Universidade Federal de Viçosa - Campus Florestal

Trabalho prático 03 - Algoritmos e Estrutura de Dados I

Arthur De Bellis Gomes - 03503

Pablo Ferreira - 03480

Saulo Miranda Silva - 03475

Objetivo do trabalho

Este trabalho prático tem como objetivo colocar em prática o que aprendemos em sala de aula sobre algoritmos de ordenação, comparando os seus desempenhos em 12 cenários diferentes, para mostrar que para cada situação há um algoritmo mais viável do que os outros.

Projeto do Sistema Implementado

Inicialmente a divisão de funções definiu que cada um ficaria encarregado de implementar 2 algoritmos de ordenação, as outras funções(Menu, entrada por arquivo, preenchimento das matrizes, etc) que envolviam a implementação do Tad do primeiro trabalho foram feitas com a colaboração dos 3 integrantes, para garantir um melhor aproveitamento do tempo e do trabalho em equipe.

Principais decisões do grupo

As principais decisões do grupo foram em torno da forma de implementação no tocante ao preenchimento do vetor, formas de ordenação e configuração das funções implementadas.

Módulos desenvolvidos

As funções adicionadas ao código do primeiro trabalho foram:

- *VetorAleatorio*: Esta função recebe um vetor auxiliar como parâmetro e o altera, criando números aleatórios para preencher o vetor, os valores deste servem como índice do vetor de matrizes alocadas, sua função é provocar um preenchimento aleatório.

```c
void VetorAleatorio(int *VetorIndex, int Tamanho, int Preenchido){
  int aleatorio;
  int confirma = 0, cont = 0, i, aux = 0;
  //Ciclo de repetição
  while(confirma == 0){
    //Gerando número aleatório
    aleatorio = (rand()%(Tamanho-1));
    for (i = 0; i < aux; i++){
      if(aleatorio != VetorIndex[i])
        cont++;
        //Se o valor criado for diferente dos já existentes, incrementa-se o contador
    }
    //Se o contador for igual ao auxiliar, significa que o valor criado é diferente dos já existentes, ou seja, pode ser inserido
    if(cont == aux){
      VetorIndex[aux] = aleatorio;
      aux++;
    }
    cont = 0;
    //Se o valor do auxiliar chegar no número de casas a serem preenchidas, a função é parada
    if(aux == Preenchido)
      confirma = 1;
  }
}
```

- SetIdentificador: Função para definir de forma aleatória os IDs de cada matriz.

```c
39    void SetIdentificador(TipoVetor *Vetor, int a){
40      int NovoValor;
41      int confirma = 0, cont = 0, i;
42      //O processo para a criação de um valor aleatório para o ID assemelha-se muito ao da criação do vetor
43      while (confirma == 0) {
44        NovoValor = (rand()%1000000);
45        for(i = 0; i < a; i++){
46          if(NovoValor != Vetor->Matriz[i].IdentificadorDeMatriz){
47            cont++;
48          }
49        }
50        //A execução é parada nessa parte porque a função é chamada uma vez por matriz.
51        if(cont == a){
52          Vetor->Matriz[a].IdentificadorDeMatriz = NovoValor;
53          confirma = 1;
54        }
55      }
56    }
57
```

- AlocaVetor: Função para alocar o Vetor de matrizes.

```
void AlocaVetor(TipoVetor *Vetor, int Tamanho){
  int i;
  //Alocando as matrizes fazendo um vetor
  Vetor->Matriz = (TipoMatriz *)malloc(Tamanho*sizeof(TipoMatriz));
  for(i=0; i<Tamanho; i++){
    IniciaMatriz(&(Vetor->Matriz[i]));
    SetIdentificador(Vetor, i);
  }
}
```

- vooAleatorio: Cria um voo com todos os componentes aleatórios. Estes voos criados por essa função são inseridos no vetor de matrizes.

```
void vooAleatorio(TipoMatriz *itemMatriz){

  TVoo vooReserva;

  //Declaração das variáveis básicas.
  int i;

  int aux1, aux2, aux3;

  for(i = 0; i < 10; i++)
    {
      //Criando voos aleatoriamente

      IniciaVoo(&vooReserva);
      SetVid(&vooReserva);

      vooReserva.horaDecolagem = (rand()%23);
      vooReserva.minutosDecolagem = (rand()%59);

      vooReserva.horaPouso = (rand()%23);
      vooReserva.minutosPouso = (rand()%59);

    //=============================================================

      aux1 = (rand()%25)+65;
      aux2 = (rand()%25)+65;
      aux3 = (rand()%25)+65;

      vooReserva.aeroportoPouso[0] = aux1;
      vooReserva.aeroportoPouso[1] = aux2;
      vooReserva.aeroportoPouso[2] = aux3;
    //=============================================================

      aux1 = (rand()%25)+65;
      aux2 = (rand()%25)+65;
      aux3 = (rand()%25)+65;

      vooReserva.aeroportoDecolagem[0] = aux1;
      vooReserva.aeroportoDecolagem[1] = aux2;
      vooReserva.aeroportoDecolagem[2] = aux3;

    //=============================================================
      vooReserva.identificadorPista = (rand()%100);

      InserirMVoo(itemMatriz, &vooReserva);

}}
```

-

- PreencheVetor: Função que é usada para preencher as casas correspondentes do vetor com as matrizes.

VetorMatriz.c

```c
void PreencheVetor(TipoVetor *Vetor, int cenario, int Tamanho){
  int i = 0, j = 0, a=0, b=0;

  switch (cenario) {
    case 1:
      a = 73;
      b = 10;
      break;
    case 2:
      a = 365;
      b = 10;
      //printf("certo %d\n", cenario);
      break;
    case 3:
      a = 73;
      b = 100;
      break;
    case 4:
      a = 365;
      b = 100;
      break;
    case 5:
      a = 730;
      b = 10;
      break;
    case 6:
      a = 3650;
      b = 10;
      break;
    case 7:
      a = 730;
      b = 100;
      break;
    case 8:
      a = 3650;
      b = 100;
      break;
    case 9:
      a = 7300;
      b = 10;
      break;
    case 10:
      a = 36500;
      b = 10;
      break;
    case 11:
      a = 7300;
      b = 100;
      break;
    case 12:
      a = 36500;
      b = 100;
      break;
  }
```

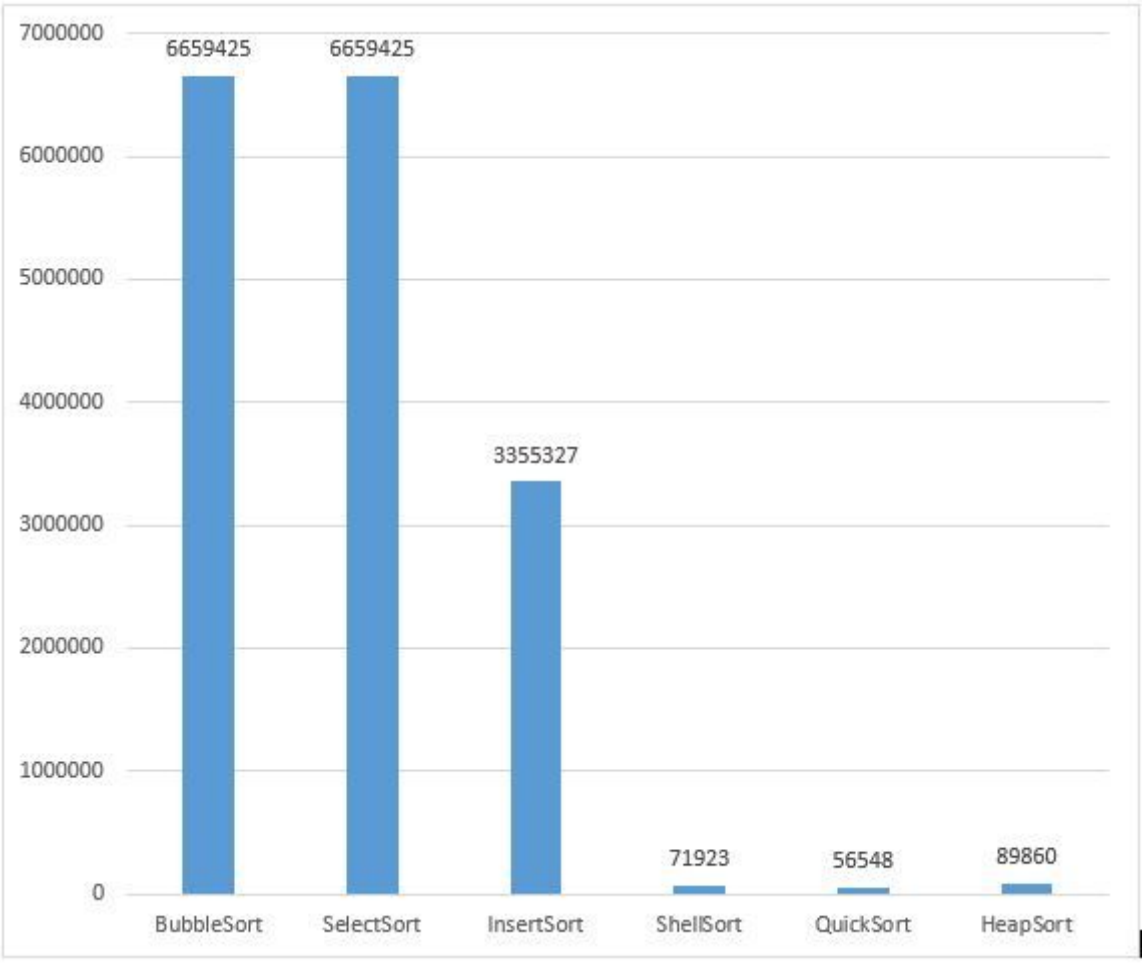
Além dessas funções criadas pelo grupo, temos as funções de ordenação que só tiveram pequenas alterações para se adequar à ordenação.

Especificações da Máquina:

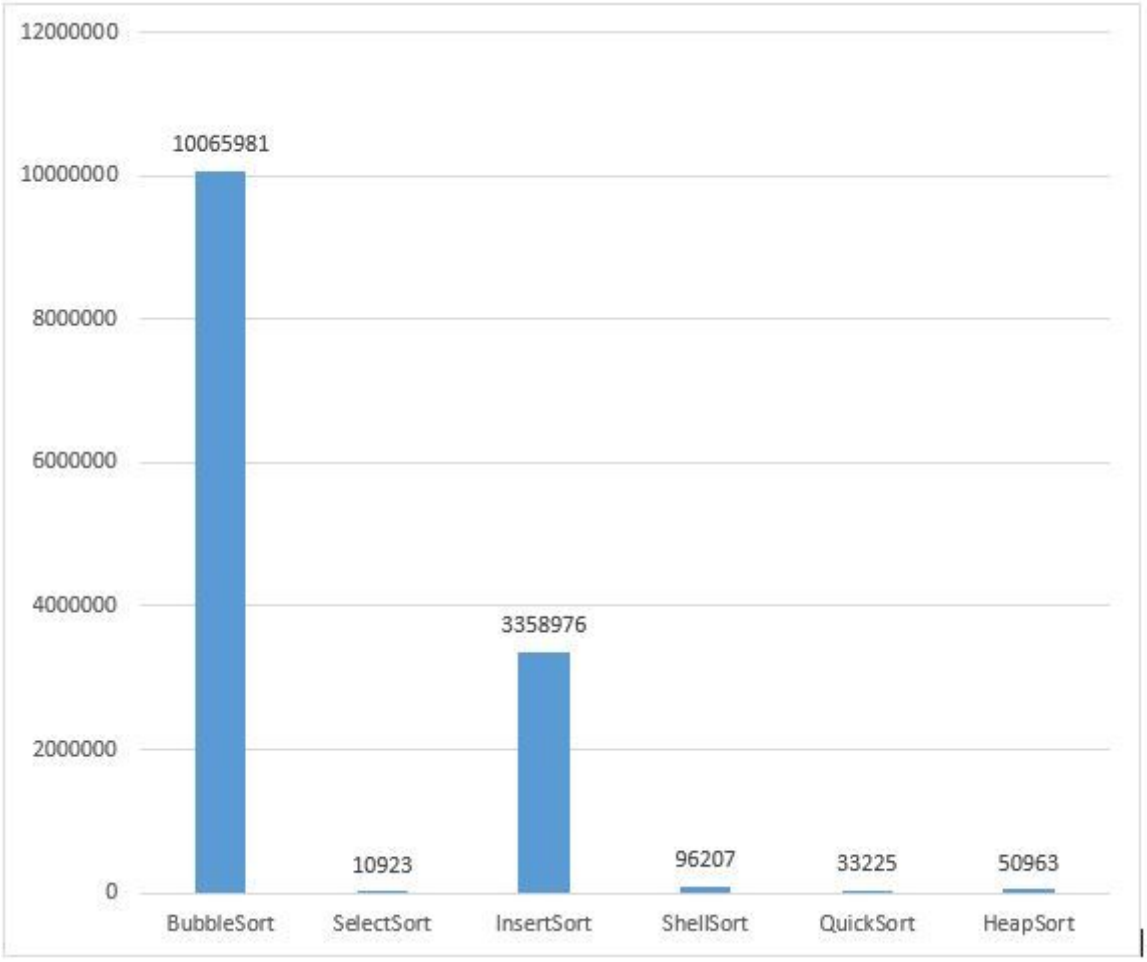
Gráficos de consumo

Para construir o gráfico, pegamos como cenário, o cenário 6.

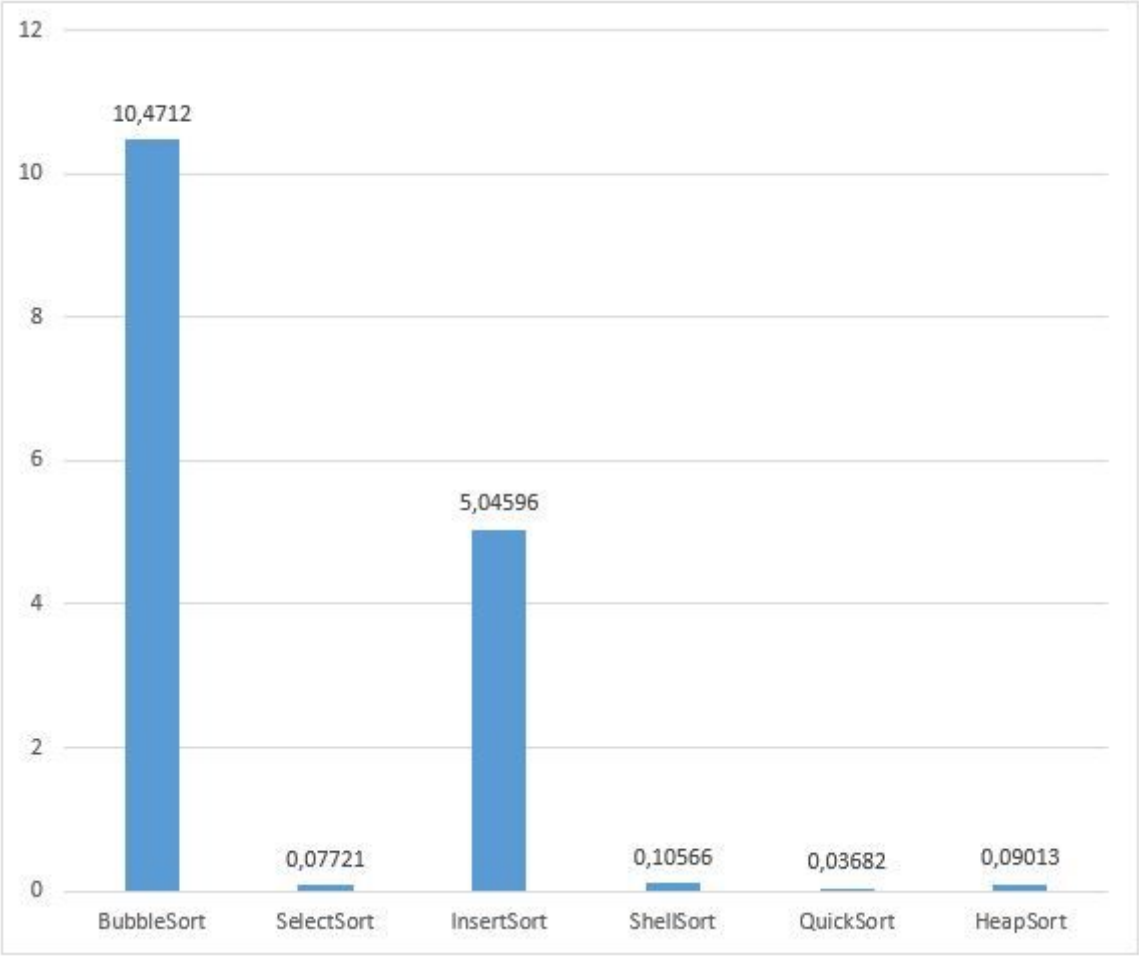
Números de Comparações:



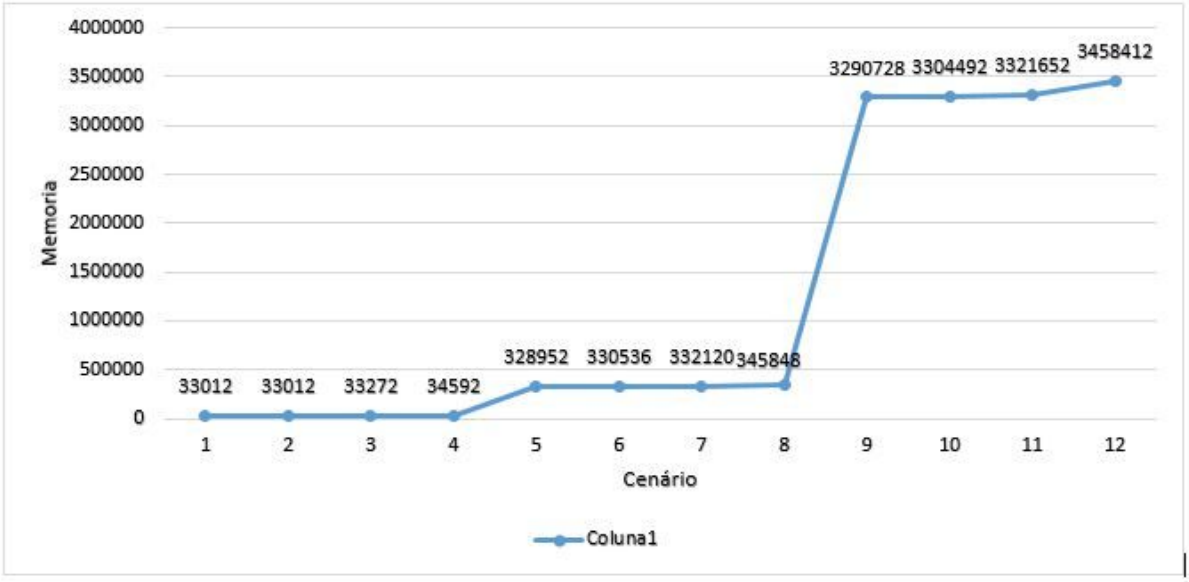
Número de Movimentações:



Tempo de Execução em segundos:



Uso de Memória em cada cenário em Kbts



Métodos implementados

Para a edição do código os três integrantes do grupo usaram o editor de texto Atom. A integração do código foi feita através do GitHub e compilamos através do Clang e GCC no terminal do linux.

Após certificarmos que estávamos na pasta do trabalho que contém os arquivos ".c", a pasta Sources , o seguinte comando era executado no terminal:

clang main.c -o exec ItemMatriz.c voo.c ListadeVoos.c MatrizVoos.c menu.c VetorMatriz.c

E para executar:

./exec

Início da execução do programa:

Após entrar no programa, um menu mostra as opções de entrada possíveis para o usuário:

1. Modo automático
2. Arquivo
0. Sair

Utilização do modo automático:

No modo automático, o usuário precisa entrar somente com o cenário desejado, o resto da execução é feita pelo próprio algoritmo. Após todo o preenchimento, o usuário deverá escolher um dos algoritmos de ordenação para poder ordenar seu vetor.

Modo arquivo:

No arquivo, a única entrada digitada pelo usuário é qual dos arquivo ele deseja entrar. A execução do programa é feita então seguindo os comandos digitados no arquivo.txt indicado pelo usuário.

Conclusão:

O trabalho prático 03, foi muito útil para o aprendizado mais aprofundado acerca de algoritmos de ordenação e como um cenário caótico, com milhares de variáveis pode ser difícil de ser rodado até em ótimos computadores.

Dedicamos também um agradecimento especial para nossos monitores, Angelo e Kayque, além de nossa professora Thais.