**OBJETIVO DO TRABALHO**

O objetivo do trabalho tende a desenvolver dois programas estruturados com algoritmos de ordenação de dados onde à unidade de medida para efeito de comparação entre os dois algoritmos foi o tempo de ordenação entre eles.

O tempo de aquisição não foi contabilizado, e sim apenas o processo específico da ordenação.

**INTRODUÇÃO**

Algoritmo de ordenação, em ciência da computação, é um algoritmo que coloca os elementos de uma dada sequência em certa ordem. Em outras palavras efetua sua ordenação completa ou parcial. O objetivo da ordenação é facilitar a recuperação dos dados de uma lista.

Sabemos que as principais linguagens de programação atuais (para não afirmar todas) possuem alguma função para essa tarefa, ou seja, com apenas uma chamada de função (sort (), qsort (), qsort (), etc…) conseguimos ordenar determinada estrutura de dados. Mas muitas vezes precisamos fazer alguma coisa enquanto estamos ordenando esses dados, seja para adicionar mais coisas aos dados, seja para guardar determinadas informações desse processo. Para isso temos que programarmos nós mesmos essa função. Abaixo vou explicar um pouco sobre os principais algoritmos existentes:

**Algoritmo de ordenação por inserção**

Esse algoritmo, como muitos outros, é baseado em ações que nós (como pessoas) fazemos no dia-a-dia para resolver o problema. Lembra quando você está jogando baralho, e suas cartas já estão na mesa e você precisa colocá-las na mão de uma forma ordenada? Essa ordenação deve ser feita de uma maneira que você esteja acostumado a escolher uma carta facilmente para jogar mais rápido e melhor… É exatamente isso que o algoritmo de ordenação por inserção faz por baixo dos panos. A idéia principal dele é: adiciona-se um item qualquer à estrutura de dados, depois, para cada item que ainda não esteja na estrutura, antes de adicioná-la, comparar com cada item que já está nela (consequentemente já ordenada) até encontrar a posição a ser encaixada. É exatamente o que fazemos com o baralho. Essa opção é boa quando temos uma entrada pequena de dados, para entradas grandes pode se consumir muito tempo de processamento.

**Algoritmo Bubblesort**

Talvez um dos mais populares dos algoritmos para ordenação seja o bubblesort, isso pela fácil memorização de como funciona e como é fácil a sua implementação. Ele consiste basicamente em intercalar elementos, por isso se enquadra na categoria de ordenação por intercalação, a implementação dele é simples. Com uma estrutura de dados desordenada inicia-se o algoritmo pelo primeiro elemento, depois faz-se a comparação dele com todos os que estão depois dele na estrutura desordenada, portanto com 4 linhas de código dá para se implementar.

**Algoritmo Mergesort**

Um algoritmo bem interessante por alguns motivos. Esse é um algoritmo que segue uma técnica de dividir para conquistar na base de sua idéia principal. O raciocínio básico para que ele funcione é o seguinte: ordenar uma estrutura significa ordenar várias subestruturas internas já ordenadas, caso essas estruturas não estejam ordenadas, basta ordená-las pelo mesmo método (ordenar suas subestruturas internas… até o infinito), é um algoritmo com uma base bem matemática e com um método bem interessante para estudo. Como a idéia pode não ter ficado bem clara vou mostrar um exemplo: ordenar 6, 3, 4, 8, 1, 2, 3, 5… o algoritmo irá dividir em pares ordenados: {3,6}, {4,8}, {1,2}, {3,5}, depois ir fazendo o merge desses dados, ou seja, juntando-os em dois pares ordenados: {3,4,6,8}, {1,2,3,5}, depois juntar novamente: 1,2,3,3,4,5,6,8. No final do algoritmo a seqüência inicial está ordenada a partir de divisões… Por sua base bem estruturada esse algoritmo tem tempo médio bem rápido.

**Algoritmo Heapsort**

Esse algoritmo tem tempo de execução de um algoritmo de ordenação por intercalação, e faz as suas operações localmente, sempre apenas um número constante de elementos é armazenados fora da estrutura de dados, como é feito o algoritmo por inserção. Mas o que ele tem de interessante é a forma que ele arranja os dados para depois ordená-los, ele utiliza uma estrutura chamada de heap ou monte, que é uma arvore binária que é extremamente importante para vários conceitos e problemas computacionais, pois depois de ordenados os dados podem, por exemplo, saber em que nível da árvore se encontra determinado item, operações sobre arvores binárias são conceitos muito importantes para outras estruturas como grafos.