|  |
| --- |
| AMF_1  TAP-2016 |

DUM – Documento Único de Manutenção

1. Levantamento dos Requisitos
   * 1. Cenário Atual

|  |
| --- |
| **Cenário Atual – Definição do Problema** |
| Criar um sistema em que apresente a comparação entre variados algoritmos de ordenação. |

1. Detalhamento Funcional

|  |  |  |  |
| --- | --- | --- | --- |
| **ID Req.** | **ID**  **Func.** | **Funcionalidade** | **Descrição – Detalhamento Funcional** |
| 1 | 1.1 | Interface e definições do programa | **O programa foi desenvolvido em C# Windows Forms.**  **O Programa possui uma interface bem simples com um campo *textbox* para o usuário informa a quantidade de números a serem gerados, um *button* que executa a ação de gerar números aleatórios, um *listbox* que mostra os números após serem ordenados, uma *label*  que mostra o tempo demorado para fazer a ordenação e 10 *buttons* , onde cada um executa um método de ordenação diferente.**    **Além disso existe um *button* que testa todos os métodos e monta um gráfico entre eles, porém não está com o funcionamento correto ainda.** |
| 2 | 2.1 | Funcionamento do Programa  **Campo “Quantidade de Números”** | * **Ao Ser executado o usuário terá somente o campo “Quantidade de Números” disponível para executar uma informação.** * **Para liberar os outros botões no sistema, o usuário deve inserir no campo somente números inteiros maiores que zero.** * **Além disso, o campo só permite digitar números.** * **Se a tecla enter for pressionada ele chamará o método do botão “Gerar”;** |
| 2 | 2.2 | Funcionamento do Programa  **Botão “Gerar”** | * **O Botão “Gerar” irá gerar números aleatórios que começam em 0 e vai até o número digitado no campo “Quantidade de Números”, podendo se repetir.** |
| 2 | 2.3 | Funcionamento do Programa  **Botões de Ordenação** | * **Cada Botão na tela representa um método de ordenação.** * **Ao ser clicado deve executar o método de ordenação que representa.** * **O resultado da ordenação será exibido no campo “Resultado” junto com o tempo gasto pela ordenação.** * **Será Mostrado No Título do Programa o Método selecionado junto com seu tempo de execução.** |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 2.4 | Funcionamento do Programa  **Botões Testar Todos** | * Executa todos os métodos de ordenação e monta um gráfico deles. * Funcionamento ainda não está pronto, pois está acontecendo uma divergência de tempo entre alguns métodos de ordenação. |

1. Detalhamento Técnico
   1. Métodos de Ordenação Utilizados

**3.1.1 Insertion Sort**

Esse método de ordenação é um dos mais simples que existe. Ele ordena os elementos um por um comparando-os.

Suas principais características são:

* Bastante eficiente para ordenação com poucos elementos, porém muito ineficiente para listas com muitos elementos.
* Ele realiza a ordenação por inserção adaptativa, o que significa que terá um número de passos reduzidos à medida que tem os números da lista já estarem parcialmente ordenados.
* Sua complexidade é baixa.
* É estável, uma vez que não se altera a ordem relativa dos elementos com chaves iguais.

Análise de Complexidade:

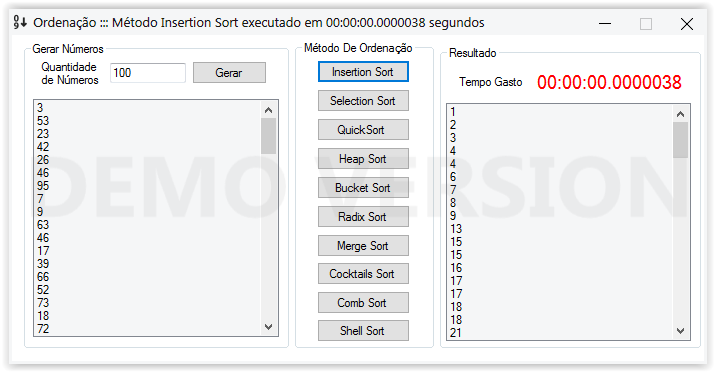
Pior Caso: O(n2)

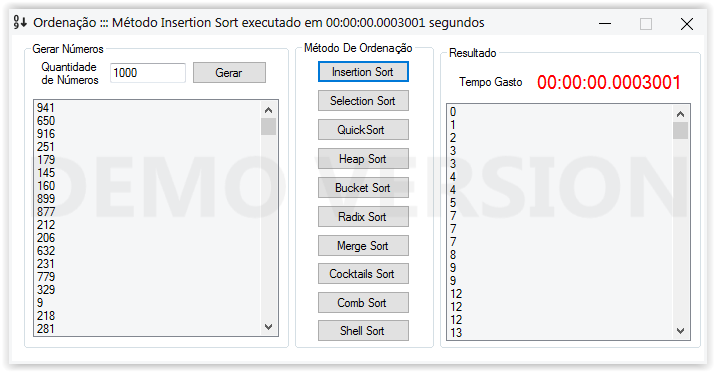
Melhor Caso: O(n)

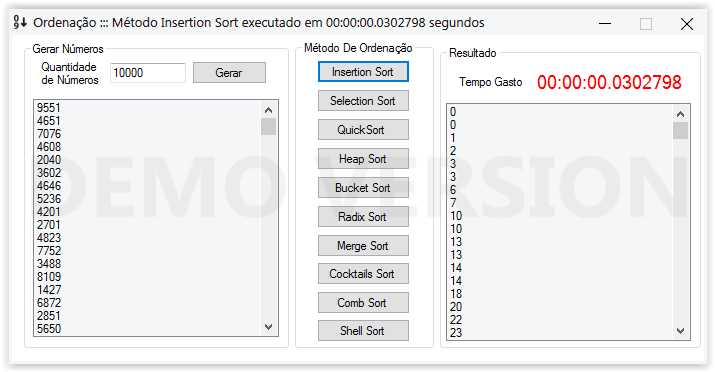
Caso Mediano: O(n2)

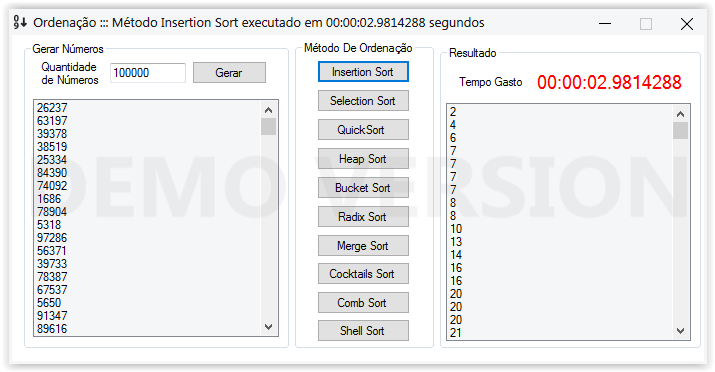
Complexidade de Espaço: O(1)

Comparação entre Listas com Tamanho Diferentes:









**3.1.2 Selection Sort**

Este algoritmo é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

Neste algoritmo de ordenação é escolhido um número a partir do primeiro, este número escolhido é comparado com os números a partir da sua direita, quando encontrado um número menor, o número escolhido ocupa a posição do menor número encontrado. Este número encontrado será o próximo número escolhido, caso não for encontrado nenhum número menor que este escolhido, ele é colocado na posição do primeiro número escolhido, e o próximo número à sua direita vai ser o escolhido para fazer as comparações. É repetido esse processo até que a lista esteja ordenada.

Análise de Complexidade:

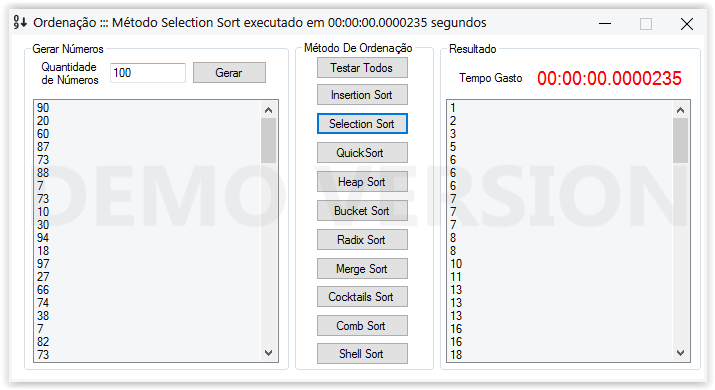
Pior Caso: O(n²)

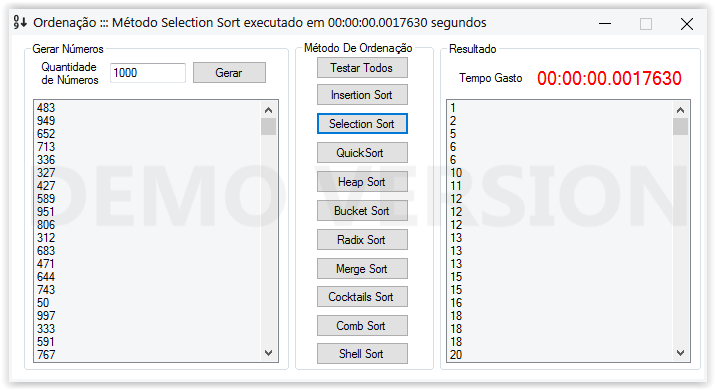
Melhor Caso: O(n²)

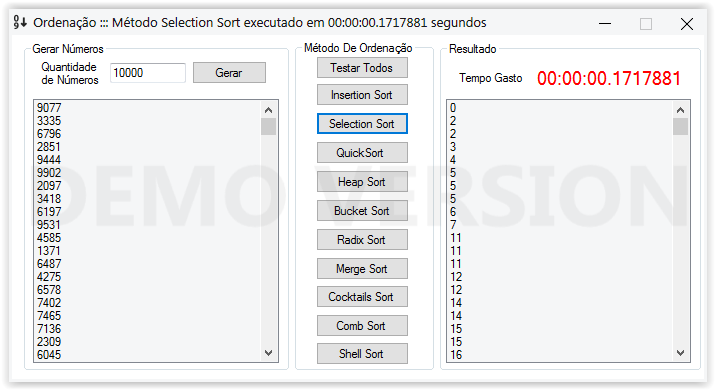
Caso Mediano: O(n²)

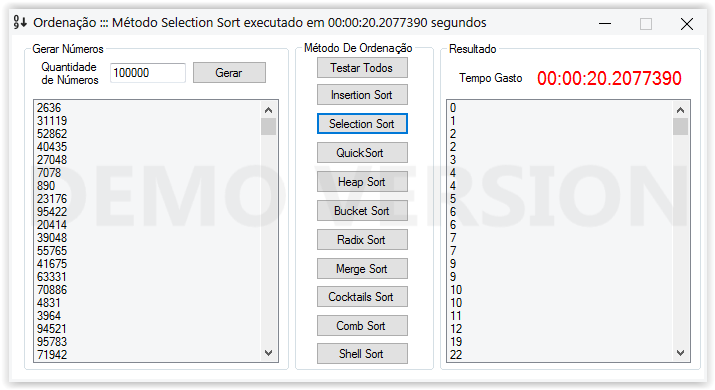
Complexidade de Espaço: O(1),O(n)

Comparação entre Listas com Tamanho Diferentes:









**3.1.3 Quick Sort**

O algoritmo Quicksort é um método de ordenação muito rápido e eficiente, inventado por C.A.R. Hoare em 1960, quando visitou a Universidade de Moscovo como estudante. Naquela época, Hoare trabalhou em um projeto de tradução de máquina para o National Physical Laboratory. Ele criou o Quicksort ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos mais fácil e rápido. Foi publicado em 1962 após uma série de refinamentos.

O Quicksort é um algoritmo de ordenação por comparação não-estável.

Esse método de ordenação utiliza a técnica divide and conquer (dividir o problema inicial em dois subproblemas e resolver um problema menor utilizando a recursividade) fazendo a divisão da lista em duas sub-listas, dependendo de um elemento chamado pivô, normalmente o 1º elemento da lista ou o último. Uma das sub-listas contém os elementos menores que o pivô enquanto a outra contém os maiores. O pivô é colocado entre ambas, ficando na posição correta. As duas sub-listas são ordenadas de forma idêntica, até que se chegue à tabela com um só elemento.

Análise de Complexidade:

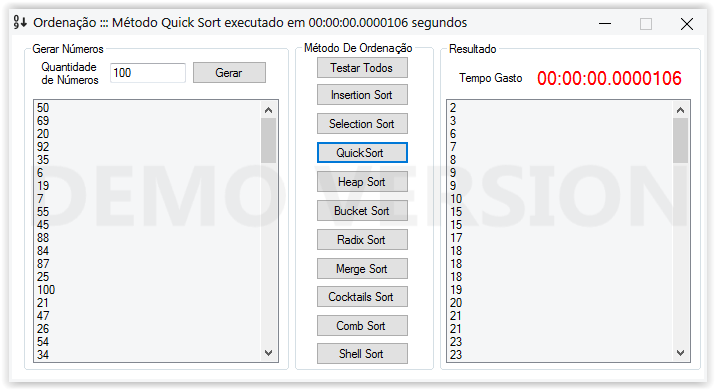
Pior Caso: O(n²)

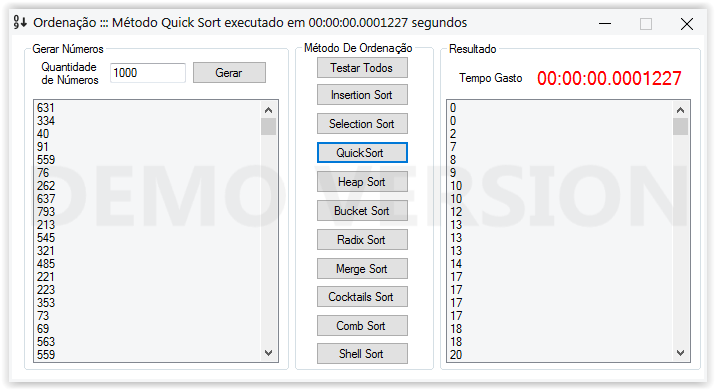
Melhor Caso: O(n *log* n)

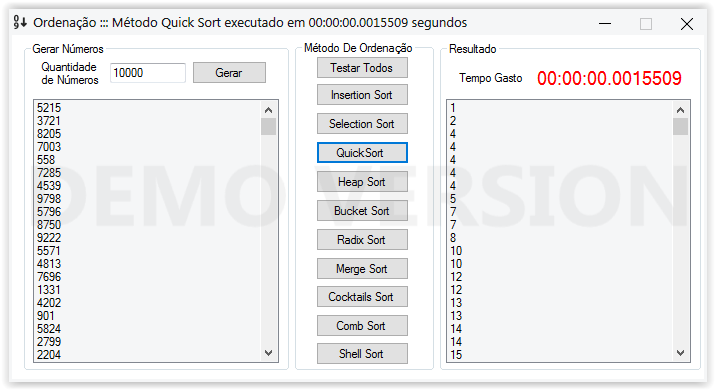
Caso Mediano: O(n *log* n)

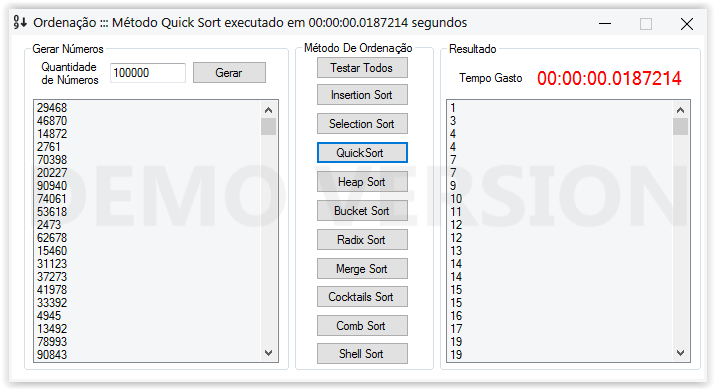
Complexidade de Espaço: O(n)

Comparação entre Listas com Tamanho Diferentes:









**3.1.4 Heap Sort**

O algoritmo heapsort é um algoritmo de ordenação generalista, e faz parte da família de algoritmos de ordenação por seleção. Foi desenvolvido em 1964 por Robert W. Floyd e J.W.J Williams.

O heapsort começa do final do array pesquisando os maiores elementos, enquanto o selectionsort começa do início do array pesquisando os menores.

Para ordenar, o heapsort usa um Heap. Heap é uma árvore binária com as seguintes propriedades:

O valor de cada nó não é menor que os valores armazenados em cada filho.

A árvore é perfeitamente balanceada e as folhas no último nível estão todas nas posições mais à esquerda.

Análise de Complexidade:

\Theta(n\log n)

Pior Caso:

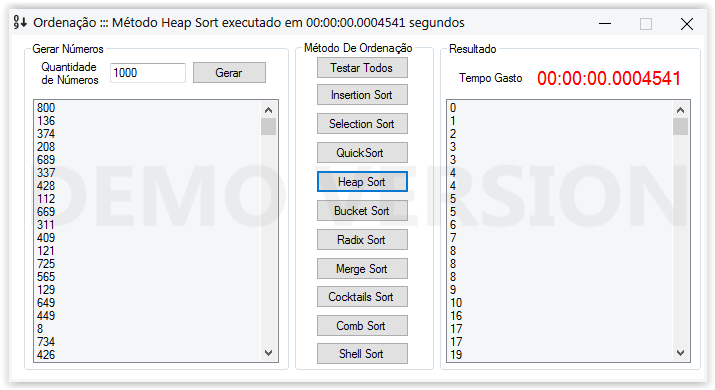
Melhor Caso: \Theta(n\log n)

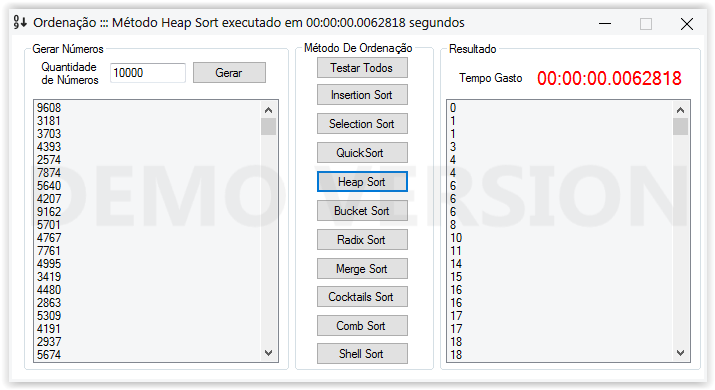
\Theta(n\log n)Caso Mediano:

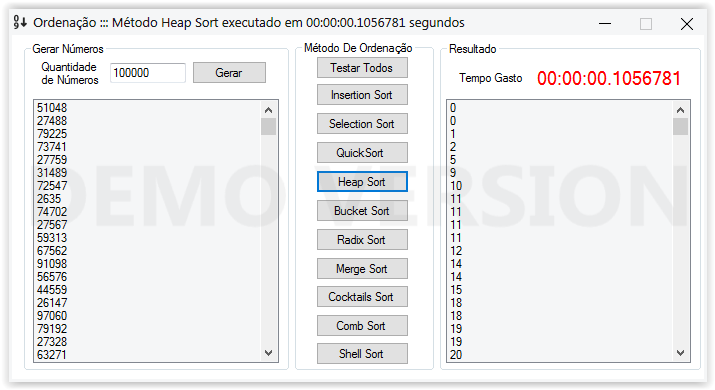
\Theta(n)Complexidade de Espaço: e \Theta(1)

Comparação entre Listas com Tamanho Diferentes:









**3.1.5 Shell Sort**

Criado por Donald Shell em 1959, publicado pela Universidade de Cincinnati, Shell Sort é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática. É um refinamento do método de inserção direta. O algoritmo difere do método de inserção direta pelo fato de no lugar de considerar o array a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles. Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em menores. Nos grupos menores é aplicado o método da ordenação por inserção. Implementações do algoritmo

Análise de Complexidade:

Pior Caso: O(n log ² n)

Melhor Caso: O(n)

Caso Mediano: Depende da Sequência da lista

Complexidade de Espaço: O(n)

Comparação entre Listas com Tamanho Diferentes:

