**Trabalho 1 – Classificação e Pesquisa de Dados**

**Dupla:** Rhuan Lucas Barbosa Fernandes e Davi Santos Ferrarez

1. **Introdução:**

**Bubble sort:**

Sua complexidade é O(n²), por este motivo o bubble sort não é recomendado para uma entrada muito grande de dados.

Este algoritmo de ordenação consiste em percorrer todo o vetor de elementos comparando o valor da posição atual com seu próximo, caso a condição seja verdadeira, os dois são trocados de posição. O vetor é percorrido em loop até que todos os elementos estejam ordenados da forma correta.

**Insertion sort:**

O insertion sort possui complexidade O(n²) e é o mais eficiente entre os algoritmos para solução de pequenas entradas.

Sua ordenação é realizada de forma que o elemento atual seja comparado com seus antecessores, um por vez, até que a condição não seja mais satisfeita, e após isto seja posicionado no seu devido local. Para ordenação crescente ele será posicionado a frente do primeiro elemento menor que ele, já para decrescente, será posicionado a frente do primeiro maior.

**Merge sort:**

Sua complexidade é O(n log n), a mesma em comparação a outros algoritmos de divisão e conquista, mas é muito mais eficiente para entradas muito grandes em relação aos algoritmos de comparação e troca.

Seu funcionamento se resume em dividir o vetor ao meio gerando dois subvetores e, recursivamente, fazer o mesmo nestes subvetores. Após toda divisão, é feita, também recursivamente, a comparação e junção dos subvetores em vetores maiores até se tornar apenas um vetor com todos os elementos ordenados.

1. **Tabela:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vetores | Bubble  sort | Selection  sort | Insertion  sort | Quick  sort | Merge  sort | Heap  sort |
| Aleatório | 31.532 s | 13.053 s | 6.756 s | 0.014 s | 0.037 s |  |
| Crescente | 12.44 s | 12.85 s | 0.001 s |  | 0.039 s |  |
| Decrescente | 23.969 s | 23.064 s | 0.001 s |  | 0.052 s |  |