévia do vetor. Talvez o Selection Sort seja o método que tem a lógica mais simples, mas é o pior quanto a desempenho.

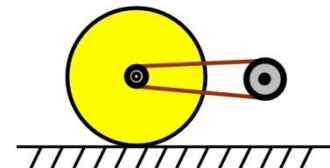
O método **Shell Sort**, variação do Insertion Sort (não incluso na tabela abaixo) pode chegar na média de $O(n^{1,25})$. Então seria o melhor dos quatro métodos estudados.

Método de ordenação	complexidade (tempo)		
	melhor caso	médio	pior caso
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

Obs.: estes métodos não utilizam praticamente memória adicional, são de $O(1)$ quanto ao espaço.

Métodos de ordenação estudados - observações finais

- Os métodos que foram mostrados ordenam em ordem **crescente**. Para ordenar em ordem **decrescente** podemos alterar a comparação no **comando if**.
- Os métodos mostrados ordenam valores reais (vetores de tipo double). Seria simples alterar estes métodos para ordenar outros tipos de valores (vetores de valores inteiros, Strings, chars...).
- Os métodos mostrados utilizam um vetor simples. Um vetor de objetos, por exemplo, também poderia ser ordenado por qualquer atributo da classe. Analise como alterar o `double vetor[]` por um vetor de objetos.



Exercício 1 - para entregar

Utilize (adapte) o método de ordenação Bubble Sort para **ordenar textos** em ordem crescente, de forma a ordenar alfabeticamente um vetor com nomes de países:

```
String paises[] = {"México", "Brasil", "Cuba", "Chile", "Argentina", "Espanha"};
```

Implemente o método e os testes. Envie um projeto completo.

Sugestão, procure informação sobre:

`compareTo`

`compareToIgnoreCase`

[https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#compareToIgnoreCase\(java.lang.String\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#compareToIgnoreCase(java.lang.String))

Exercício 2 - para entregar

Utilize (adapte) o método de ordenação Selection Sort para ordenar **valores inteiros**, neste caso para ordenar **de maior a menor** um vetor com números inteiros:

```
int numeros [] = { 4, 2, 10, 123, -3, 32, 0, 34, 12, 91, 45, 3, 21, 87, 61};
```

Implemente o método e os testes. Envie um projeto completo.

Bibliografia sugerida para a disciplina

BIBLIOGRAFIA BÁSICA	BIBLIOGRAFIA COMPLEMENTAR
CORMEN, T. H.; et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.	ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010. (eBook)
GOODRICH, M. T.; TAMASSIA, R. Estrutura de dados e algoritmos em java. 5. ed. Porto Alegre: Bookman, 2013. (livro físico e e-book)	PUGA, S.; RISSETTI, G. Lógica de programação e estruturas de dados, com aplicações em Java. 3. ed. São Paulo: Pearson Education do Brasil, 2016. (eBook)
CURY, T. E., BARRETO, J. S., SARAIVA, M. O., et al. Estrutura de Dados (1. ed.) ISBN 9788595024328, Porto Alegre: SAGAH, 2018 (e-book)	DEITEL, P.; DEITEL, H. Java como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017. (eBook)
	BARNES, D. J.; KOLLING, M. Programação Orientada a Objetos com Java: uma introdução prática usando o Blue J. São Paulo: Pearson Prentice Hall, 2004. (eBook)
	BORIN, V. POZZOBON. Estrutura de Dados. ISBN: 9786557451595, Edição: 1ª. Curitiba: Contentus, 2020 (e-book)

Sapientia University. Bubble-sort with Hungarian folk dance.

<http://www.youtube.com/watch?v=lyZQPjUT5B4>

GeeksforGeeks , Insertion Sort. <https://www.youtube.com/watch?v=OGzPmgsl-pQ>

GeeksforGeeks , Selection Sort. <https://www.youtube.com/watch?v=xWBP4IzkoyM>

Ishida, Moriteru. Algoritmos: Explicados e Animados

<https://play.google.com/store/apps/details?id=wiki.algorithm.algorithms>