**UNIVERSIDADE FEDERAL DE MATO GROSSO**

**CAMPUS UNIVERSITÁRIO DO ARAGUAIA**

**Instituto de Ciências Exatas e da Terra**

**Curso de Bacharelado em Ciência da Computação**

Disciplina: Projeto e Análise de Algoritmos

Professor: Robson Silva Lopes

***Guilber Domingues Leal***

**Trabalho 1 – Algoritmos de ordenação**

**Tabela: Tempo gasto em segundos na execução dos Algoritmos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Bubble** | **Quick** | **Count** | **Quick Random** | **Insertion** | **Merge** |
| **1- Arquivo**  **1.000/100.000** | **0.001769** | **0.000088** |  | **0.0147449** | **0.000637** | **0.000162** |
| **2- Arquivo**  **10.000/100.000** | **0.273082** | **0.001156** |  | **1.397594** | **0.067464** | **0.007950** |
| **3- Arquivo**  **100.000/100.000** |  |  |  |  |  |  |
| **4- Arquivo**  **100.000/10.000** |  |  |  |  |  |  |
| **5- Arquivo**  **1.000.000/50.000** |  |  |  |  |  |  |
| **6-Arquivo**  **1.000.000/500.000** |  |  |  |  |  |  |

Observe como é possível visualizar o crescimento exponencial do tempo de execução de cada ordenador de acordo que o valor da instância cresce.

**Comparação- 1**

**Quick com Pivo vs Quick aleatório**

O Quick com pivô fixo e o Quick com pivô randômico demandam tempos muito parecidos para a ordenação de grandes valores e se destacam entre todos os algoritmos. Curiosamente, ele pode ser pior ou igual a outros ordenadores no pior caso, no entanto, o pior caso é muito difícil de se ocorre e vai depender de como ocorre o particionamento.

Com base nisso, podemos afirmar que o pivô fixo e o aleatório tem complexidade assintótica iguais, por ser improvável a ocorrência deste particionamento ruim e por ambos garantirem uma divisão com proporcionalidade constante.

**Comparação- 2**

**insertion vs Bubble**

O Bubble compara um valor com todos os outros N vezes garantindo que o um valor por vez esteja na posição correta, já o Insertion constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação. O insertion Sort em comparação com o Bubble Sort teve uma considerável diferença no tempo de execução, o que leva a concluir que, o Insertion Sort é mais eficiente que ele.

**Comparação- 3**

**Merge vs Count**

O Merge aborda o método de divisão e conquista, particionando a instância até chegar no objeto elementar, no caso do vetor, apenas um valor, após a divisão o algoritmo volta comparando e reconstruindo o vetor ordenadamente. Com está abordagem acelera o processo de conquista, porém, aumenta o uso de memória na pilha de execução do algoritmo e na necessidade de um vetor auxiliar. Já a ideia básica do counting sort é determinar, para cada entrada x, o número de elementos menor que x. Essa informação pode ser usada para colocar o elemento x diretamente em sua posição no array de saída. Por exemplo, se há 17 elementos menores que x, então x pertence a posição 18. Esse esquema deve ser ligeiramente modificado quando houver vários elementos com o mesmo valor, uma vez que nós não queremos que sejam colocados na mesma posição. A classificação por contagem é eficiente se o intervalo de dados de entrada não for significativamente maior que o número de objetos a serem classificados e o Count não é um algoritmo estável. Com tudo o Count sort pode ser mais rápido pois tem sua complexidade O(n) já o merge tem complexidade O(n log n).

**considerações finais**

De acordo com os testes dos algoritmos de ordenação, o algoritmo que foi mais eficiente é o Count, em todos os conjuntos de números ele obteve uma média mais rápida que os outros algoritmos, seguido pelo Quick que nos arquivos testados foi o melhor ordenador com complexidade O(n log n), por isso obteve uma média pouco abaixo do Count. O algoritmo que obteve o maior tempo de execução foi o Bubble, em todos os arquivos testados ele ficou com uma média de tempo de execução muito acima do que os outros algoritmos, principalmente à medida que os valores gerados cresceram.

Complexidade assintótica no caso médio:

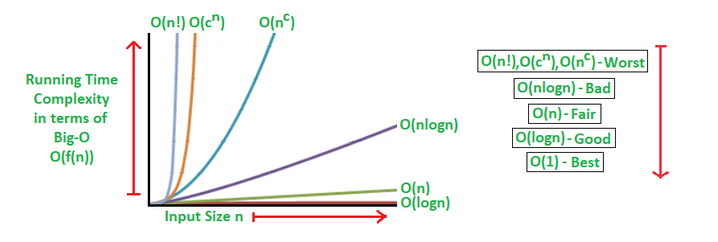
**Bubble sort: O(n²)**

**Quick sort: O(n log n)**

**Quick sort (random): O(n log n)**

**Merge: Θ(n log n)**

**Count sort: O(n)**

**Insertion sort: O(n²)**