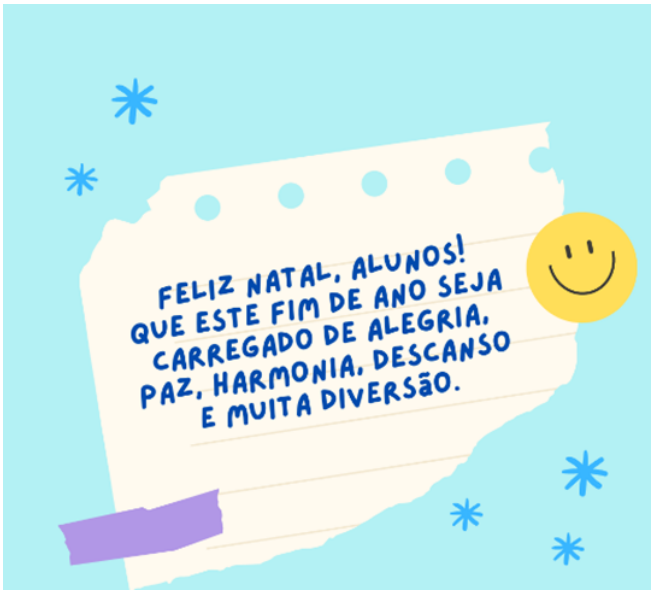
TRABALHO DE ORDENAÇÃO

Disciplina SIN 213 - Projeto de Algoritmos

Data de Entrega: 13 de dezembro de 2023
Valor: 8 Vistos



1. Ordenação: Trabalho sobre HeapSort e Filas de Prioridades (Tudo minimo)

O HeapSort é um algoritmo de ordenação que utiliza uma estrutura de dados chamada heap (ou monte) para organizar os elementos. Ele segue os seguintes passos:

- ➔ **Construção do Heap Máximo:** Os elementos do array são organizados de forma a criar um heap máximo, garantindo que cada pai seja maior ou igual aos seus filhos.
- ➔ **Ordenação:** O elemento máximo (raiz do heap) é trocado com o último elemento do array não ordenado. O heap é reorganizado para manter a propriedade de heap máximo. Esse processo é repetido até que todo o array esteja ordenado.

O HeapSort possui uma complexidade de tempo de O(n log n) para o pior e médio caso, sendo um algoritmo eficiente para ordenação em locais de memória limitada.

1.1. Objetivo:

3.1.1. Experimentar algoritmos.

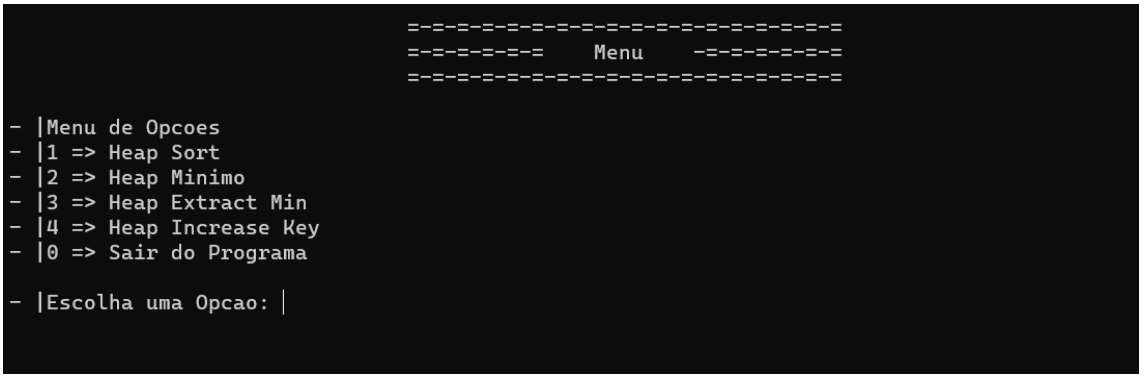
3.1.2. Comparar os tempos dos experimentos.

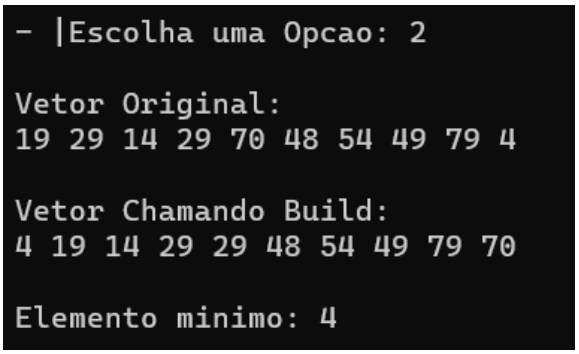
3.1.3. Analisar os tempos e especular as circunstâncias em que um algoritmo se sobressai ao outro.

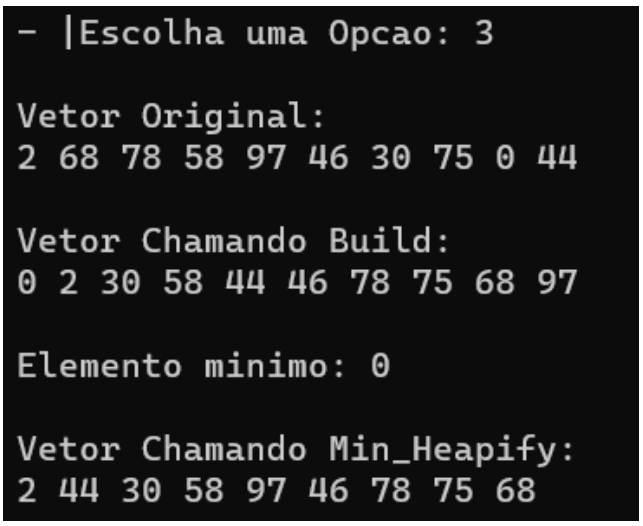
1.2. Código:

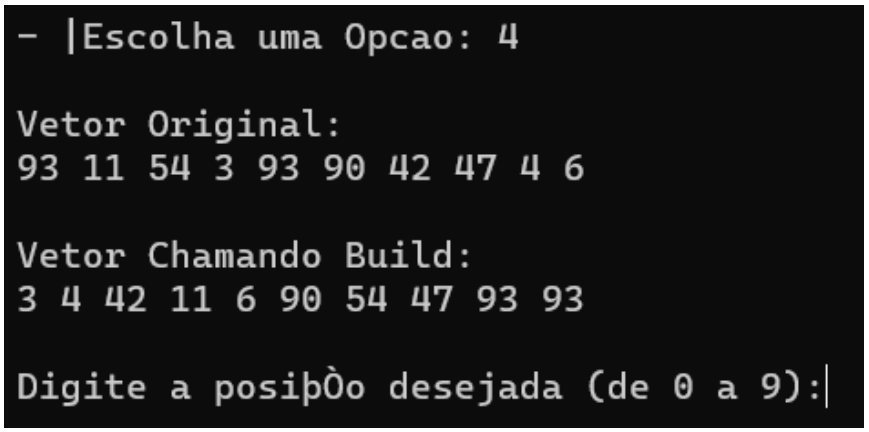
Implemente em **linguagem C ou C++** os algoritmos de ordenação (Usando vetor e arquivos):

- **Heapsort (Mínimo)**
    - Criar e aplicar o Algoritmo do Heapsort para ordenação de entradas Crescentes, Decrescentes e Aleatórias de tamanhos entre 10 a 1.000.000, da mesma forma dos códigos anteriores. Lembre-se de utilizarem a BUILD_MIN_HEAP e MIN_HEAPIFY para a criação do algoritmo.
- **HEAP_MINIMUM**
    - Criar um vetor da ordem e tamanho que desejar. Após isto chame a BUILD_MIN_HEAP para deixar o vetor em modo de Heap mínimo para retornar o menor elemento que se encontra na raiz. Printe o heap criado e o elemento mínimo para conferir.
- **HEAP_EXTRACT_MIN**
    - Criar um vetor da ordem e tamanho que desejar. Após isto chame a BUILD_MIN_HEAP para deixar o vetor em modo de Heap mínimo para extrair o mínimo que se encontrará na raiz. Lembrando que depois de remover o elemento da primeira posição, haverá uma troca e necessitará deixar o vetor em modo heap após a remoção, só que agora a utilizando o MIN_HEAPIFY começando no índice do primeiro elemento. Printe o menor elemento retornado e o heap criado antes e depois da operação de extração.
- **HEAP_INCREASE_KEY**
    - Criar o vetor; deixar em modo de heap com a BUILD_MIN_HEAP; inserir o elemento 1000 em qualquer posição que o aluno desejar.
- **MAX_HEAP_INSERT**
    - Criar o vetor; deixar em modo de heap com a BUILD_MIN_HEAP; inserir o elemento 1000 no heap.



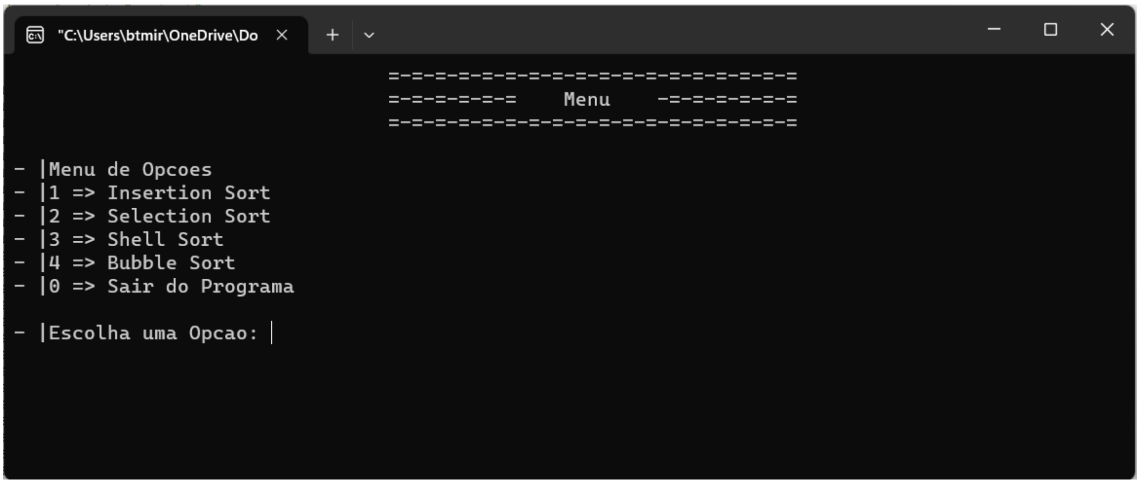


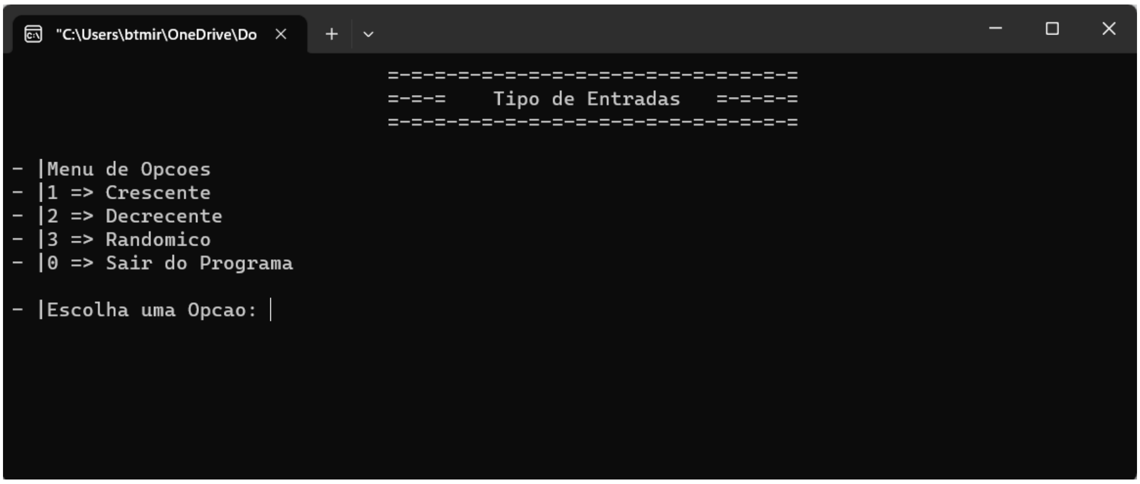


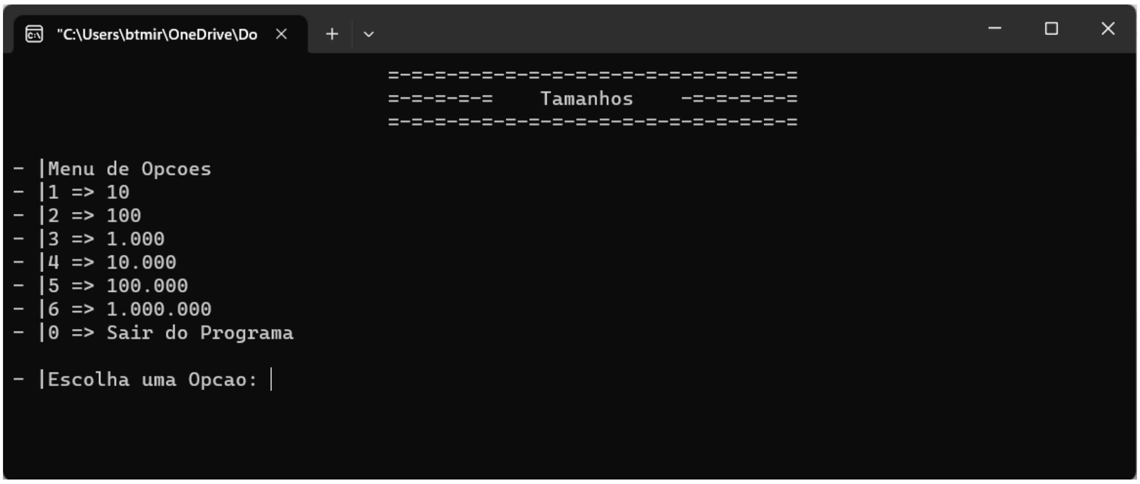


Exemplo de Saídas dos menus 2, 3 e 4;

1.2.1. Sugestão de Menu: implemente um **Menu** na aplicação para melhor interação com o usuário. Este Menu poderá conte:







1.2.2 Geração de Entradas: O código deve gerar instâncias de tamanhos: **10, 100, 1.000, 10.000, 100.000** e **1000,000** para todas as formas **(randômico, crescente e decrescente)**, e execute o algoritmo implementado.

12.3 Geração de Pastas e Arquivos: O código deve gerar automaticamente as pastas e arquivos, salvando seus respectivos valores de cada.

Na parte de arquivo, gere:

- Um **Arquivo de Entrada** contendo as instâncias geradas.
- Um **Arquivo de Saída** com as instâncias ordenadas.
- Um **Arquivo de Tempo** gasto pela ordenação.

Os arquivos tanto os de **Entrada** como o de **Saída** devem ter na primeira linha o **tamanho da instância** e o restante dos dados, um por linha, como é indicado na Figura 1. O **Arquivo de Tempo** salve apenas o tempo final gasto em segundos, ou formate em sua preferência.

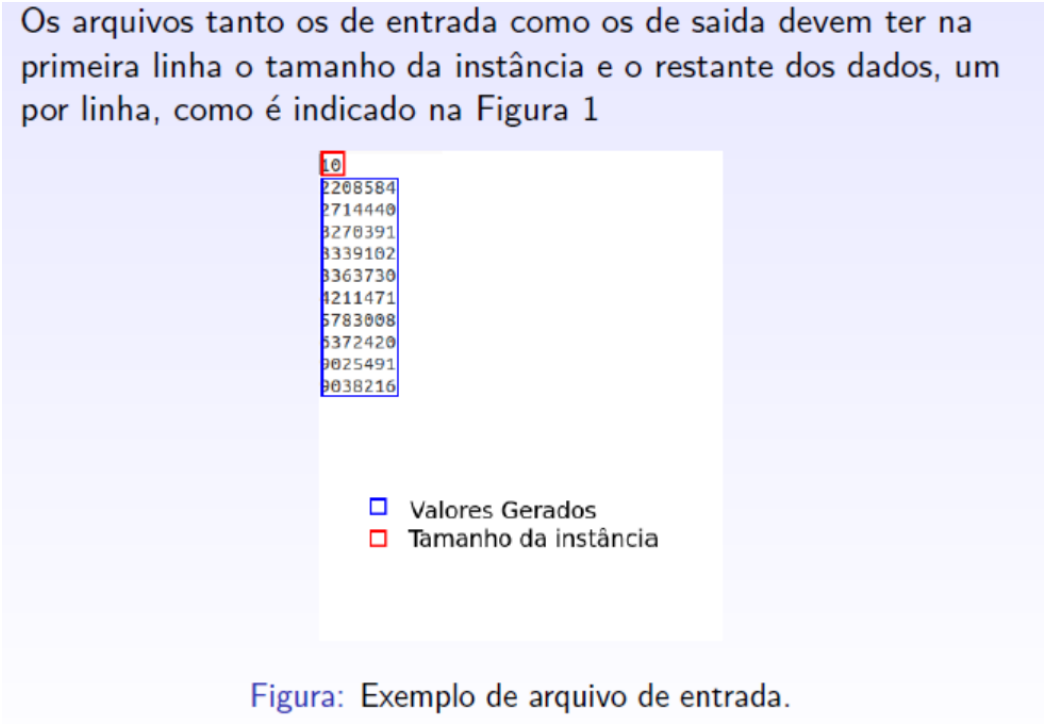


Figura 1: Exemplo de arquivo de entrada.

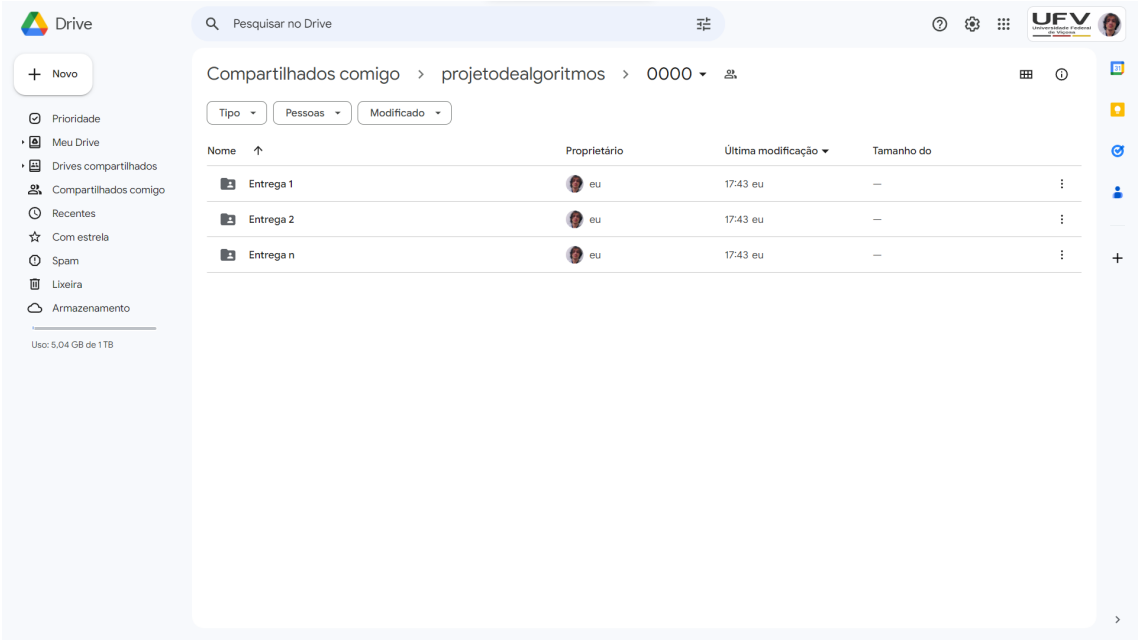
2. Entrega:

2.1. A entrega dos códigos (Código fonte e arquivos de entrada, tempo e o de saída, para cada instância do trabalho testando cada algoritmo deste trabalho) deverão ser colocadas em suas respectivas pastas do Drive, diferenciando por pastas de cada entrega.

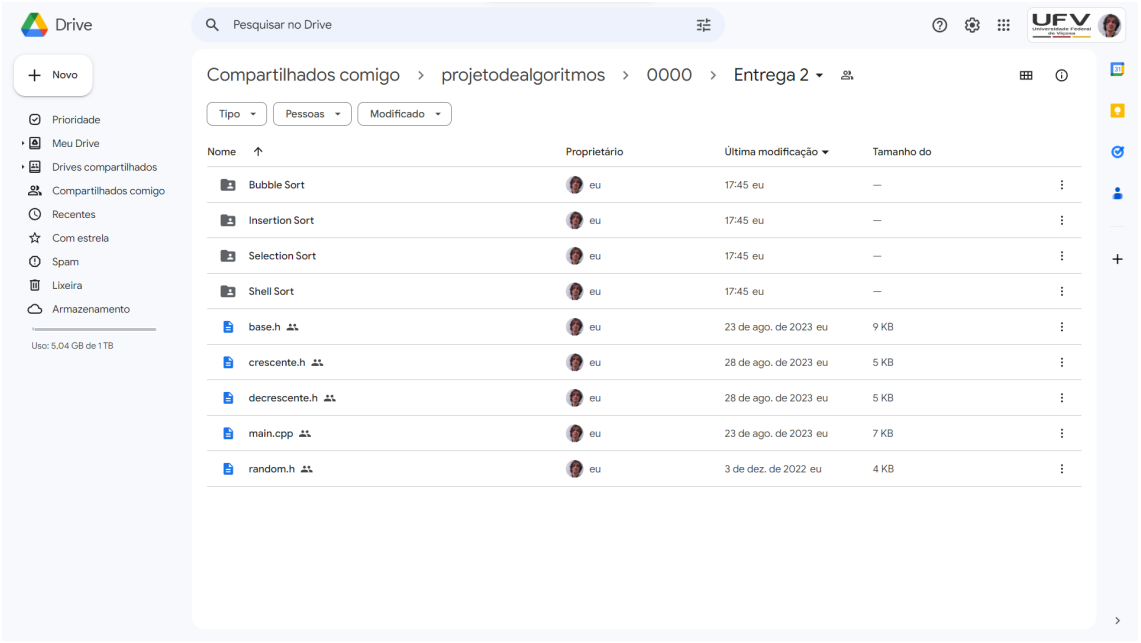
2.2. A entrega do relatório deverá ser enviada no Moodle em formato PDF.

3. Exemplo de Organização:

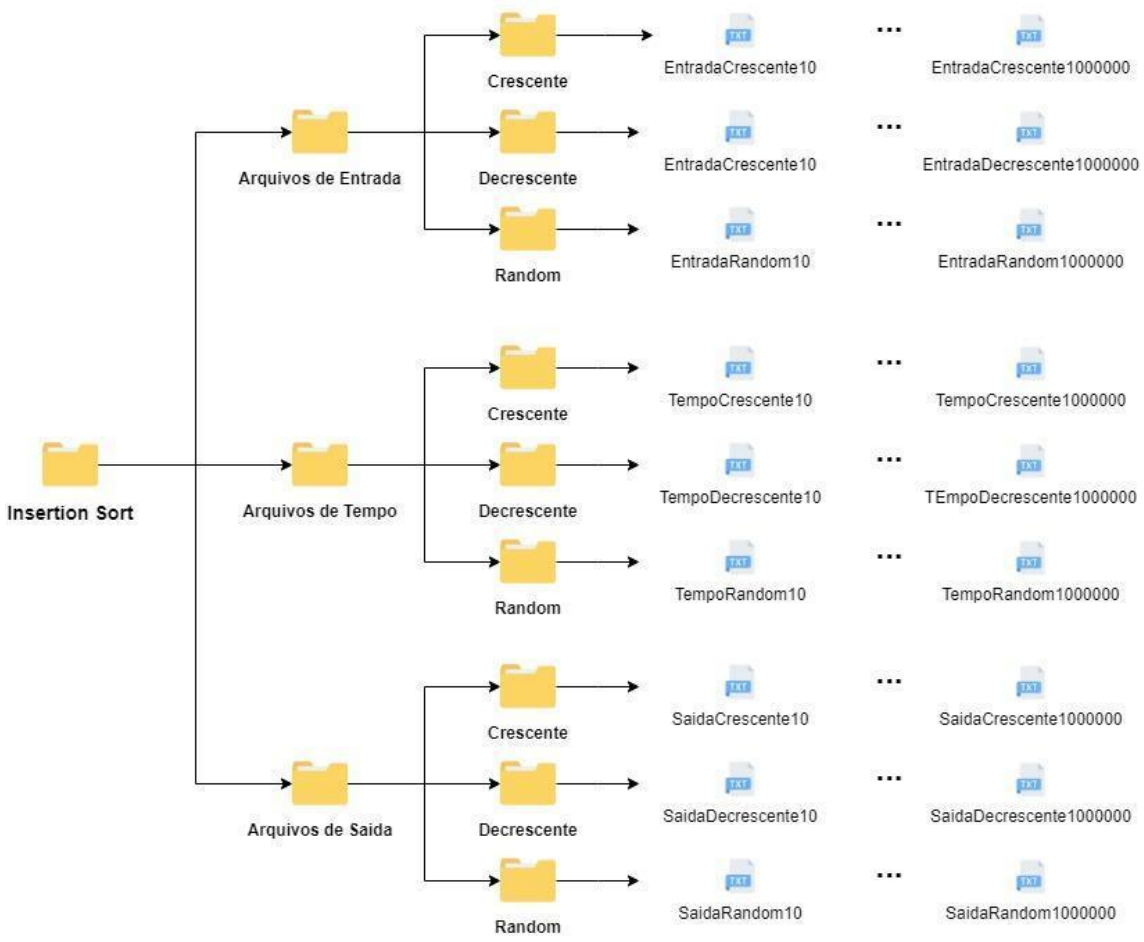
3.1. Crie pastas para diferenciar cada entrega do seu trabalho.



3.2 Dentro da pasta deve ser enviado o seu código fonte e as pastas dos respectivos Algoritmos feitos, contendo os arquivos de entrada, saída e tempo conforme a imagem abaixo.



3.3. Cada pasta deve seguir o seguinte formato: **NÃO ESQUEÇA QUE DEVE SER FEITO A CRIAÇÃO DAS PASTAS E ARQUIVOS PELO CÓDIGO.**



Faça a criação do mesmo para os outros algoritmos para a realização do relatório. **NÃO ESQUEÇA QUE DEVE SER FEITO A CRIAÇÃO DAS PASTAS E ARQUIVOS PELO CÓDIGO.**

Sugestão: Gere as entradas em um vetor e salve elas no arquivo de entrada com seu respectivo tamanho. Em seguida faça a ordenação do vetor, calculando o tempo gasto e salvando ela em um arquivo de tempo com seu respectivo tamanho. Por último salve o vetor organizado no arquivo de Saída com o seu respectivo tamanho.

4. Resultado Esperado

4.1. Crie um relatório seguindo os tópicos abaixo:

- Capa e Contracapa;
- Resumo;
- Sumário;
- Introdução;
- Algoritmo;
    - Algoritmo x;
- Análise de Complexidade;

    - Algoritmo x;
- Tabelas e Gráficos;
- Conclusão;
- Referências;

Faça uma boa **introdução** de **uma página** falando sobre como funciona cada algoritmo de forma resumida.

Provar a Análise de Complexidade de cada algoritmo em seus respectivos casos: o melhor, médio e o pior caso. (exatamente como foi feito em sala de aula para o Insertion Sort, coloque um custo para cada linha e o número de vezes que ela irá ser executada e faça os cálculos)

Explicar o funcionamento do HEAPSORT com seu modo de manutenção com mínimo e suas respectivas filas de prioridades. Dar a complexidade como feito em sala de aula.

É necessário fazer a **conclusão** que você obteve sobre os algoritmos, e comparar caso possível com os demais algoritmos feitos uns com os outros apontando suas qualidades e defeitos e explicando o porquê isso acontece, comparando as entradas, tempos, e algoritmos e casos.

Faça uma tabela **para cada algoritmo**. Deve ser feito a execução do código proposto gerando as saídas de tempo das instâncias: 10, 100, 1.000, 10.000, 100.000 e 1.000.000 que serão utilizadas para fazer a tabela de comparação e os gráficos.

Faça um gráfico **para cada algoritmo** e um **gráfico geral** contendo os algoritmos propostos. Cada um dos gráficos deve possuir duas curvas: uma representando o comportamento do algoritmo quando submetidos a dados em ordem decrescente e outra quando submetido a dados já ordenados.

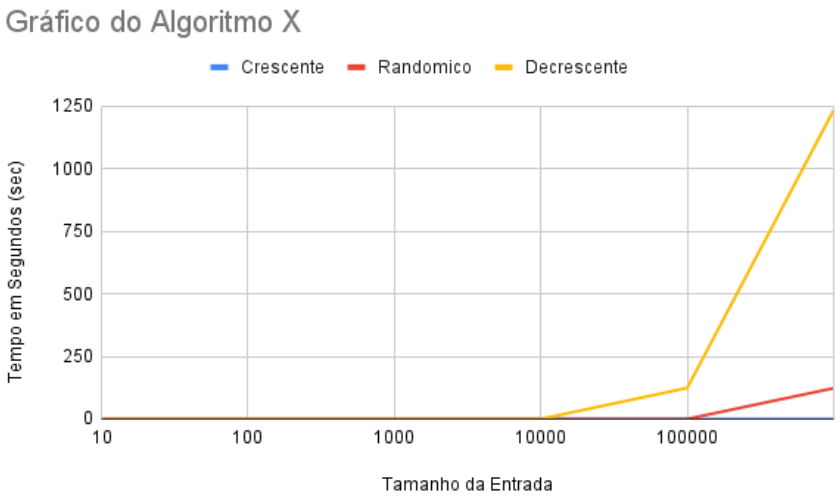


Figura 2: Exemplo de Gráfico do Algoritmo X com dados fictícios.

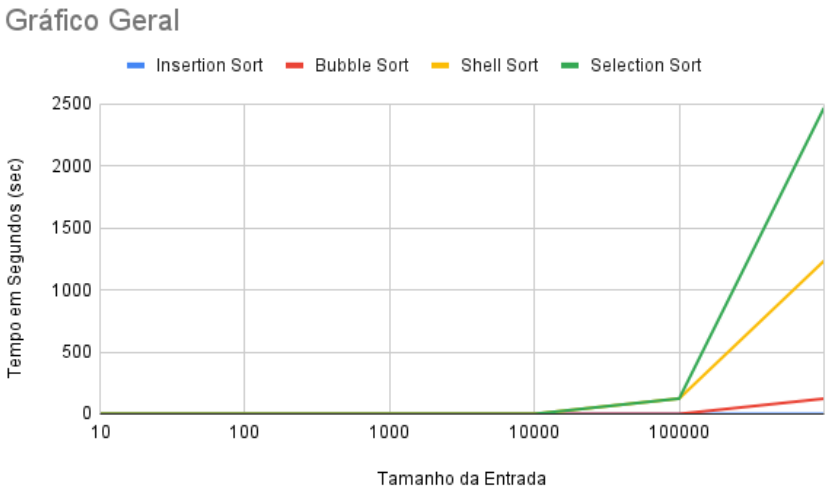


Figura 3: Exemplo de Gráfico Geral usando resultados de cada algoritmo usando dados fictícios.