Centro Universitário de Belo Horizonte

2019

# Relatório:

# Ordenação – TP2

**Aluno**: Douglas José Tertuliano dos Santos

**RA**: 11513928

**Ciência da Computação**

[douglasjtds@gmail.com](mailto:douglasjtds@gmail.com)

[douglasjtds.github.io](https://douglasjtds.github.io)

## 1 – Introdução:

Este trabalho tem como objetivo implementar os principais algoritmos de ordenação – ***BubbleSort***, ***InsertionSort***, ***SelectionSort*** e ***Quicksort*** – para posteriormente analisar os resultados de cada um deles de acordo com o número de comparações, atribuições e tempo de execução; conforme descrito pelo professor Lucas Schmidt em seu edital com instruções. Abaixo segue roteiro e descrição do que foi construído e como foi feito.

Foi utilizada a linguagem *C#* para construção de um *Console* *Application* com o .NET *Framework* 4.7.2. Para compilar e construir esse projeto foi utilizada a IDE *Visual* *Studio* 2019 na sua edição *Community* distribuída gratuitamente pela *Microsoft*.

## 2 – Descrição:

As principais classes utilizadas estão na pasta minhaImplementacao e a classe principal *Program.cs*, as outras classes presentes estão apenas para testes e estudos.

Foi utilizada uma interface *IOrdenacao2.cs* e uma classe que implementa ela, *Ordenacao2.cs*, nessa classe é feita toda a lógica dos métodos de ordenação; exceto o *Quicksort*, pois foi feito uma classe específica para ele por questões de organização e porque ele precisa de outros métodos, essa classe é o *Quicksort.cs*.

Logo no início do programa são declaradas as variáveis globais e é utilizado o método *Main*. O primeiro passo então é pedir para que o usuário entre com o valor para o tamanho do vetor que vai ser gerado.

Depois que é escolhido o tamanho do vetor, deve ser escolhido o tipo de vetor (suas características) que deve ser gerado. Pois o vetor pode ser totalmente aleatório, sem importar com as repetições dos elementos; ou pode ser aleatório, porém sem repetições de números dentro do vetor.

## 3 – Resultados:

Foram feitas algumas observações após vários testes dos resultados do console. Esses testes foram feitos alterando o valor da variável tamanhoVetor entre **10**, **100**, **1.000**, **10.000**, **100.000** e **1.000.000** de elementos armazenados dentro do vetor.

Essas análises foram feitas levando em consideração a **Quantidade de Comparações**, a **Quantidade de Atribuições** e o **Tempo de Execução** necessário para cada tipo de método, foram então geradas as tabelas abaixo com a média desses dados. Além disso, foi verificada também a diferença quando os vetores aleatórios são gerados **com** ou **sem** **repetição**.

## 3.1 – Quantidade de Comparações:

Com repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 18 | 28 | 13 | 15 |  |
| **100** | 2398 | 2472 | 310 | 468 |  |
| **1.000** | 253117 | 247787 | 5274 | 8599 |  |
| **10.000** | 24942105 | 25075253 | 77615 | 157440 |  |
| **100.000** | 2500448161 | 2503704227 | 978460 | 3142956 |  |
| **1.000.000** |  |  |  |  |  |

Sem repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 23 | 24 | 12 | 21 |  |
| **100** | 2076 | 2273 | 302 | 456 |  |
| **1.000** | 242934 | 248516 | 5289 | 8665 |  |
| **10.000** | 25100156 | 24532205 | 77241 | 167217 |  |
| **100.000** | 2499661328 | 2494853507 | 998242 | 3125144 |  |
| **1.000.000** |  |  |  |  |  |

## 3.2 – Quantidade de Atribuições:

Com repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 18 | 9 | 8 | 15 |  |
| **100** | 2398 | 99 | 93 | 342 |  |
| **1.000** | 253117 | 999 | 994 | 5457 |  |
| **10.000** | 24942105 | 9999 | 9991 | 75243 |  |
| **100.000** | 2500448161 | 99999 | 99988 | 967146 |  |
| **1.000.000** |  |  |  |  |  |

Sem repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 23 | 9 | 8 | 15 |  |
| **100** | 2076 | 99 | 93 | 342 |  |
| **1.000** | 242934 | 999 | 992 | 5457 |  |
| **10.000** | 25100156 | 9999 | 9993 | 75243 |  |
| **100.000** | 2499661328 | 99999 | 99980 | 967146 |  |
| **1.000.000** |  |  |  |  |  |

## 3.3 – Tempo de Execução:

Com repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 4 | 3 | 5 | 4 |  |
| **100** | 48 | 50 | 59 | 56 |  |
| **1.000** | 514 | 550 | 520 | 541 |  |
| **10.000** | 941 | 555 | 585 | 416 |  |
| **100.000** | 72366 | 15569 | 24959 | 3635 |  |
| **1.000.000** |  |  |  |  |  |

Sem repetição dos elementos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BubbleSort** | **InsertionSort** | **SelectionSort** | **ShellSort** | **QuickSort** |
| **10** | 5 | 5 | 4 | 4 |  |
| **100** | 57 | 52 | 49 | 48 |  |
| **1.000** | 532 | 521 | 506 | 502 |  |
| **10.000** | 927 | 451 | 536 | 465 |  |
| **100.000** | 74213 | 16711 | 23964 | 3676 |  |
| **1.000.000** |  |  |  |  |  |

## 4 – Conclusão:

Pode-se perceber que os algoritmos são bem parecidos para quantidades pequenas de elementos, quando passamos de 10.000 elementos, já podemos observar melhor a diferença entre eles, pois já começa a ter um crescimento exponencial dos valores analisados.

Quanto ao número de comparações, com ou sem repetição de elementos no vetor, os números são basicamente os mesmos.

Quanto ao número de atribuições, foi interessante notar que o *InsertionSort* segue um padrão de sempre aumentar um 9 em sua quantidade. Os outros métodos não tem muito um padrão, mas também aumentam de maneira gradativa de acordo com o tamanho do vetor.

Quanto ao tempo de execução, observa-se que quase sempre o método de *ShellSort* é o mais rápido. E também percebe-se que ao utilizar vetores sem repetição de elementos, o tempo de processamento também é um pouco mais demorado, tanto pra gerar o vetor, quanto para realizar as comparações.

Não foi possível fazer a análise para 1.000.000 (1 milhão) de elementos no vetor devido ao tempo excessivo de processamento. O algoritmo não chegava ao final de seu processamento, provavelmente estaria dando estouro de memória na máquina.