**LABORATÓRIO DE PROGRAMAÇÃO AVANÇADA**

**NONO TRABALHO PRÁTICO**

**COMPARAÇÃO DE MÉTODOS DE ORDENAÇÃO**

**Ludymila Lobo – 21101577**

**OBSERVAÇÕES**

*A comparação será feita através dos seguintes parâmetros:*

*(a) número de comparações*

*(b) número de trocas*

*(c) tempo de execução*

*Os seguintes métodos de ordenação devem ser implementados:*

* *Bolha*
* *Inserção*
* *Seleção*
* *Mergesort*
* *Quicksort.*
* *Heapsort*

*As comparações devem ser feitas em três cenários distintos, isto é:*

* *ordem crescente*
* *ordem decrescente*
* *aleatórios*

*Para o caso de vetores ALEATÓRIOS, as comparações devem ser feitas usando o VALOR MÉDIO obtido na aplicação do método por 10 (DEZ) vezes. Em cada uma, deve-se gerar novos números aleatórios. Para o caso de vetores ORDENADOS, as métricas correspondem à única execução realizada. Em todos os casos, deve-se usar vetores de tamanhos 1000, 5000, 10000, 20000 e 50000.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Aleatórios** | | | | | **Ordem Crescente** | | | | | | **Ordem Decrescente** | | | | | |
| Métrica | n | 1K | 5K | 10K | 20K | 50K | 1K | 5K | 10K | 20K | 50K | 1K | | 5K | 10K | 20K | 50K |
| Nº de compara-  ções | Bolha | 499500 | 12497500 | 49995000 | 199990000 | 18446744073671036428 | 499500 | 12497500 | 49995000 | 199990000 | 1249975000 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Seleção | 499500 | 12497500 | 49995000 | 199990000 | 18446744073671036428 | 499500 | 12497500 | 49995000 | 199990000 | 1249975000 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Inserção | 249695 | 6174837 | 24848641 | 99304721 | 189079693 | 0 | 0 | 0 | 0 | 0 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Mergesort | 8701 | 55152 | 120189 | 260153 | 716460 | 4932 | 29804 | 64608 | 139216 | 382512 | 5044 | | 32004 | 69008 | 148016 | 401952 |
| Quicksort | 4943 | 18519 | 37726 | 87448 | 218196 | 0 | 0 | 0 | 0 | 0 | 250000 | | 6250000 | 25000000 | 100000000 | 625000000 |
| HeapSort | 18648 | 116640 | 252482 | 544264 | 1492998 | 0 | 0 | 0 | 0 | 0 | 250000 | | 6250000 | 25000000 | 100000000 | 625000000 |
| Nº de trocas | Bolha | 251049 | 6204110 | 24884007 | 98903778 | 188747620 | 0 | 0 | 0 | 0 | 0 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Seleção | 988 | 4936 | 9886 | 19793 | 49493 | 0 | 0 | 0 | 0 | 0 | 500 | | 2500 | 5000 | 10000 | 25000 |
| Inserção | 249695 | 6174837 | 24848641 | 99304721 | 189079693 | 0 | 0 | 0 | 0 | 0 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Mergesort | 9976 | 61808 | 133616 | 287232 | 784464 | 9976 | 61808 | 133616 | 287232 | 784464 | 9976 | | 61808 | 133616 | 287232 | 784464 |
| Quicksort | 12946 | 163655 | 577380 | 2178552 | 12921177 | 499500 | 12497500 | 49995000 | 199990000 | 1249975000 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| HeapSort | 9889 | 61285 | 132089 | 283996 | 776283 | 499500 | 12497500 | 49995000 | 199990000 | 1249975000 | 499500 | | 12497500 | 49995000 | 199990000 | 1249975000 |
| Tempo  (Microssegundos) | Bolha | 3775.000 | 84589.000 | 373094.000 | 1600615.000 | 10763802.000 | 1500.000 | 39745.000 | 151891.000 | 621550.000 | 3967187.000 | 4195.000 | | 100123.000 | 398483.000 | 1601159.000 | 10312200.000 |
| Seleção | 1463.000 | 34901.000 | 140618.000 | 576360.000 | 3567231.000 | 1343.000 | 36304.000 | 148610.000 | 568808.000 | 3516063.000 | 1563.00 | | 39026.000 | 151784.000 | 605643.000 | 4016928.000 |
| Inserção | 1563.000 | 41592.000 | 156821.000 | 573040.000 | 3621118.000 | 3.000 | 17.000 | 30.000 | 70.0000 | 166.000 | 2823.000 | | 77819.000 | 354018.000 | 1180464.000 | 7886805.000 |
| Mergesort | 149.000 | 846.000 | 1796.000 | 3687.000 | 9605.000 | 112.000 | 653.000 | 1260.000 | 2468.000 | 6658.000 | 116.000 | | 766.000 | 1200.000 | 2462.000 | 6861.000 |
| Quicksort | 103.000 | 838.000 | 2348.000 | 7716.000 | 40368.000 | 1431.000 | 33924.000 | 134612.000 | 646705.000 | 3598312.000 | 2666.000 | | 65483.000 | 309943.000 | 1174959.000 | 7044034.000 |
| Heapsort | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 6033.00 | 169668.00 | 636124.0 | 2412033.0 | 14666174.0 | 17753 | | 326991.00 | 1302192.00 | 5016709.00 | 31613795.0 |

**TABELA DE DADOS DE EXECUÇÃO**

**QUESTÕES**

1. **Para cada método de ordenação, qual foi seu melhor e pior caso observado?**
   * **Bubble Sort**
     + Melhor caso: vetor em ordem crescente, pois não ocorre nenhuma troca.
     + Pior caso: vetor em ordem decrescente.
   * **Insertion Sort** 
     + Melhor caso: quando o vetor está ordenado.
     + Pior caso: vetor em ordem decrescente.
   * **Selection Sort**
     + Melhor caso: quando o vetor está ordenado.
     + Pior caso: vetor em ordem decrescente.
   * **Merge Sort**
     + Melhor caso: quando o vetor está ordenado, devido a menor quantidade de comparações realizadas.
     + Pior caso: vetor desordenado (de forma aleatória), cenário no qual foi obtido o maior número de comparações e maior tempo de execução.
   * **Quick Sort**
     + Melhor caso: vetor desordenado (ordenado de forma aleatória), com tempo de execução e numero de trocas menor do que um vetor em ordem crescente ou decrescente
     + Pior caso: em ordem decrescente, situação na qual os três parâmetros foram consideravelmente pior em comparação aos outros cenários. Porem, em ordem crescente apresentou tempo de execução e numero de trocas pior que quando o vetor estava em ordem aleatória.
   * **HeapSort**
     + Melhor caso: vetor em ordem aleatória, mantendo um tempo de execução extremamente rápido em todos os tamanhos de conjuntos aleatórios testados.
     + Pior caso: ordem decrescente, no qual obtém as piores métricas de tempo de execução, numero de comparações e de trocas.
2. **Qual função melhor descreve o desempenho de cada método (, ,, )?**

Para definir as funções de cada método, iremos usar o pior caso como parâmetro.

|  |  |
| --- | --- |
| Ordenação | Função Representativa |
| Bubble Sort |  |
| Selection Sort |  |
| Insertion Sort |  |
| Merge Sort |  |
| Quick Sort |  |
| HeapSort |  |

1. **Em quais métodos (e quantidade de dados nos vetores), o número de comparações é um bom substituto para o tempo de execução (ou seja, as duas métricas dão resultados relativos parecidos)?**

Bubble Sort, Selection Sort e Insertion Sort.

1. **Qual o melhor algoritmo dentre os testados?**

Depende do cenário. Para um vetor em ordem crescente, seria o Insertion Sort. Caso o objetivo seja reduzir acessos a memória, seria o MergeSort.

Levando em consideração que as situações nas quais há a necessidade de ordenação de dados, geralmente estes são obtidos e armazenados de forma completamente desordenada e aleatória (sem estar já ordenado crescentemente e sem estar na ordem inversa) seria recomendado o uso do HeapSort de maneira geral, em termos da rapidez proporcionada pelo algoritmo, e de sua performance com vetores ordenados de maneira aleatória.

**5.Gráficos**

Abaixo, os gráficos correspondentes aos métodos estudados e cada uma das variáveis analisadas. Os métodos de ordenação que não estão aparecendo no gráfico contém todos os 5 valores (em 1k, 5k, 10k, 20k e 50k) sendo 0. Portanto, não são representáveis em escala logarítmica, mas podemos imaginalos como uma linha reta no eixo horizontal correspondente ao valor 0.