Nomes: Jonas Antonio Gomes Vicente e Juan Ferreira Carlos

**Trabalho de Ordenação Interna**

**1 - Descrição:**

**1.1 - Seleção:** é baseado em se passar sempre o menor ou o maior valor de um vetor para a primeira posição, depois o de segundo valor de acordo com a ordem requerida para a segunda posição, e assim é feito sucessivamente até os últimos dois elementos, possui uma complexidade de O(n²);

**1.2 -** **Shell Sort:** é um método criado a partir do método de ordenação por inserção, o qual é necessário realizar a troca de itens adjacentes para que assim possa ser definido o ponto de inserção. No Shell Sort, esse problema é solucionado, pois nele ocorrem trocas de registros distantes um do outro. Dentre suas vantagens, a principal é que o Shell Sort é uma ótima opção para a ordenação de arquivos pequenos. No melhor e nos médios casos, o algoritmo possui complexidade O(n), e no pior dos casos, O(n log2 n);

**1.3 -** **Quick Sort:** é o algoritmo de ordenação interna mais rápido e um dos mais usados. Esse método divide o conjunto de n chaves em dois partindo de um pivô, de forma que as chaves menores fiquem a esquerda e as maiores a direita. Cada conjunto é ordenado recursivamente até que sejam combinados de maneira totalmente ordenada. A complexidade no melhor e nos médios casos é de O(n log n), e no pior dos casos O(n²);

**1.4 - Merge Sort:** é um método cuja ideia básica é criar uma sequência ordenada a partir de duas outras também ordenadas. No Mergesort, divide-se uma sequência em pares de dados, as quais são ordenadas e agrupadas em sequências de quatro elementos, e assim sucessivamente até que restem apenas duas partes no final do processo recursivo. Sua complexidade no melhor caso é O(n), e nos médios e piores casos é O(n log n);

**1.5 - Heap Sort:** O heapsort utiliza uma estrutura de dados chamada heap, para ordenar os elementos conforme são inseridos. A heap é um vetor que pode ser visto como uma árvore binária praticamente completa. O método segue a ideia do método de seleção, porém usando uma fila de prioridades.

**1.6 - Bubble Sort:** é um método muito simples e usado para poucos dados. Sua ideia é comparar dois elementos adjacentes e trocá-los de posição se o segundo for maior que o primeiro até que o menor valor fique no início e o maior no final. Sua complexidade no melhor dos casos é de O(n), e nos casos médios e piores, O(n²).

**2 - Tabela de Comparações:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tamanho dos Vetores | 100 | 1.000 | 10.000 | 100.000 | 100 | 1.000 | 10.000 | 100.000 | 100 | 1.000 | 10.000 | 100.000 |
| Método | Movimentações | | | | Comparações | | | | Tempo | | | |
| Seleção | 92 | 991 | 9988 | 99988 | 387 | 6186 | 85874 | 1092105 | 0,001s | 0,004s | 0,248s | 15,241s |
| Shell Sort | 740 | 13916 | 229974 | 3872999 | 748 | 13928 | 229982 | 3873021 | 0,001s | 0,003s | 0,087s | 0,981s |
| Quick Sort | 162 | 2362 | 31323 | 393739 | 483 | 10271 | 133276 | 1706661 | 0,001s | 0,003s | 0,092s | 0,925s |
| Merge Sort | 1344 | 19952 | 267232 | 3337856 | 772 | 10976 | 143616 | 1768928 | 0.003s | 0.003s | 0.122s | 0.896s |
| Heap Sort | 1452 | 22680 | 310153 | 3935696 | 770 | 12616 | 175991 | 2260865 | 0.002s | 0.003s | 0.097s | 0.970s |
| Bubble Sort | 4508 | 510856 | 49750216 | 4994513848 | 2254 | 255428 | 24875108 | 2497256924 | 0.001s | 0.006s | 0.401s | 37.136s |

**3 - Considerações Finais:**

Com a realização deste trabalho, pudemos notar que o Quicksort é realmente o mais rápido tendo em vista que conforme o tamanho dos vetores aumentava, ele se mantinha com o melhor desempenho considerando a quantidade de movimentações, comparações e tempo de execução. Os métodos de Seleção e Bubblesort não tem um desempenho adequado para maior quantidade de dados, nesse caso, podemos ver que o Bubblesort foi o mais lento ao executar o algoritmo com um vetor de cem mil posições preenchido com valores aleatórios. O Mergesort também teve uma queda no desempenho com vetores maiores em comparação com os outros. Acreditamos que isso se deve ao fato de no Mergesort, o vetor sobre uma divisão em duas listas e outras subdivisões recursivas até que possa-se combinar as duas sublistas.

No mais, a realização deste trabalho nos ajudou a perceber na prática a eficiência dos algoritmos e ver na prática o que vimos em sala nos ajudando a entender melhor o funcionamento dos métodos de ordenação.