**Estrutura de Dados 2**

José Higor Ferreira de Oliveira Código: **20552** **Engenharia de Software**

Hyan Kelvin Código: **20478** **Engenharia de Software**

|  |
| --- |
| **Aleatório** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **100** | **1000** | **10.000** | **100.000** |
| **Inserção Direta** | 0.7000000041443855 | 21.399999997811392 | 333.4000000031665 | 32452.000000019325 |
| **Bubblesort** | 0.4999999946448952 | 19.20000000973232 | 292.5000000104774 | 34213.19999999832 |
| **Seleção Direta** | 0.8999999845400453 | 12.900000001536682 | 81.80000001448207 | 9330.900000000838 |
| **Quicksort** | 5.599999974947423 | 37.60000001057051 | 89.50000000186265 | 1151.1999999929685 |
| **HeapSort** | 3.899999981513247 | 57.40000001969747 | 174.59999999846332 | 1791.800000006333 |
| **Shelsort** | 0.29999998514540493 | 11.399999988498166 | 1.400000008288771 | 33.699999999953434 |

|  |
| --- |
| **Crescente** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **100** | **1000** | **10.000** | **100.000** |
| **Inserção Direta** | 0.10000000474974513 | 1.6999999934341758 | 50.99999997764826 | 6050.199999997858 |
| **Bubblesort** | 0.10000000474974513 | 1.400000008288771 | 51.500000001396984 | 6554.000000003725 |
| **Seleção Direta** | 1.0999999940395355 | 13.800000015180558 | 100.39999999571592 | 9761.499999993248 |
| **Quicksort** | 3.700000001117587 | 30.699999973876402 | 71.10000000102445 | 813.4000000136439 |
| **HeapSort** | 4.6999999904073775 | 34.59999998449348 | 219.50000000651926 | 1622.1999999834225 |
| **Shelsort** | 0.1999999803956598 | 11.799999978393316 | 1.3000000035390258 | 17.700000025797635 |

|  |
| --- |
| **Decrescente** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **100** | **1000** | **10.000** | **100.000** |
| **Inserção Direta** | 0.8000000088941306 | 27.10000000661239 | 630.9999999939464 | 60470.19999998156 |
| **BubbleSort** | 0.1999999803956598 | 20.699999993667006 | 545.0999999884516 | 58399.00000000489 |
| **Seleção Direta** | 0.10000000474974513 | 3.1999999773688614 | 105.60000000987202 | 12107.9000000027 |
| **QuickSort** | 2.800000016577542 | 22.700000001350418 | 64.29999999818392 | 813.6999999987893 |
| **HeapSort** | 3.599999996367842 | 35.299999988637865 | 133.00000000162981 | 1408.499999990454 |
| **ShellSort** | 0.20000000949949026 | 12.100000021746382 | 1.3000000035390258 | 17.100000026402995 |

**Comparação dos Métodos**

**Vantagens e Desvantagens**

**ShellSort**

**Vantagens**

1. Shell é uma ótima opção para arquivos de tamanhos moderado
2. Sua implementação é simples e uma quantidade de códigos menores
3. Rápido e é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática
4. É o método que se adequa as todas as situações

**Desvantagens**

1. O método não é estável, um método simples
2. Tempo de execução sensível á ordem inicial do arquivo

**QuickSort**

**Vantagens**

1. É bastante eficiente para ordenar dados
2. Para dados bem bagunçados esse método é o mais vantajoso por ser mais econômico
3. Funciona muito bem com listas grandes
4. Necessita apenas de uma pequena pilha como memoria extra

**Desvantagens**

1. Sua implementação é delicada e difícil
2. O método não é estável

**HeapSort**

**Vantagens**

1. E bom para arquivos com grandes registros
2. Não necessita de nenhuma memoria adicional
3. Para dados imprevisíveis, pode ser o mais vantajoso por ser previsível em termos de execução

**Desvantagens**

1. O método não é estável
2. Construir a árvore-heap pode consumir muita memoria
3. Não é recomendado para vetores de entrada com poucos elementos, devido a complexidade do heap

**Bubble Sort**

**Vantagens**

1. O algoritmo simples e facilita o aprendizado
2. Algoritmo estável
3. Fácil implementação
4. Os elementos são trocados de lugar sem utilizar armazenamento temporário, faz o requerimento de espaço ser mínimo

**Desvantagens**

1. O método é lento
2. Não apresenta bom resultados quando a lista contem muitos itens
3. Menos eficiente

**Inserção Direta**

**Vantagens**

1. O número mínimo de comparações e movimentos ocorre quando os itens já estão originalmente ordenados
2. Fácil implementação
3. O vetor já ordenado favorece a ordenação
4. O algoritmo é estável

**Desvantagens**

1. Número grande de movimentações
2. Ineficiente quando o vetor está ordenado inversamente
3. Ocorre o deslocamento cada vez em que o elemento é colocado na sua posição correta, onde o numero de deslocamento pode ser significativo

**Seleção Direta**

**Vantagens**

1. O algoritmo de fácil implementação
2. Pequenos números de movimentações, interessantes para arquivos pequenos
3. É o algoritmo utilizado para os arquivos com registros muito grandes, pode ser também para arquivos pequenos

**Desvantagens**

1. O fato de o conjunto já estar ordenado não ajuda em nada, pois o número de comparações continua o mesmo
2. O algoritmo não é estável, isto é, os registros com chaves iguais nem sempre irão manter a mesma posição relativa de antes do início da ordenação

**Complexidade**

**1. BubbleSort**

A complexidade deste algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados

**2. SeleçãoDireta**

A seleção direta compara a cada interação um elemento com os outros, visando encontrar o menor. Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno. A complexidade deste algoritmo será sempre 0(N)² enquanto que, por exemplo, os algoritmos heapsort e mergesort possuem complexidades.

**3. ShellSort**

A ordenação ShellShort utiliza a quebra sucessiva da sequência a ser ordenada e implementa a ordenação por inserção na sequência obtida. Devido a sua complexidade possui excelentes desempenhos em N muito grandes, inclusive sendo melhor que o Merge Sort.

**4. QuickSort**

A complexidade do QuickSort é o desempenho ser dependente do particionamento ser balanceado ou não balanceado.

Particionamento

Exemplo: A = [0 1 3 4 5 7 8]

Critério de escolha do pivo: primeiro elemento

**5.HeapSort**

A complexidade do heapSort é devido à sua equivalência em inserir cada elemento, um de cada vez, em um heap vazio. Isso pode parecer contra-intuitivo, pois, de relance, é aparente que o primeiro faz apenas metade das chamadas para sua função de peneiramento logarítmico do tempo como o segundo. Eles parecem diferir apenas por um fator constante, que nunca afeta a análise assintótica.

**6. Inserção Direta**

Quando cada elemento na entrada não está mais do que lugares longe de sua posição classificada estável. isto é, não altera a ordem relativa de elementos com chaves iguais.

**Pior e Melhor Caso**

**1**. **BubbleSort**

Apresentou-se como pior método de ordenação.

**2.** **Inserção Direta**

Em arranjos já ordenados demonstrou um comportamento linear, sendo útil para ordenação de arquivo com poucos elementos fora da ordem.

**3.** **Seleção Direta**

Apresentou uma vantagem apenas no número de movimentações, porém seu tempo de execução é pior que o Inserção Direta. A sua utilização deve ser feita quando os registros dos arquivos forem grandes, em virtude da quantidade de movimentos que é realizada para a operação.

**4.** **ShellSort**

Apresentou-se eficiente em todos os testes. Em geral é um método que se adéqua a todas as situações. A sua relação com o QuickSort cresce a medida que o número de elementos aumenta. Em todas as situações manteve-se próximo do algoritmo mais rápido do teste.

**5**. **QuickSort**

Obteve o melhor resultado na maioria dos testes. Sua eficiência foi identificada apenas nos grandes arquivos.

**6. HeapSort**

Quando é a pergunta é “qual o maior elemento de vetor?” o resultado é obtido instantaneamente, quando o valor da posição do vetor é alterado o heap consegue se restabelecer muito rapidamente.