#README

Algoritmo em C focado em analisar as diversas estruturas de dados a nível de complexidade e eficiente.

A partir da análise dos resultados obtidos pela execução dos métodos ordenação, podemos chegar a algumas conclusões em relação ao funcionamento e eficácia de cada um, como pode ser visto na tabela abaixo:

Métodos de Ordenação	1000		10000		100000	
	Trocas	tempo(s)	Trocas	tempo(s)	Trocas	tempo(s)
Bubble	251,814	0.000	25,056,772	0.276	4874549565	29.714
Selection	1000	0.000	1000	0.140	100000	11.56
Insertion	999	0.000	9999	0.094	99999	8.082
Shell	8006	0.000	12005	0.000	1500006	0.0300
Merge	4499	0.000	304141	0.000	773391	0.0240
Quick	5288	0.000	76362	0.000	1035793	0.014

Bubble Sort: Apesar de ser o mais simples de ser executado, se apresenta como o menos eficiente.No Bubble Sort temos a análise de um elemento da posição “i”, que será comparado com o elemento da posição i + 1. Isso torna a execução do programa lento, devido ao fato de que o vetor deverá ser percorrido o quanto for necessário, sendo ineficiente para casos de listas muito grandes. Com isso podemos observar o progressivo aumento de tempo de execução que tivemos em nossa execução, à medida que o número de entradas aumentava. Levando um tempo de execução extremamente alto em comparação aos outros métodos, tendo assim, o bubble como o método menos eficiente.

Selection Sort: Temos a utilização de uma comparação feita entre os valores obtidos, onde o menor número sempre será buscado e assim trocado com o elemento da primeira posição até que a lista seja ordenada. Apesar de ser mais eficiente do que o Bubble, continua lento, para listas longas.

Insertion Sort: Nesse caso, a lista será percorrida da esquerda para a direita, de forma que ao avançar, deixa os elementos mais à esquerda ordenados. Dessa forma, temos o insertion como um algoritmo eficiente para pequenas listas.

Shell Sort: Podemos caracterizar o Shell Sort como um método de ordenação de dividir para conquistar onde se faz trocas a uma certa

distância que diminui a cada repetição. Comparando primeiramente elementos separados por “salto” posições e o rearranja e vai diminuindo a distância de comparação até que salto = 1 correspondendo assim, ao algoritmo de inserção. Dessa forma tendo uma eficiência significativa até em um número de listas maiores.

Merge Sort: O merge short também se caracteriza como um algoritmo de dividir e conquistar criando uma sequência ordenada a partir de duas outras também ordenadas. Porém tem uma desvantagem, vista pelo fato de requerer o dobro de memória, precisando de um vetor com a mesma dimensão do vetor que está sendo classificado. Apesar disso, tem um tempo de execução relativamente curto.

Quick Sort: No Quick tiramos um elemento como pivô, e organizamos os números entre ele (menores e maiores), no final fazendo os dois grupos se ordenarem recursivamente. Tornando o quick sort o algoritmo mais eficiente e que demandou menos tempo de execução mesmo com listas maiores.