|  |  |  |
| --- | --- | --- |
| C:\Users\me\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Logomarca_jpg.jpg | **Universidade Federal de Ouro Preto – UFOP**  Instituto de Ciências Exatas e Aplicadas - ICEA  Departamento de Computação e Sistemas - DECSI  CSI488 - Algoritmos e Estrutura de Dados I |  |

**RELATÓRIO DE COMPARAÇÃO DOS PRINCIPÁIS ALGORITMOS DE ORDENAÇÃO**

MATRÍCULA: 17.1.8343

DISCENTE: Vinícius Fiedler Caldeira

DOCENTE: Rafael Frederico Alexandre

João Monlevade

2018

# INTRODUÇÃO:

O trabalho consiste em: criar um programa em linguagem C em que dado um arquivo com *n* números inteiros em uma ordem (crescente, decrescente ou aleatória) realiza-se a ordenação crescente desses dados utilizando os algoritmos de ordenação:

* **Selection-Sort**
* **Insertion-Sort**
* **Shell-Sort**
* **Quick-Sort**

Posteriormente, deve-se fazer as comparações dos resultados do programa quanto aos seguintes temas:

* **Número de Comparações.**
* **Número de Trocas.**
* **Tempo de Execução (em ms “milissegundos”).**

# ESTRUTURA DO PROGRAMA:

O programa foi desenvolvido em linguagem C na **IDE** (**I**ntegrated **D**evelopment **E**nviroment) Code::Blocks e é composto de X módulos:

**1: Bibliotecas utilizadas:**

**stdio.h:** Biblioteca com funções para manipulação de arquivos (fopen, fprintf, fgetc);

**stdlib.h:** Biblioteca para geração de números aleatórios e alocação de memória (malloc, free, rand, srand);

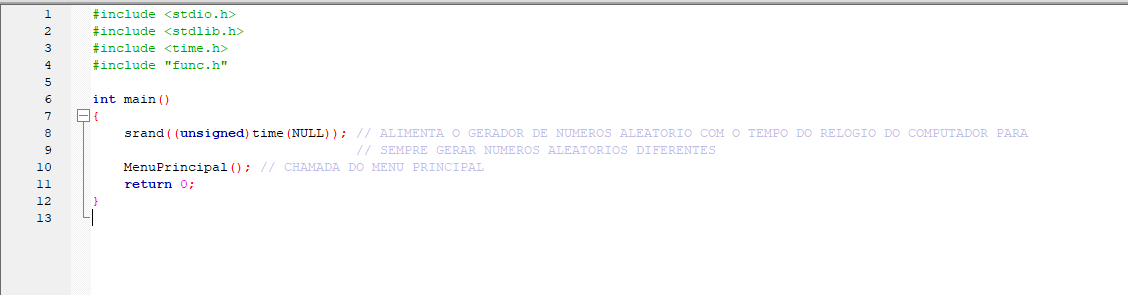
**time.h:** Biblioteca para obter tempo de execução de um algoritmo (time, clock);

**string.h:** Biblioteca com funções de manipulação de strings (fgets, strcpy, strcat, itoa).

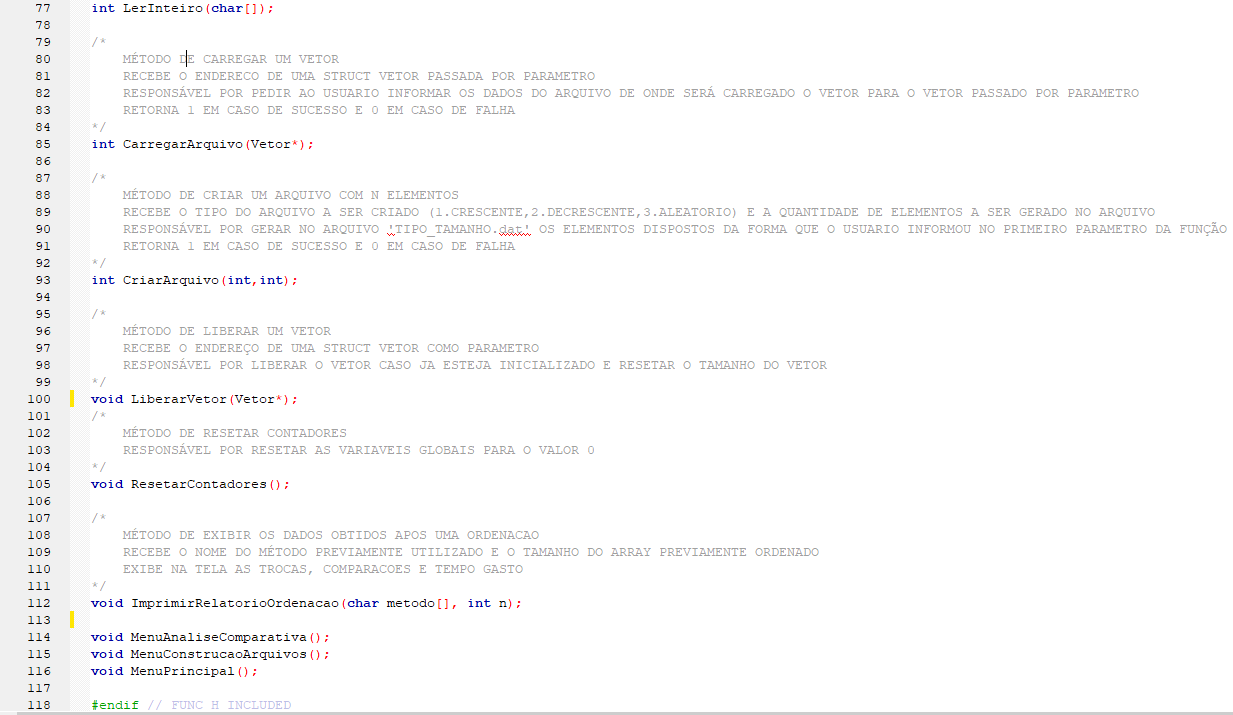
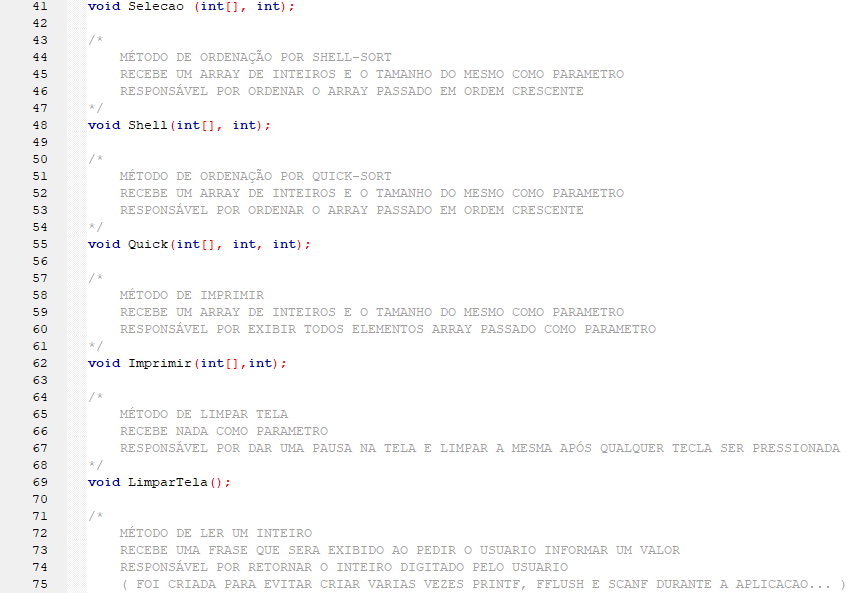
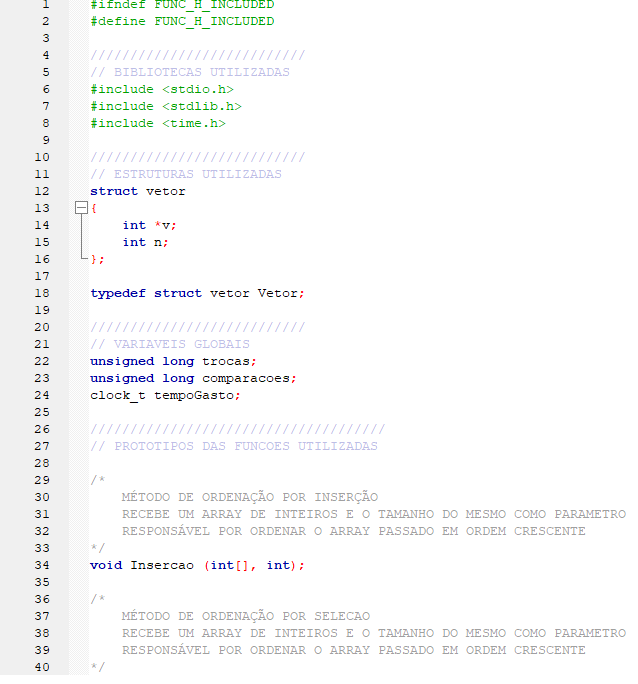
**func.h:** Biblioteca contendo estruturas utilizadas, protótipos das funções utilizadas no desenvolvimento do software e o protótipo dos principáis métodos de ordenação (Insertion-Sort, Selection-Sort, Shell-Sort).

**2: Organização do software:**

***Main.c:*** Ambiente main, a função do menu principal é chamada.



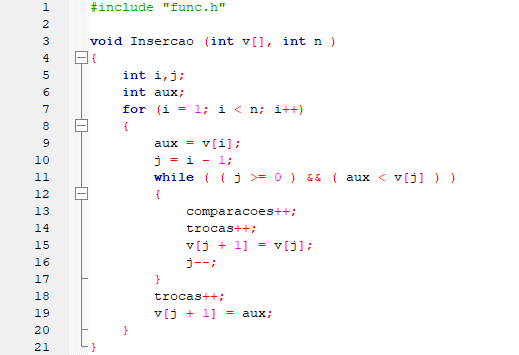
***func.h:*** Ambiente de desenvolvimento dos protótipos e definições de estrututas e variáveis.



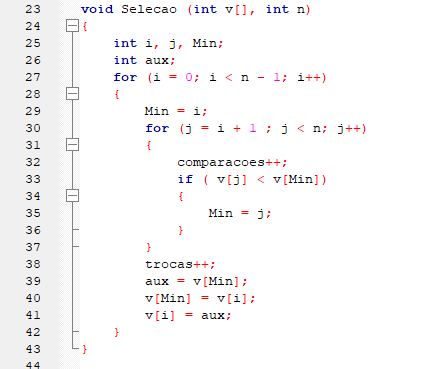
\*\*\*Imagem capturada com zoom inferior a 100%, pressione ctrl + scroll up para mais detalhes.\*\*\*

***func.c:*** Ambiente de desenvolvimento dos métodos e implementação dos principáis métodos de ordenação e demais funções.

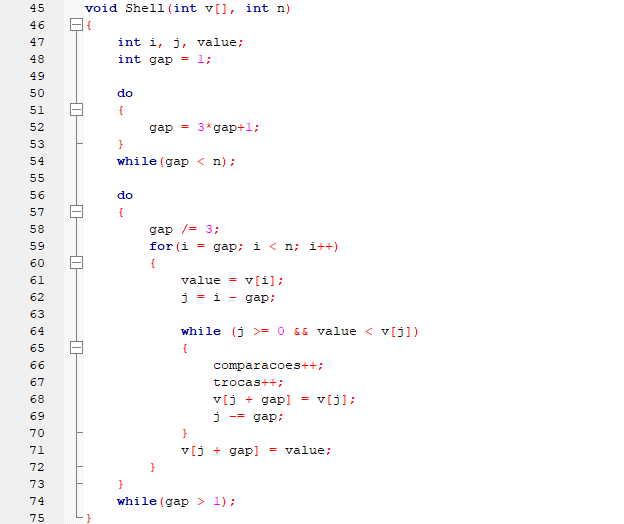
**Insertion-Sort:**



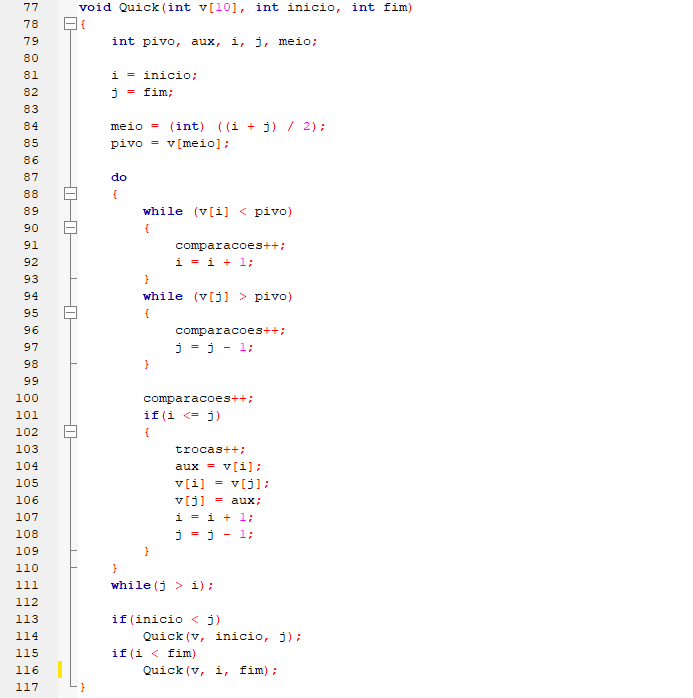
**Selection-Sort:**



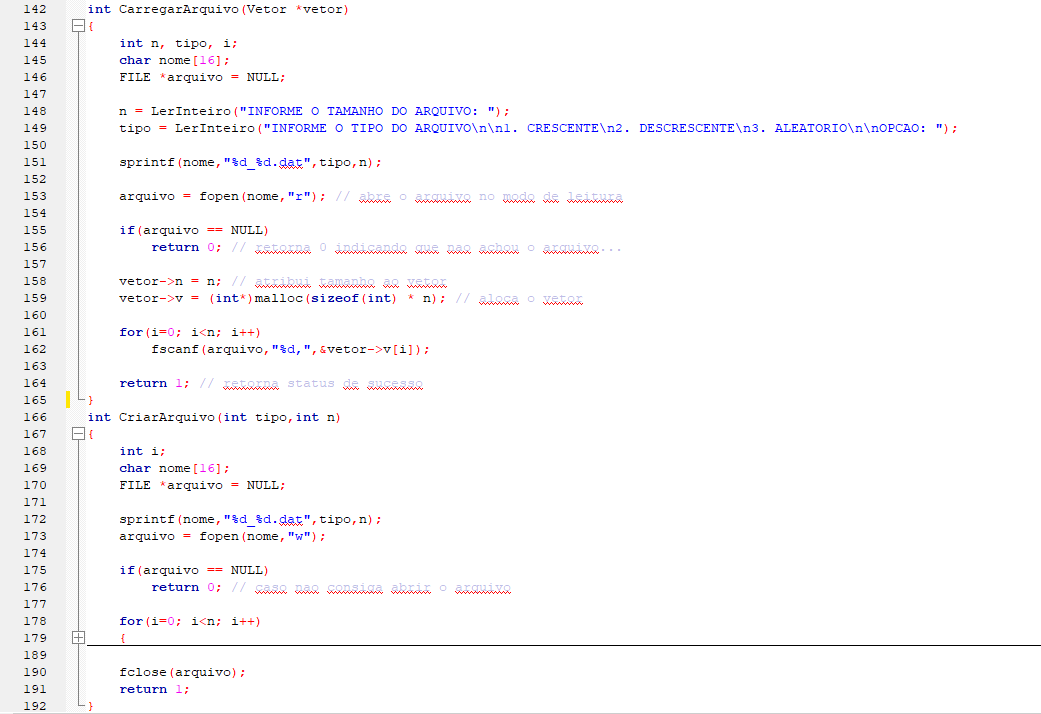
**Shell-Sort:**



**Quick-Sort:**



Funções de **carregar arquivo** e **criar arquivo** também foram desenvolvidas para esse software. Demais funções como ler a sequência de entrada e funções para desalocar variáveis foram abstraidas para melhor entendimento.



**METODOLOGIA APLICADA:**

Para calcular o número de trocas e o número de comparações foi criado variáveis contadoras para ambas as operações.

Para calcular o tempo de execução foi utilizado a média aritmética simples do total de 10 execuções para cada entrada. Veja a equação abaixo:

Equação: Fórmula do cálculo de média aritmética simples.

# EXECUÇÃO DO PROGRAMA:

O programa foi executado entre os dias 11/12/2018 e 12/12/2018 em um hardware com as especificações abaixo:

**Modelo:** AN515-51-75KZ

**Marca:** ACER

**Sistema Operacional:** Microsoft Windows 10 PRO 64bits

**Processador:** Intel® Core™ i7-7700HQ Quad Core

**Memória RAM:** DDR4 16GB

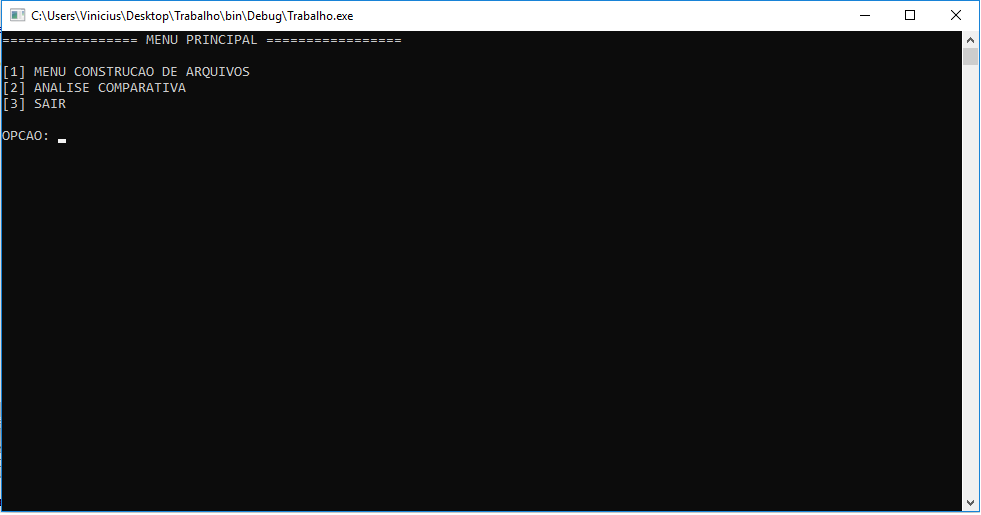
**Placa de video**: Geforce GTX 1050Ti 4GB

**SSD**: 240GB Kingston A1000 M.2 2280 Pcie NVMe

**HDD:** 1TB



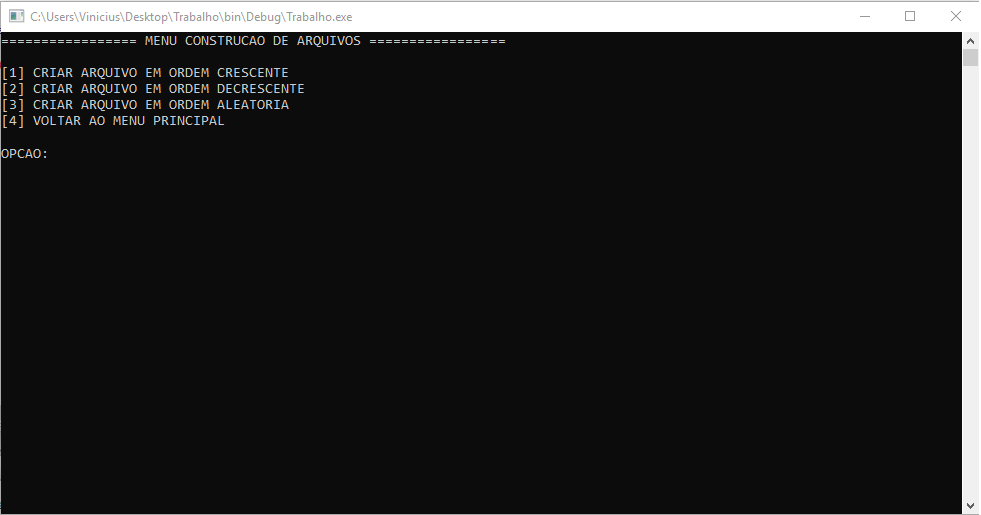
Ao executar o programa ele apresenta um menu de opções e para realizar uma escolha deve-se digitar no teclado o número correspondente e teclar ENTER:



Terminal de execução mostrando o menu inicial do programa.

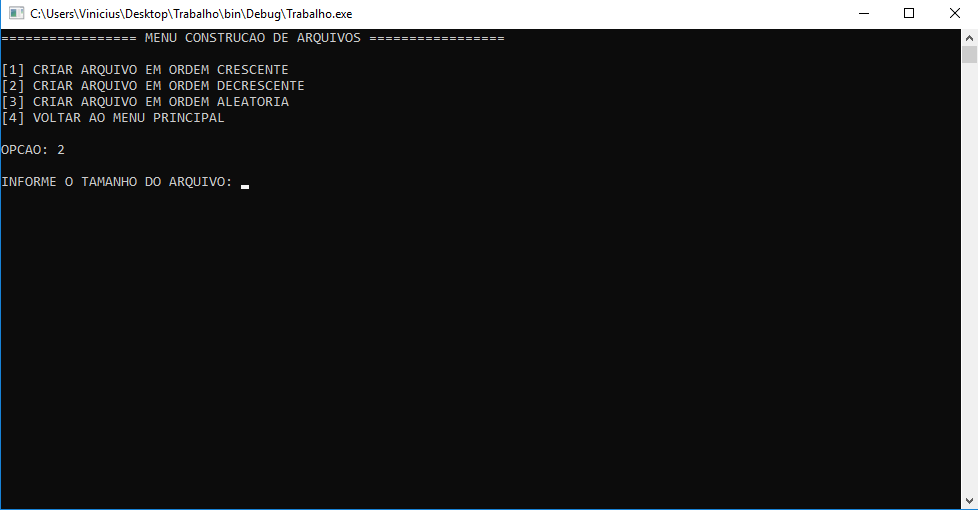
**1: Gerando um arquivo “.dat”.**

Para gerar um arquivo “.dat” executa-se o programa e após aparecer a tela inicial digita-se 1 no teclado e tecla-se ENTER, apos isso o sistema pede que digite qualquer tecla para continuar...apos isso a seguinte tela aparecerá:

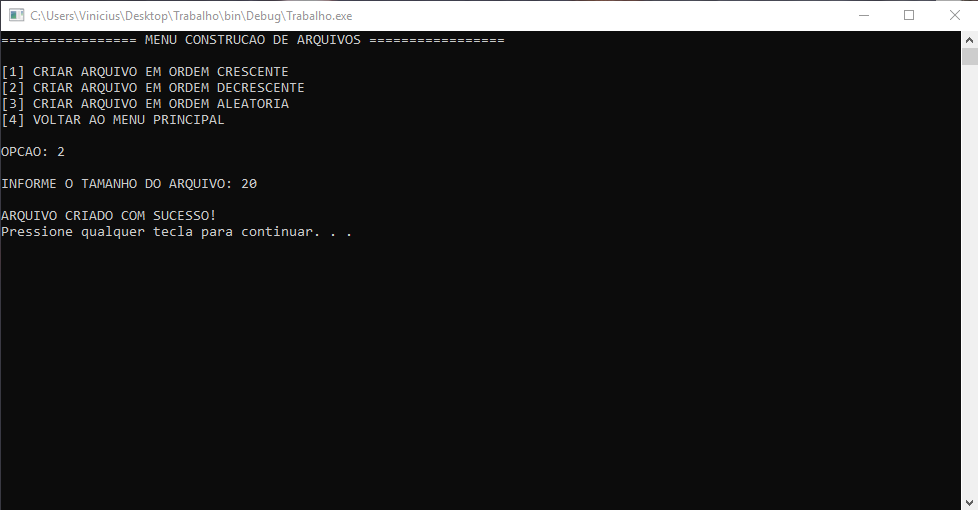


Terminal de execução mostrando o menu de escolha de tipo de arquivo a ser gerado.

Agora digita-se no teclado o número correspondente ao tipo de arquivo que deseja criar ou 4 caso desejar voltar ao menu principal. Após a escolha do tipo de arquivo, o programa solicitará ao usuário que digite a quantidade de números para o arquivo, apos isso pressione enter e em seguida a seguinte mensagem aparecerá:



Terminal de execução mostrando a solicitação do tamanho final do arquivo.

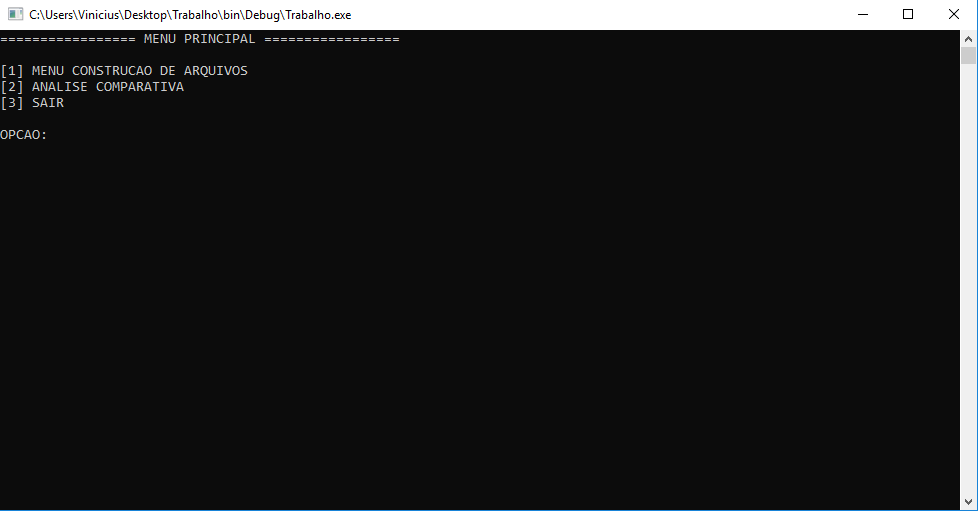


Terminal de execução mostrando a criação de um arquivo.

Após a geração do arquivo personalizado, o programa volta ao menu de construção de arquivos e existe também a opção de voltar ao menu principal.

**2: Comparações dos algoritmos de ordenação através do arquivo “.dat” gerado pelo software.**

Para realizar as comparações deve-se certificar de que o arquivo “.dat” está localizado no mesmo diretório em que o programa está localizado.



Terminal de execução para a análise comparativa.

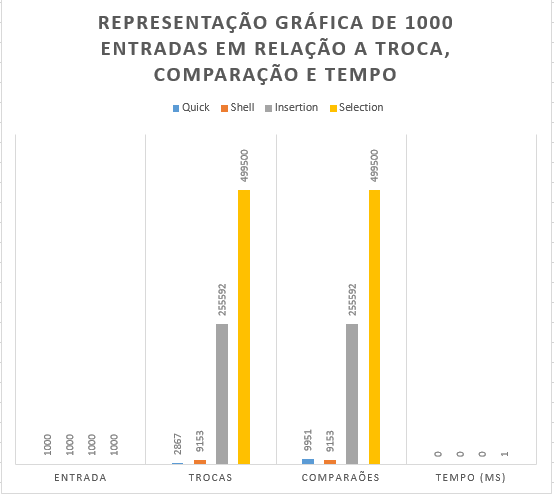
**Observação**: O nome do arquivo deve estar no formato “XX\_YYYY.dat”, onde XX corresponde ao tipo de arquivo (“01” ordenado crescente,”02” ordenado decrescente ou “03” ordenado aleatoriamente) e YYYY corresponde a quantidade de números existentes no arquivo.

Com o software rodando vamos analisar empiricamente cada método de ordenação.

Para a escolha da entrada, foi convencionado os tamanhos de 1.000, 100.000 e 200.000.

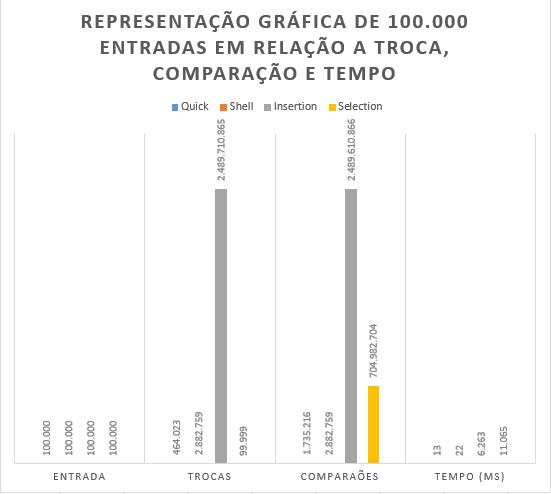
**Teste de ordenação com arquivo em ordem aleatória para crescente**

**1: Gráfico de 1000 entradas em relação a troca, comparação e tempo.**



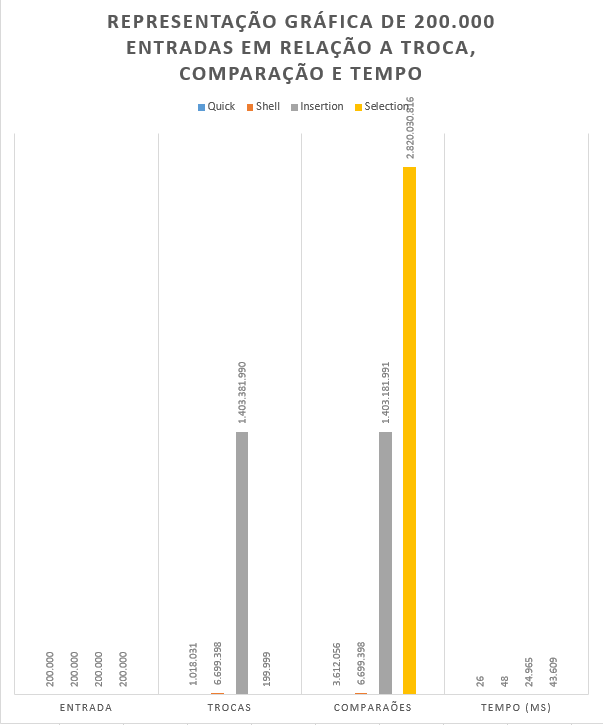
Percebe-se que o algoritmo Selection-Sort com uma entrada de 1000 valores se mostrou o pior em termos de trocas e comparações que todos os outros, apesar do tempo ser maior, continua O(1).

**2: Gráfico de 100.000 entradas em relação a troca, comparação e tempo.**



Desta vez o algoritmo Insertion-Sort com uma entrada de 100.000 valores se mostrou o pior em termos de trocas e comparações e também em relação ao tempo.

**3: Gráfico de 200.000 entradas em relação a troca, comparação e tempo.**



Novamente o algoritmo Insertion-Sort com uma entrada de 200.000 valores se mostrou o pior em termos de trocas e ficou em segundo lugar atrás do Selection-Sort em comparações e também em relação ao tempo.

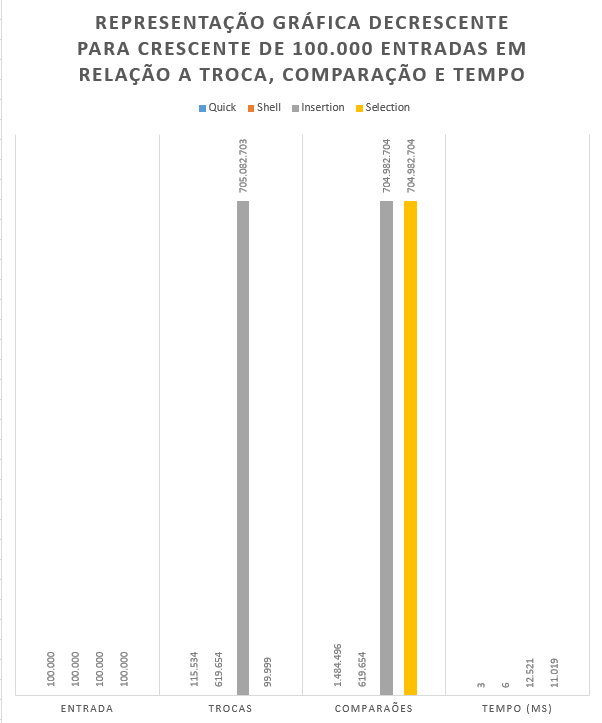
**Teste de ordenação com arquivo em ordem decrescente para crescente**

**4: Gráfico de 1000 entradas decrescente para crescente em relação a troca, comparação e tempo.**



Desta vez o algoritmo Insertion-Sort com uma entrada de 1.000 valores se mostrou o pior em termos de trocas e ficou empatado em comparações e também em relação ao tempo.

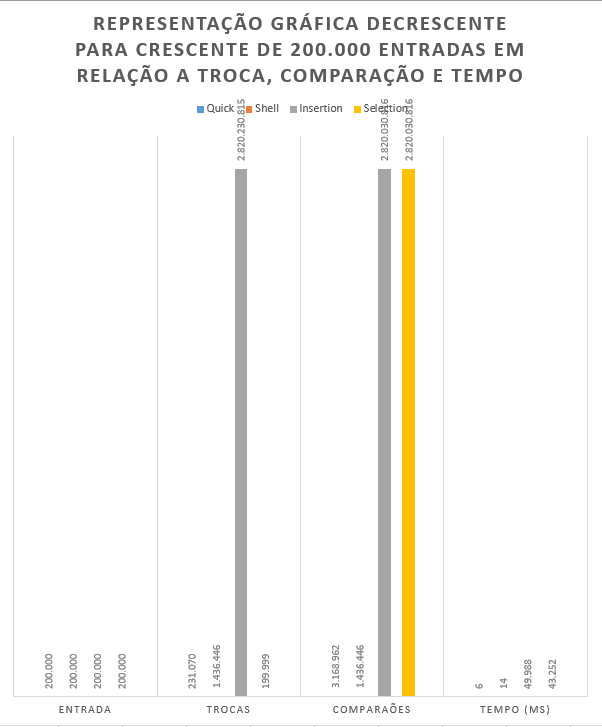
**5: Gráfico de 100.000 entradas decrescente para crescente em relação a troca, comparação e tempo.**



O Insertion-Sort com uma entrada de 100.000 valores se mostrou o pior em termos de trocas e empatou novamente em comparações com o Selection-Sort e gastou mais tempo.

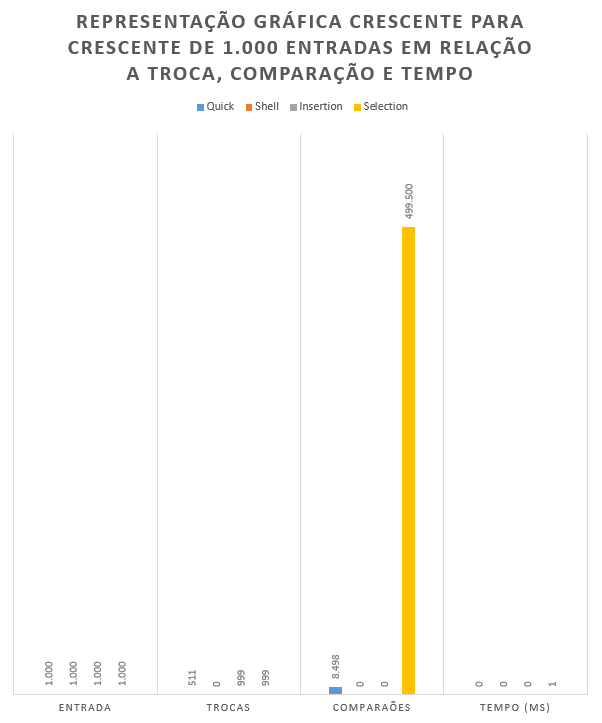
**6: Gráfico de 200.000 entradas decrescente para crescente em relação a troca, comparação e tempo.**

O algoritmo Insertion-Sort com uma entrada de 200.000 valores se mostrou o pior em termos de trocas, empatou novamente em comparações com o Selection e o tempo foi ligeiramente maior.



**Teste de ordenação com arquivo em ordem crescente para crescente**

**7: Gráfico de 1.000 entradas crescente para crescente em relação a troca, comparação e tempo.**

****

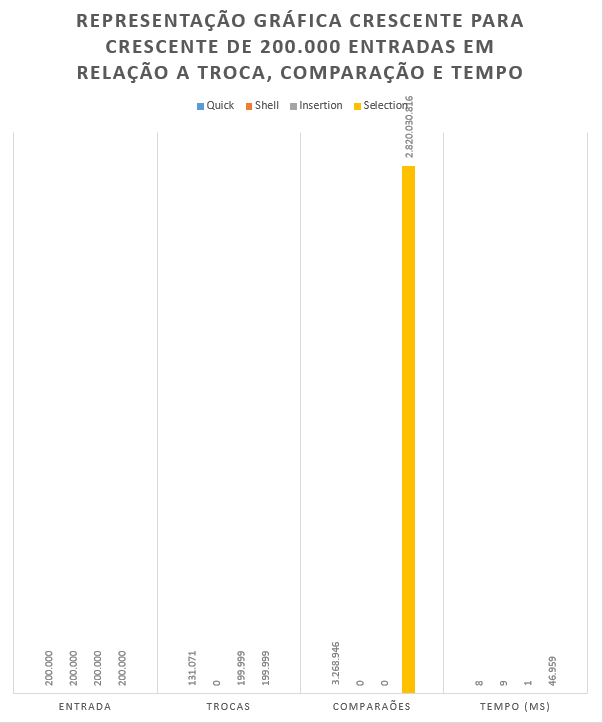
Desta vez o algoritmo Selection-Sort com uma entrada de 1.000 valores se mostrou o pior em termos de comparações e o Shell-Sort foi o melhor em todas as provas e também em relação ao tempo.

**8: Gráfico de 100.000 entradas crescente para crescente em relação a troca, comparação e tempo.**



Mais uma vez o Selection-Sort com uma entrada de 100.000 valores se mostrou o pior em termos de comparações, empatou com o Shell e foi levemente inferior em relação ao Quick em trocas. O Selection foi o que mais gastou tempo.

**9: Gráfico de 200.000 entradas crescente para crescente em relação a troca, comparação e tempo.**



Desta vez o algoritmo Selection-Sort com uma entrada de 200.000 valores se mostrou o pior em termos de comparações, empatou novamene com o Insertion e ficou logo atrás do Quick, o Shell-Sort foi o melhor em todas as provas e o Selection gastou muito tempo comparado a todos os outros.

**Conclusao:**

Com os resultados vistos nos 3 diferentes tipos de testes, conclui-se que diversos fatores influenciam na eficácia de cada metodo, maquina utilizada, ordem que os valores já se encontram e o tamanho da entrada. O Quick sort certamente é o algoritmo mais eficiente em listas totalmente desordenadas, ele se torna muito eficiente em relação aos outros no quesito de tempo. O Insertion sort se mostrou mais eficiente que todos os outros algoritmos em relação ao tempo e comparações. O Selection Sort nao necessita de vetor auxiliar, é simples de ser implementado, é um dos mais velozes na ordenação de entradas pequenas mas é um dos mais lentos em entradas grandes e nao é estável. A ordenação do Shell sort utiliza a quebra sucessiva da sequência a ser ordenada e implementa a ordenação por inserção na sequência obtida. Devido a sua complexidade possui excelentes desempenhos em N muito grandes

**Referências:**

[**https://medium.com/@henriquebraga\_18075/algoritmos-de-ordena%C3%A7%C3%A3o-ii-selection-sort-8ee4234deb10**](https://medium.com/@henriquebraga_18075/algoritmos-de-ordena%C3%A7%C3%A3o-ii-selection-sort-8ee4234deb10)

**https://www.geeksforgeeks.org/insertion-sort/**

<https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/shellsort.pdf>

[**https://pt.wikipedia.org/wiki/Quicksort**](https://pt.wikipedia.org/wiki/Quicksort)