TDE 03 - Ordenação

Guilherme de Quadro Daudt e João Pedro Fonseca

1 Funcionamento Geral do Programa

Cada código realiza a ordenação de arrays com diferentes tamanhos e relata o tempo médio de execução, o número médio de trocas e iterações em várias rodadas de execução para calcular médias e apresentar os resultados. Os códigos utilizam uma semente fixa para garantir reprodutibilidade nos números aleatórios gerados. Os resultados são exibidos no final da execução de cada algoritmo para permitir comparações de desempenho entre eles.

2 Códigos

2.1 Shell Sort

O Shell Sort é um algoritmo de ordenação que pertence à classe dos algoritmos de ordenação por inserção. O algoritmo divide o array em várias sub-sequências e, em seguida, classifica essas sub-sequências usando o algoritmo de ordenação por inserção. Ele começa com um grande intervalo entre elementos a serem comparados e diminui esse intervalo em cada passo até que o array esteja totalmente ordenado. Durante o processo, o algoritmo faz comparações e trocas de elementos para posicionar cada elemento na sequência correta.

2.2 Insert Sort

O Insertion Sort é um algoritmo de ordenação que percorre o array da esquerda para a direita, construindo a parte ordenada do array à medida que avança. Ele insere cada elemento não ordenado na parte ordenada, movendo elementos maiores para a direita para fazer espaço para a inserção. É eficiente para arrays pequenos e quase ordenados, mas menos eficiente para arrays grandes devido ao seu desempenho no pior caso.

2.3 Merge Sort

O Merge Sort é um algoritmo de ordenação que segue a abordagem "dividir e conquistar". Ele divide o array em duas metades, recursivamente ordena essas metades e, em seguida, mescla as metades ordenadas para obter um array ordenado. O algoritmo é eficiente e estável, garantindo que a ordem relativa de elementos iguais seja preservada. É frequentemente usado para ordenar grandes conjuntos de dados devido ao seu desempenho consistente.

2.4 Bubble Sort

O Bubble Sort é um algoritmo de ordenação simples que percorre repetidamente o array, comparando pares de elementos adjacentes e trocando-os se estiverem fora de ordem. O algoritmo continua fazendo passagens pelo array até que nenhum elemento seja trocado durante uma passagem. Embora seja fácil de entender, o Bubble Sort não é eficiente para grandes conjuntos de dados e é geralmente usado apenas para fins educacionais.

2.5 Tempos e Comparações

Insert Sort

```
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro> &
nMessages' '-cp' 'C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Regist
Resultados do Insertion Sort:
Tamanho dos Dados: 50
Tempo médio de execução (nanos): 19860
Número médio de trocas e interações: 578
-------------
Resultados do Insertion Sort:
Tamanho dos Dados: 500
Tempo médio de execução (nanos): 912700
Número médio de trocas e interações: 61629
-------------
Resultados do Insertion Sort:
Tamanho dos Dados: 1000
Tempo médio de execução (nanos): 1038940
Número médio de trocas e interações: 251080
-------------
Resultados do Insertion Sort:
Tamanho dos Dados: 5000
Tempo médio de execução (nanos): 11862120
Número médio de trocas e interações: 6274193
-------------
Resultados do Insertion Sort:
Tamanho dos Dados: 10000
Tempo médio de execução (nanos): 27767320
Número médio de trocas e interações: 24903571
-------------
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro>
```

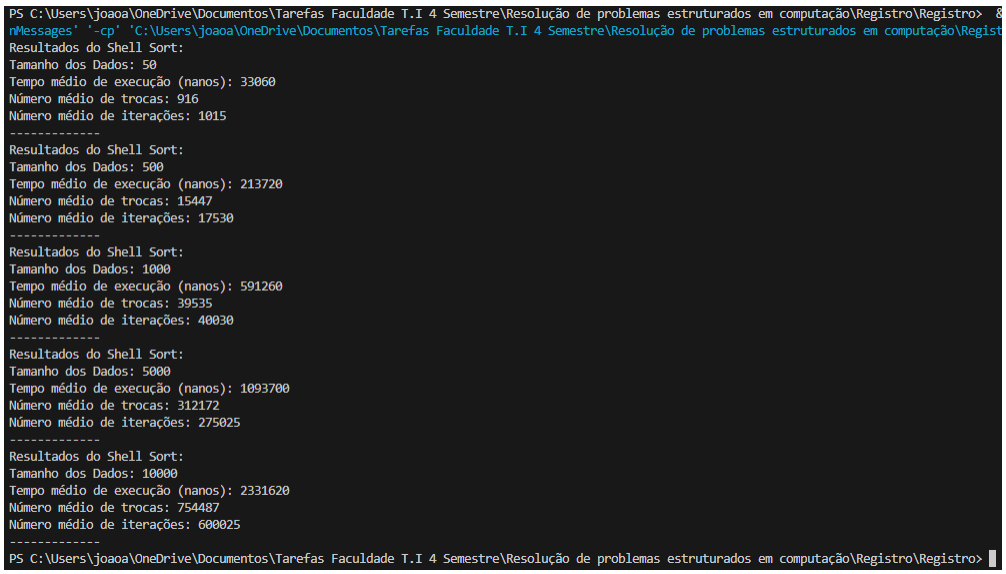
Merge Sort

```
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro> &
nMessages' '-cp' 'C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Regist
Resultados do Merge Sort:
Tamanho dos Dados: 50
Tempo médio de execução (nanos): 32120
Número de trocas: 739
Número de iterações: 1096
-------------
Resultados do Merge Sort:
Tamanho dos Dados: 500
Tempo médio de execução (nanos): 351500
Número de trocas: 11146
Número de iterações: 19292
-------------
Resultados do Merge Sort:
Tamanho dos Dados: 1000
Tempo médio de execução (nanos): 219960
Número de trocas: 25019
Número de iterações: 43627
-------------
Resultados do Merge Sort:
Tamanho dos Dados: 5000
Tempo médio de execução (nanos): 813840
Número de trocas: 154441
Número de iterações: 276359
-------------
Resultados do Merge Sort:
Tamanho dos Dados: 10000
Tempo médio de execução (nanos): 1911000
Número de trocas: 334519
Número de iterações: 602099
-------------
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro>
```

Bubble Sort

```
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro>
nMessages' '-cp' 'C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Reg
Tamanho dos Dados: 50
Tempo médio de execução (nanos): 49760
Número médio de trocas: 578
Número médio de iterações: 1225
-------------
Tamanho dos Dados: 500
Tempo médio de execução (nanos): 1662640
Número médio de trocas: 61629
Número médio de iterações: 124750
-------------
Tamanho dos Dados: 1000
Tempo médio de execução (nanos): 2574420
Número médio de trocas: 251080
Número médio de iterações: 499500
-------------
Tamanho dos Dados: 5000
Tempo médio de execução (nanos): 21934460
Número médio de trocas: 6274193
Número médio de iterações: 12497500
-------------
Tamanho dos Dados: 10000
Tempo médio de execução (nanos): 110872500
Número médio de trocas: 24903571
Número médio de iterações: 49995000
-------------
PS C:\Users\joaoa\OneDrive\Documentos\Tarefas Faculdade T.I 4 Semestre\Resolução de problemas estruturados em computação\Registro\Registro>
```

Shell Sort



2.6 Conclusão

Com base na análise dos resultados obtidos, podemos concluir que o algoritmo de ordenação Merge Sort demonstrou ser a escolha mais eficiente para uma ampla variedade de tamanhos de dados. Sua capacidade de manter um desempenho consistente, mesmo em conjuntos de dados maiores, o torna uma opção confiável para a tarefa de ordenação. O Merge Sort oferece estabilidade e previsibilidade, tornando-o uma excelente escolha para muitos cenários de ordenação de dados.