

## Ordenação de estruturas lineares

Prof. Marcelo de Souza

UDESC Ibirama  
Bacharelado em Engenharia de Software

marcelo.desouza@udesc.br  
Versão compilada em 13 de agosto de 2020

Leitura obrigatória:

- Capítulo 4 de [Ziviani \[2010\]](#) – Ordenação.

Leitura complementar:

- Capítulo 13 de [Pereira \[2008\]](#) – Ordenação e busca.
- Capítulo 15 de [Preiss \[2001\]](#) – Algoritmos de ordenação e ordenadores.

### Algoritmos de ordenação

- Dada uma coleção de elementos, devolve a coleção ordenada.
- Existem diversos algoritmos diferentes para ordenação.
- Veja demonstrações dos algoritmos em:
  - <https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>.
  - <https://visualgo.net/en/sorting>.

### Algoritmo *BubbleSort*

- Algoritmo baseado em trocas.
  - Funcionamento:
    - Percorre elementos vizinhos e os inverte (*swap*) quando estão fora de ordem.
-

- Na primeira passagem, maior elemento vai para última posição.
- Na segunda passagem, segundo maior elemento vai para penúltima posição.
- Etc.
- Complexidade  $O(n^2)$ .
- Veja o funcionamento do algoritmo em <https://bit.ly/1wFxLr0>.

### Algoritmo BubbleSort:

```
1  public class BubbleSort {
2      public void sort(int[] array) {
3          for(int i = 0; i < array.length; i++)
4              for(int j = 0; j < array.length - (i + 1); j++)
5                  if(array[j] > array[j + 1])
6                      swap(j, j + 1, array);
7      }
8
9      private void swap(int i, int j, int[] array) {
10         int temp = array[i];
11         array[i] = array[j];
12         array[j] = temp;
13     }
14 }
```

### Comentários:

- Cada vez que percorre o vetor vai até a última posição não-ordenada.
- Método swap troca elementos de posição.

### Algoritmo *SelectionSort*

- Algoritmo baseado em seleção.
- Funcionamento:
  - Percorre a lista para encontrar o menor valor.
  - Troca o menor valor com o primeiro elemento.
  - Repete o processo para a sub-lista ainda não ordenada.
- Complexidade  $O(n^2)$ .
- Veja o funcionamento do algoritmo em <https://bit.ly/1wFxLr0>.

Algoritmo SelectionSort:

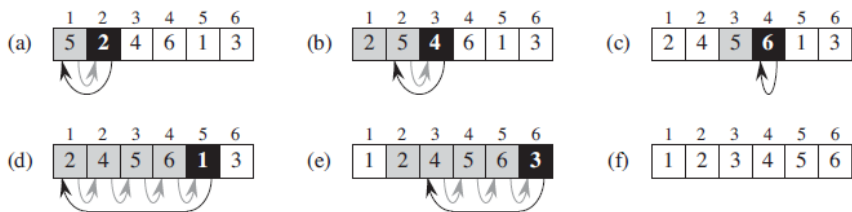
```
1  public class SelectionSort {
2      public void sort(int[] array) {
3          for(int i = 0; i < array.length - 1; i++) {
4              int min = i;
5              for(int j = i + 1; j < array.length; j++)
6                  if(array[j] < array[min])
7                      min = j;
8              swap(min, i, array);
9          }
10     }
11
12     private void swap(int i, int j, int[] array) {
13         int temp = array[i];
14         array[i] = array[j];
15         array[j] = temp;
16     }
17 }
```

Comentários:

- A cada iteração o menor elemento é levado para a frente.

## Algoritmo *InsertionSort*

- Algoritmo baseado em inserção.
- Funcionamento:
  - Em uma lista de 0 até  $n - 1$ , inicializa  $i = 1$ .
  - Percorre a lista de  $i$  até 0 e posiciona o elemento  $i$ .
  - Incrementa  $i$  e volta ao passo anterior.
- Complexidade  $O(n^2)$ .
- Veja o funcionamento do algoritmo em <https://bit.ly/1wFxFr0>.



Algoritmo InsertionSort:

```

1  public class InsertionSort {
2      public void sort(int[] array) {
3          int temp;
4          for(int i = 1; i < array.length; i++) {
5              for(int j = i ; j > 0 ; j--)
6                  if(array[j] < array[j - 1])
7                      swap(j, j - 1, array);
8          }
9
10     private void swap(int i, int j, int[] array) {
11         int temp = array[i];

```

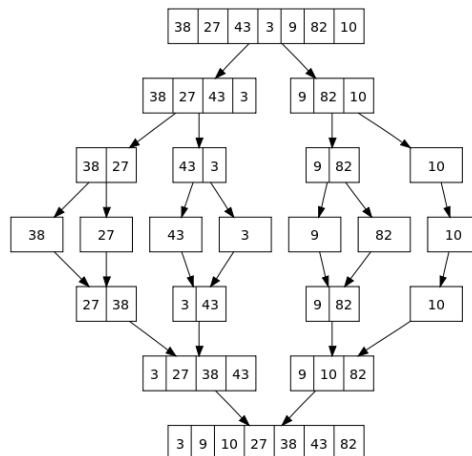
```
12     array[i] = array[j];  
13     array[j] = temp;  
14 }  
15 }
```

### Comentários:

- A cada iteração, temos a lista de 0 até  $i$  ordenada.

### Algoritmo MergeSort

- Algoritmo baseado no conceito de *divisão e conquista*.
- Funcionamento:
  - Iterativamente divide o vetor pela metade, até que cada parte contenha um elemento.
  - Iterativamente junta as partes (*merge*), ordenando os elementos.
- Complexidade  $O(n \log n)$ .
- Veja o funcionamento do algoritmo em <https://bit.ly/1wFxlR0>.



### Algoritmo MergeSort:

```
1  public class MergeSort {
2
3      int[] tempMergArr;
4
5      public void sort(int array[]) {
6          tempMergArr = new int[array.length];
7          mergeSort(array, 0, array.length - 1);
8      }
9
10     private void mergeSort(int[] array, int start, int end) {
11         if (start < end) {
12             int middle = start + ((end - start) / 2);
13             mergeSort(array, start, middle);
14             mergeSort(array, middle + 1, end);
15             merge(array, start, middle, end);
16         }
17     }
18
19     private void merge(int[] array, int start, int middle, int end) {
20         for (int i = start; i <= end; i++) {
21             tempMergArr[i] = array[i];
22         }
23
24         int i = start;
25         int j = middle + 1;
26         int k = start;
27         while (i <= middle && j <= end) {
28             if (tempMergArr[i] <= tempMergArr[j]) {
29                 array[k] = tempMergArr[i];
30                 i++;
31             } else {
32                 array[k] = tempMergArr[j];
33                 j++;
34             }
35             k++;
36         }
37     }
38 }
```

```
37
38     while (i <= middle) {
39         array[k] = tempMergArr[i];
40         k++;
41         i++;
42     }
43     while (j <= end) {
44         array[k] = tempMergArr[j];
45         k++;
46         j++;
47     }
48 }
49 }
```

### Comentários:

- O método `mergeSort` é chamado recursivamente com as duas metades, até ser chamado com um único elemento (divisão).
- Após isso, cada parte é juntada pelo método `merge`.
- Na junção, o algoritmo insere no vetor os elementos ordenados das duas metades.
- Faça um teste de mesa para verificar o funcionamento do *merge sort*.

### Atividades

1. Pesquise a respeito de outros algoritmos de ordenação:
  - a. Quick sort
  - b. Shell sort
  - c. Radix sort
  - d. Bucket sort

2. No algoritmo *bubble sort*, se em uma iteração intermediária o vetor for percorrido sem nenhuma troca, significa que os elementos já estão ordenados. Neste caso, não é necessário seguir as próximas etapas do algoritmo. Faça as modificações necessárias para implementar esta estratégia.
3. Modifique o algoritmo InsertionSort para que ele execute uma busca binária para encontrar a posição de inserção do elemento. Qual a nova complexidade do algoritmo?
4. Considere uma matriz retangular. Ordene os elementos de cada linha. Após isso, ordene os elementos de cada coluna. A primeira ordenação foi perdida? Por que?
5. Cada redação do Enem é corrigida por três professores diferentes. A nota final consiste na média aritmética das três notas. Crie um programa para ler as notas de todas as redações e armazenar as médias em um vetor. Ao final, aplique um algoritmo de ordenação nesta estrutura e apresente as dez maiores notas.
6. Modifique o exercício anterior, de modo que sejam armazenadas mais informações de cada redação: nome do aluno, RG do aluno, notas dos professores e média final.

## Referências

- Pereira, S. d. L. (2008). Estruturas de dados fundamentais: Conceitos e aplicações.
- Preiss, B. R. (2001). *Estruturas de dados e algoritmos: padrões de projetos orientados a objetos com Java*. Campus.
- Ziviani, N. (2010). *Projeto de Algoritmos com Implementações em Java e C++*. Cengage Learning.