

# UNIVERSIDADE PAULISTA

## GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

### ESTRUTURA DE DADOS

**TEMA:** Desenvolvimento de sistema para análise de performance de algoritmos de ordenação de dados.

**ASSUNTO:** "Estudo comparativo entre os diversos métodos de algoritmos de ordenação".

**Alunos:** WAGNER FERREIRA DE LIRA - R.A.: D130BC-0  
ARNON DAMASCENO NEVES DE SOUZA - R.A.: N1047H-2  
KAMILA DE FATIMA S OLIVEIRA - R.A.: D042EC-5  
GABRIEL DO AMARAL VAZQUEZ DIAS – R.A.:D06FDA-5  
VINICIUS DINIZ CARLOS PRUDENCI – R.A.:N866CE-2

## **INDICE**

ATIVIDADES DE PRATICAS SUPERVISIONADAS (APS).....	4
OBJETIVO DO TRABALHO .....	4
INTRODUÇÃO.....	4
Informação e seu significado.....	4
Informação na Ciência da Computação.....	5
Bubblesort.....	5
Selectionsort.....	6
Insertionsort.....	7
Quicksort.....	7
Shellsort.....	8
Referencial Teórico.....	10
Análise dos métodos de ordenação.....	11
DESENVOLVIMENTO.....	12
RESULTADOS E DISCUSSÃO.....	25
-Gráficos sobre o rendimento do programa.....	25
DOCUMENTO DO PROGRAMA (MANUAL DO USUÁRIO).....	82
CONSIDERAÇÕES FINAIS.....	88
PROGRAMA EXECUTÁVEL E CÓDIGO FONTE (COMENTADO).....	90

## **ATIVIDADES DE PRATICAS SUPERVISIONADAS (APS)**

### **OBJETIVO DO TRABALHO**

O principal objetivo do presente trabalho é desenvolver um programa que use e integre pelo menos três tipos de métodos de ordenação, sendo que o método quicksort é obrigatório. Através do desenvolvimento deste programa conhecer os métodos de ordenação existentes e saber sobre suas aplicações, suas vantagens e desvantagens. Saber também qual é a importância desses algoritmos para ciência da computação.

### **INTRODUÇÃO:**

Um computador é uma máquina que manipula informações. O estudo da ciência da computação inclui o exame da organização, manipulação e utilização destas informações num computador. Consequentemente, é muito importante para um estudante da ciência da computação entender os conceitos de organização e manipulação de informações para continuar o estudo do campo. (A.Tenenbaum, 1995)

### **Informação e seu significado:**

A informação é um conjunto organizado de dados, que constitui uma mensagem sobre um determinado fenômeno ou evento. A informação permite resolver problemas e tomar decisões, tendo em conta que o seu uso racional é a base do conhecimento.

Como tal, outra perspectiva indica-nos que a informação é um fenômeno que confere significado ou sentido às coisas, já que através de códigos e de conjuntos de dados, forma os modelos do pensamento humano.

Existem diversas espécies que comunicam entre si através da transmissão de informação para a sua sobrevivência; a diferença para os seres humanos reside na capacidade de criar códigos e símbolos com significados complexos, que conformam a linguagem comum para o convívio em sociedade.

Os dados são percebidos através dos sentidos e, uma vez integrados, acabam por gerar a informação necessária para produzir o conhecimento. Considera-se que a sabedoria é a capacidade para julgar corretamente quando, como, onde e com que objetivo se aplica o conhecimento adquirido.

## **Informação na Ciência da Computação:**

Em ciência da computação a informação é representada por bits (digital binary), com isso podemos avaliar quantidade de informações, cujo o valor compreende uma entre duas possibilidades mutuamente exclusivas. A comunicação de idéias entre pessoas e computadores é feita através de linguagens de programação, porém, as linguagens de programação possuem um domínio de expressão mais reduzido do que o das linguagens naturais e isso facilita a comunicação de ideias computacionais.

**"Conceito de Ordenação:** *Tornar mais simples, rápida e viável a recuperação de uma determinada informação, num conjunto grande de informações".*

Pela definição de ordenação dá para notar a importância dos algoritmos de ordenação, visto que hoje vivemos na era da informação; do compartilhamento de informações, da velocidade de processamento dessas informações para a tomada de decisão. De forma que se torne mais rápida e assertiva possível para criar vantagens competitivas para as empresas. Hoje os mais conhecidos algoritmos de ordenação são: *Bubblesort*, *Insertionsort*, *Selectionsort*, *Quicksort*, *Shellsort*, *Mergesort* e *Heapsort*.

Destes falaremos somente das seis técnicas escolhidas para o presente trabalho que são: *Bubblesort*, *Insertionsort*, *Selectionsort*, *Quicksort*, *Shellsort* e *Mergesort*.

### **Bubblesort**

Algoritmo posicional comparativo:

Seleciona uma posição, compara a sequência a partir desta posição, caso a comparação não esteja ordenada, realiza a troca dos elementos. Realiza a mesma operação recursivamente em todas as posições. Este sistema pode ser melhor explicado da seguinte maneira, imaginemos um conjunto de elementos desordenados, este método selecionaria cada uma das posições e verificaria nas posições subsequentes a esta se existe um elemento que deveria estar na posição selecionada. Essa verificação é feita através da comparação do valor da posição selecionada com o valor da posição que ele está percorrendo, caso o valor do referente à posição do percurso esteja desordenado com o valor selecionado, ele realiza a troca dos mesmos

### **Utilização:**

Este algoritmo é desconsiderado em qualquer aplicação de ordenação atual, mas por ser o método mais simples de ser implementado ainda é estudado no meio acadêmico como aprendizado de ordenação.

## **Selectionsort**

Algoritmo incremental comparativo de ordenação:

Percorre a sequência restante em busca do próximo valor, colocando-o na próxima posição. Esta interação é realizada até que toda a sequência tenha sido percorrida. Este sistema de ordenação realiza a busca pelo valor seguinte entre os elementos ainda não verificados, utiliza então este valor para ser o próximo valor a ser colocado na parte já ordenada da sequência.

### **Utilização:**

Este é um algoritmo muito simples de implementação que realiza poucas trocas, o que lhe garante utilização em ambientes em que a troca é muito custosa. Em contrapartida, realiza muitas comparações, média de  $n \log n$ . Por ser um método comparativo incremental o fato dos elementos já estarem ordenados não o modifica em nada, pois

ele continua realizando  $n \cdot n$  comparações. O seu sistema não é estável, pois ele pode realizar troca de elementos de igual valor.

## **Insertionsort**

Algoritmo incremental de ordenação:

Insere cada elemento do vetor, na sua posição correta em uma subsequência ordenada de elementos, de modo a mantê-la ordenada. Esta forma de ordenação seleciona cada um dos elementos de uma sequência e os atribui uma posição relativa a seu valor, de acordo com a comparação com os elementos já ordenados da mesma sequência.

### **Utilização:**

Este é um algoritmo muito interessante quando os elementos já estão ordenados e deseja-se apenas inserir mais um elemento nesse conjunto de dados ordenados, ou quando se tem um conjunto que se imagina estar praticamente ordenado. O seu pior caso de uso é quando os elementos estão em ordem inversa.

## **Quicksort**

Método de divisão e conquista que concentra o trabalho na divisão e faz a conquista imediata, através deste método a sequência é dividida através de um de seus valores que é considerado mediano, a partir deste valor são ordenados os elementos restantes em seu lado esquerdo, caso menores que este, ou lado direito recursivamente. Este método seleciona um dos valores da sequência, este número então é considerado um pivô, e é comparado com os demais valores da sequência ainda não ordenados que são alocados em sua esquerda caso sejam menores e a sua direita caso sejam maiores, disto restam duas sequências uma a direita e outra a esquerda do pivô, em cada uma destas é realizado o mesmo método até que reste apenas um elemento nas sequências, neste instante temos a sequência ordenada.

### **Utilização:**

Este é um sistema de ordenação bem eficiente, na prática ele efetua uma média de  $n \log n$  comparações. Para termos uma ordenação boa, a escolha do pivô é a chave, então a escolha pode ser feita através da mediana de três elementos da sequência. Por exemplo, de uma sequência pegamos o primeiro valor, o valor do meio e o último, fazemos uma comparação entre eles e verificamos qual é o valor médio dentre estes três, a partir disso utilizamos este valor médio como pivô.

### **Shellsort**

Método de ordenação por inserção através de decrementos, esta é uma extensão do método de inserção que pode realizar troca entre elementos que estão distantes um do outro sem precisar realizar a troca adjacente de cada um dos elementos entre estes. O método de inserção realiza a comparação e a troca adjacente dos elementos até que encontre o ponto de inserção do elemento que ele pretende inserir. Caso este elemento esteja do outro lado da sequência de elementos, ele irá realizar  $n-1$  comparações e movimentações.

### **Utilização:**

Por ter uma implementação pequena que requer poucas linhas de códigos e por ser um método eficiente, pode ser utilizado em sistemas que não dispõe de muitos recursos de memória. O seu tempo de execução é sensível à ordem inicial do programa, o que lhe garante um bom uso em sequências já ordenadas. (herdado do método de inserção) O seu sistema não é estável, pois ele pode realizar troca de elementos de igual valor.

### **Mergesort**

Método de divisão e conquista que concentra o trabalho na conquista, neste modelo de ordenação a sequência é dividida em sequências menores com metade do tamanho da original que são ordenados recursivamente. Este método divide a sequência em sequências com metade de seu tamanho ordenando-as enquanto divide, e realiza sucessivas divisões de cada uma das sequências conseguidas, ao final destas a sequência se torna ordenada pois foi-se ordenando-a um a um os elementos.

**Utilização:**

No pior caso de uso, este método é mais eficiente que o quicksort, apesar de que na prática o pior caso não é o que acontece em todas as vezes, tornando assim o quicksort mais eficiente na prática.



## **Referencial Teórico**

Estrutura de dados usando C – (A.Tenenbaum, Y. Langsam, M.J.Augenstein, 1995)

Linguagens de programação – (A.B.Tucker, R.E.Noonan, 2008)

Fundamentos de Estrutura de dados – (Horowitz, E.Sahni, 1987)

## **Análise dos métodos de ordenação**

Foram realizados testes com seis métodos de ordenação, para cada método foi testado um tamanho diferente de amostra(vetor); 1.000, 2.000, 3.000,4.000, 5.000, 6.000 e 6.010.

Exceto para os métodos de ordenação que são extremamente rápidos (quicksort, mergesort e insertion sort) que foram realizadas com amostras de dados muito maiores: 35.000, 40.000, 45.000, 50.000, 60.000, 80.000 e 100.000.

Estes foram testados de três formas diferentes: vetor ordenado, vetor parcialmente ordenado (semi-ordenado) e vetor desordenado.

Os resultados serão apresentados pelo tipo de amostra e respectivamente o método de ordenação.

### **Os métodos de ordenação utilizados foram:**

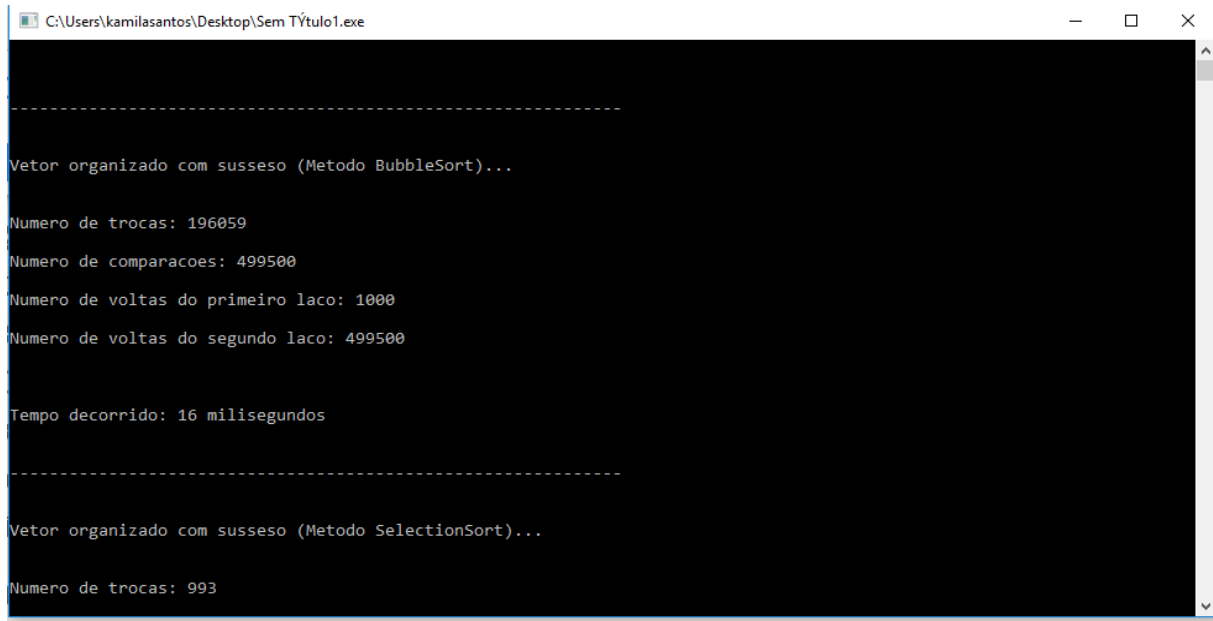
- Bubblesort
- Quicksort
- Insertionsort
- Mergesort
- Selectionsort
- Shellsort

## DESENVOLVIMENTO

Testes gerados com o vetor de forma aleatória.

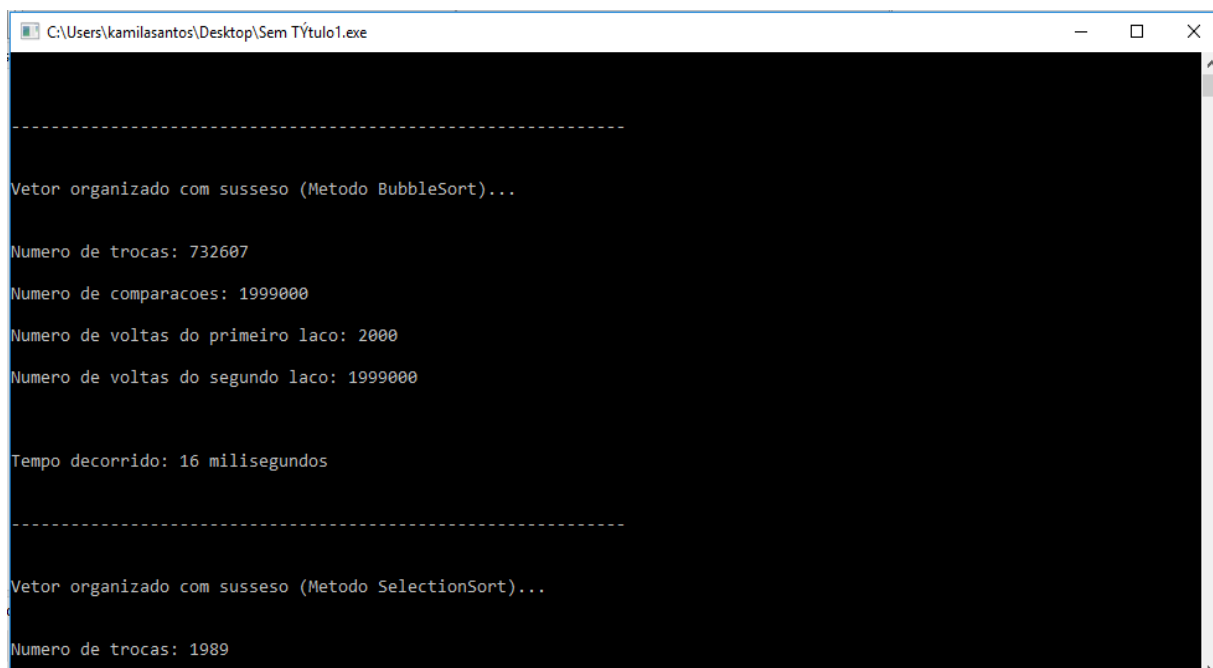
-Evidências

-Bubblesort



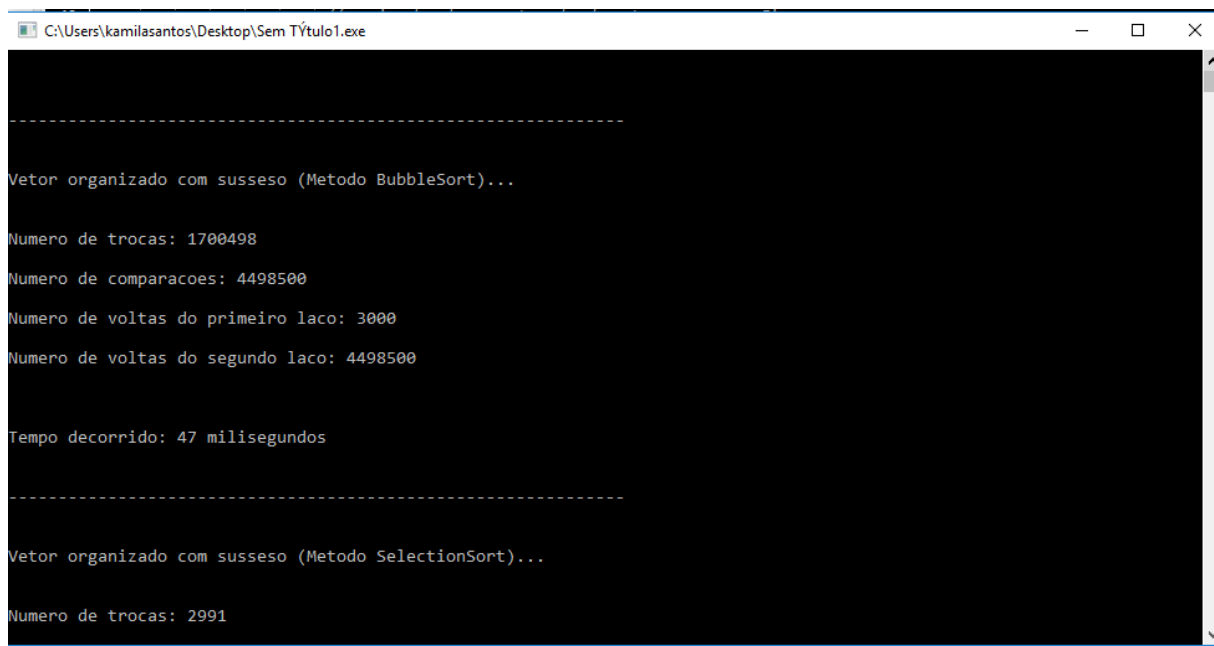
```
-----  
Vetor organizado com sucesso (Metodo BubbleSort)...  
  
Numero de trocas: 196059  
Numero de comparacoes: 499500  
Numero de voltas do primeiro laco: 1000  
Numero de voltas do segundo laco: 499500  
  
Tempo decorrido: 16 milisegundos  
  
-----  
Vetor organizado com sucesso (Metodo SelectionSort)...  
  
Numero de trocas: 993
```

Figura 01- Evidências do teste com 1000 posições.



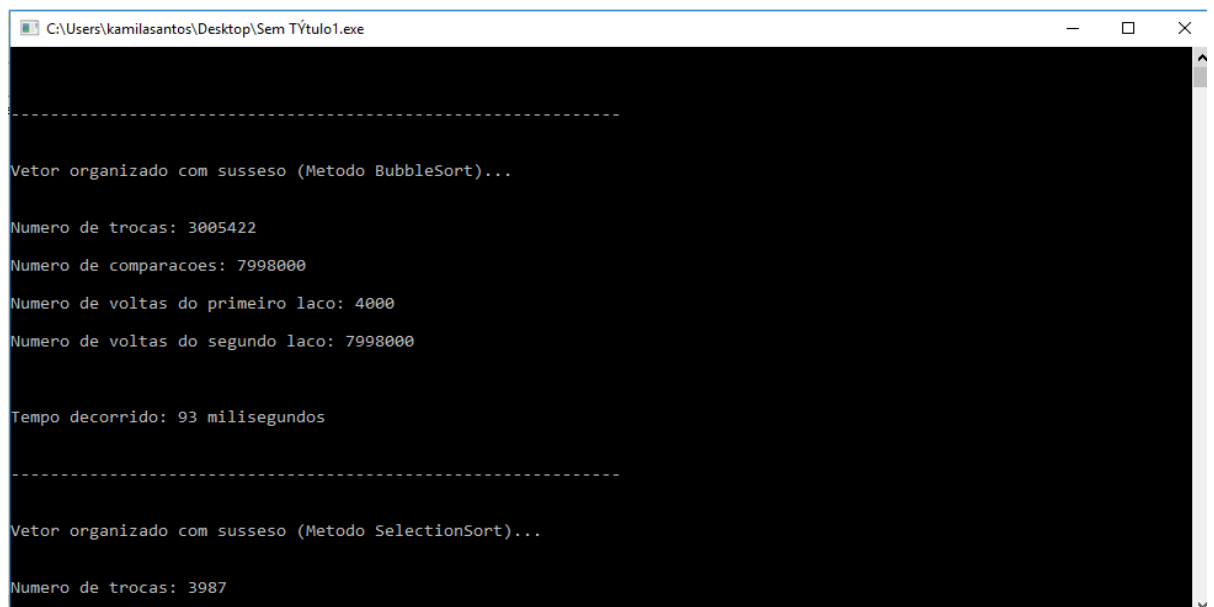
```
-----  
Vetor organizado com sucesso (Metodo BubbleSort)...  
  
Numero de trocas: 732607  
Numero de comparacoes: 1999000  
Numero de voltas do primeiro laco: 2000  
Numero de voltas do segundo laco: 1999000  
  
Tempo decorrido: 16 milisegundos  
  
-----  
Vetor organizado com sucesso (Metodo SelectionSort)...  
  
Numero de trocas: 1989
```

Figura 02- Evidências do teste com vetor de 2000 posições



```
-----  
Vetor organizado com sucesso (Metodo BubbleSort)...  
  
Numero de trocas: 1700498  
Numero de comparacoes: 4498500  
Numero de voltas do primeiro laco: 3000  
Numero de voltas do segundo laco: 4498500  
  
Tempo decorrido: 47 milisegundos  
-----  
  
Vetor organizado com sucesso (Metodo SelectionSort)...  
  
Numero de trocas: 2991
```

Figura 03- Evidências do teste com vetor de 3000 posições.



```
-----  
Vetor organizado com sucesso (Metodo BubbleSort)...  
  
Numero de trocas: 3005422  
Numero de comparacoes: 7998000  
Numero de voltas do primeiro laco: 4000  
Numero de voltas do segundo laco: 7998000  
  
Tempo decorrido: 93 milisegundos  
-----  
  
Vetor organizado com sucesso (Metodo SelectionSort)...  
  
Numero de trocas: 3987
```

Figura 04- Evidência do teste com vetor de 4000 posições.

```
C:\Users\kamilasantos\Desktop\Sem T tulo1.exe

-----
Vetor organizado com sucesso (Metodo BubbleSort)...

Numero de trocas: 4692287
Numero de comparacoes: 12497500
Numero de voltas do primeiro laco: 5000
Numero de voltas do segundo laco: 12497500

Tempo decorrido: 141 milisegundos

-----
Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 4988
```

Figura 05-Evid ncia do teste com vetor de 5000 posi  es.

```
C:\Users\kamilasantos\Desktop\Sem T tulo1.exe

-----
Vetor organizado com sucesso (Metodo BubbleSort)...

Numero de trocas: 6778308
Numero de comparacoes: 17997000
Numero de voltas do primeiro laco: 6000
Numero de voltas do segundo laco: 17997000

Tempo decorrido: 203 milisegundos

-----
Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 5988
```

Figura 06- Evid ncias do teste com vetor de 6000 posi  es.

-Quick, insertion e merge sort

```
C:\Users\kamilasantos\Downloads\aps.exe

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 1781 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 15 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 07- Evidência do teste com vetor de 35000 posições

```
C:\Users\kamilasantos\Downloads\aps.exe

Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 2532 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 16 milisegundos

Pressione qualquer tecla para continuar. . .
```

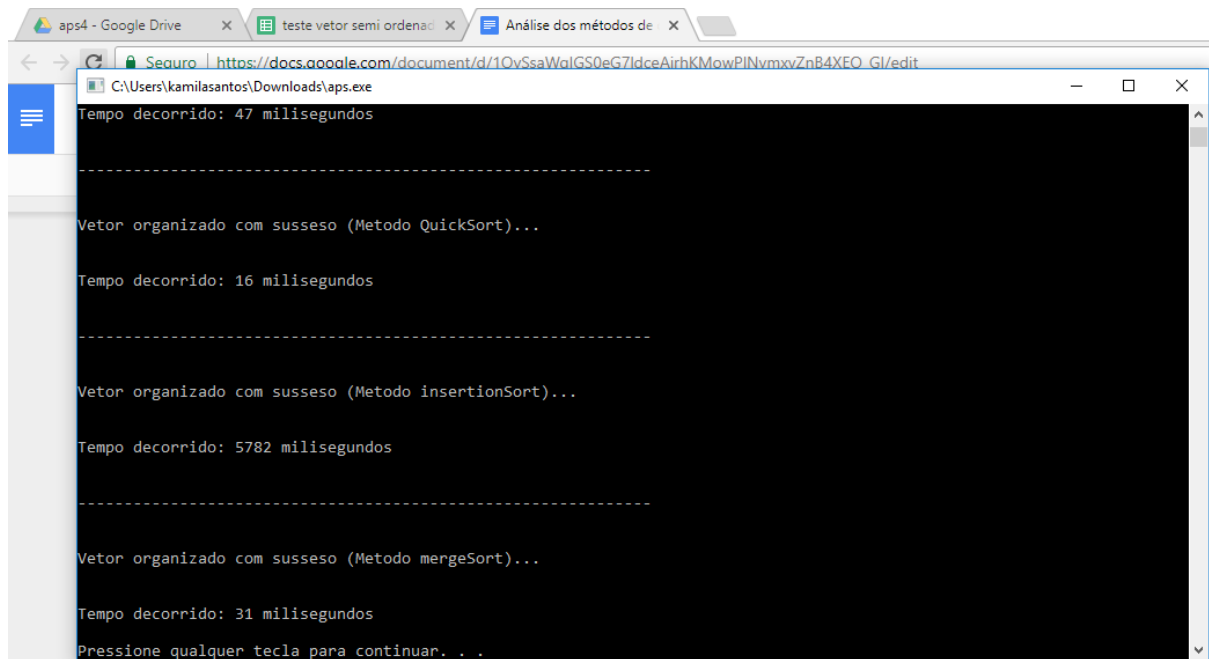
Figura 08- Evidências do teste com vetor de 40000 posições.

```
C:\Users\kamilasantos\Downloads\aps.exe
Tempo decorrido: 63 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 16 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 2891 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 15 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 09- Evidência do teste com vetor de 45000 posições.

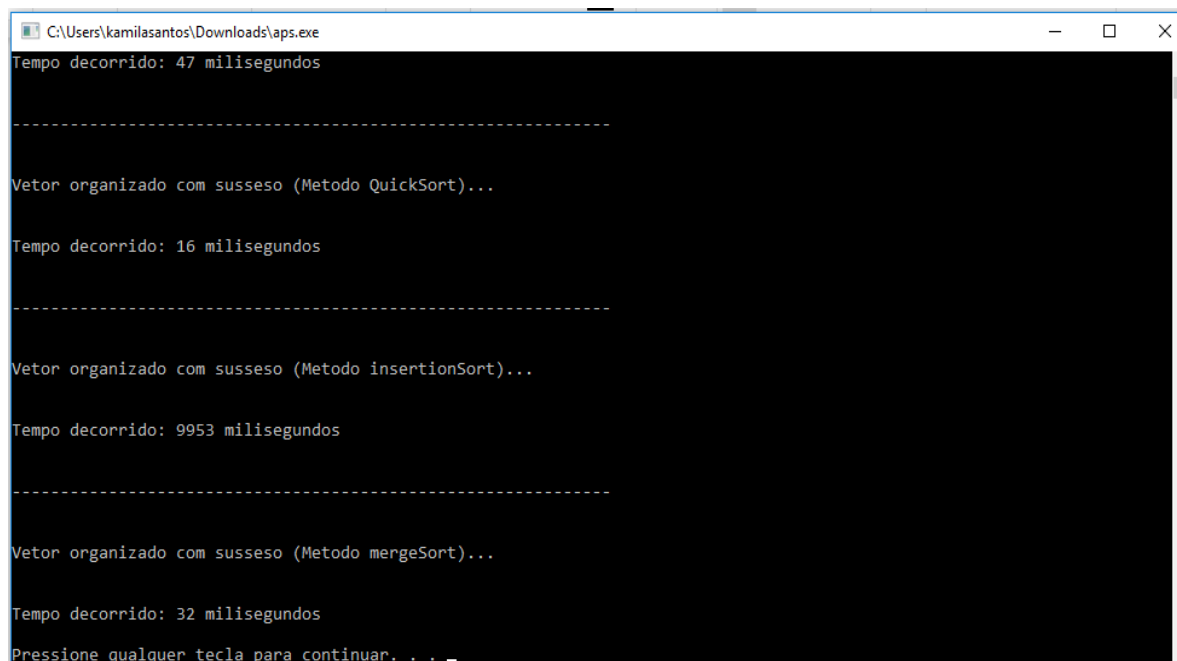
```
C:\Users\kamilasantos\Downloads\aps.exe
Tempo decorrido: 47 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 15 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 4000 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 32 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 10- Evidências do teste com vetor de 50000 posições.



```
aps4 - Google Drive X teste vetor semi ordenado X Análise dos métodos de
C:\Users\kamilasantos\Downloads\aps.exe
Tempo decorrido: 47 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 16 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 5782 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 31 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 11- Evidências do teste com vetor de 60000 posições.



```
C:\Users\kamilasantos\Downloads\aps.exe
Tempo decorrido: 47 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 16 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 9953 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 32 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 12 - Evidência do teste com vetor de 80000 posições.



```
aps4 - Google Drive X teste vetor semi ordenad X Análise dos métodos de ordenação X  
Seguro | https://docs.google.com/document/d/1OySsaWglGS0eG7ldceAjrKMoWpINvmxvZnB4XEO_Gl/edit  
Análise dos métodos de ordenação  
Arquivo C:\Users\kamilasantos\Downloads\aps.exe  
Tempo decorrido: 62 milisegundos  
-----  
Vetor organizado com sucesso (Metodo QuickSort)...  
Tempo decorrido: 31 milisegundos  
-----  
Vetor organizado com sucesso (Metodo insertionSort)...  
Tempo decorrido: 15000 milisegundos  
-----  
Vetor organizado com sucesso (Metodo mergeSort)...  
Tempo decorrido: 31 milisegundos  
Pressione qualquer tecla para continuar. . .
```

Figura 13 - Evidencia do teste com vetor de 100000 posições.

#### -Selection sort

```
Meu Drive - Google Drive X  
C:\Users\kamilasantos\Desktop\Sem Título1.exe  
Tempo decorrido: 16 milisegundos  
-----  
Vetor organizado com sucesso (Metodo SelectionSort)...  
Numero de trocas: 993  
Numero de comparacoes: 500499  
Numero de voltas do primeiro laco: 999  
Numero de voltas do segundo laco: 499500  
Tempo decorrido: 15 milisegundos  
-----  
Vetor organizado com sucesso (Metodo ShellSort)...  
Numero de trocas: 8725
```

Figura 14- Evidência do teste com vetor de 1000 posições.

```
C:\Users\kamilasantos\Desktop\Sem T tulo1.exe

-----

Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 1989
Numero de comparacoes: 2000999
Numero de voltas do primeiro laco: 1999
Numero de voltas do segundo laco: 1999000

Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 22620
Numero de comparacoes: 34984
```

Figura 15- Evidencia do teste com vetor de 2000 posi  es.

```
C:\Users\kamilasantos\Desktop\Sem T tulo1.exe

-----

Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 2991
Numero de comparacoes: 4501499
Numero de voltas do primeiro laco: 2999
Numero de voltas do segundo laco: 4498500

Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 35232
Numero de comparacoes: 54596
```

Figura 16 - Evid ncia do teste com vetor de 3000 posi  es.

```
C:\Users\kamilasantos\Desktop\Sem Título1.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Desktop\Sem Título1.exe
Tempo decorrido: 93 milisegundos
-----
Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 3987
Numero de comparacoes: 8001999
Numero de voltas do primeiro laco: 3999
Numero de voltas do segundo laco: 7998000

Tempo decorrido: 47 milisegundos
-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 49247
```

Figura 17- Evidência do teste com vetor de 4000 posições.

```
C:\Users\kamilasantos\Desktop\Sem Título1.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Desktop\Sem Título1.exe
-----
Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 4988
Numero de comparacoes: 12502499
Numero de voltas do primeiro laco: 4999
Numero de voltas do segundo laco: 12497500

Tempo decorrido: 47 milisegundos
-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 70157
Numero de comparacoes: 105241
```

Figura 18- Evidência do teste com vetor de 5000 posições.

```
C:\Users\kamilasantos\Desktop\Sem Título1.exe
Tempo decorrido: 203 milisegundos

-----

Vetor organizado com sucesso (Metodo SelectionSort)...

Numero de trocas: 5988
Numero de comparacoes: 18002999
Numero de voltas do primeiro laco: 5999
Numero de voltas do segundo laco: 17997000

Tempo decorrido: 94 milisegundos

-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 81909
```

Figura 19- Evidência do teste com vetor de 6000 posições.

#### -Shellsort

```
C:\Users\kamilasantos\Desktop\Sem Título1.exe
Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 8464
Numero de comparacoes: 13921
Numero de Quebras do vetor: 6
Numero de voltas do primeiro laco: 5457
Numero de Trocas no pivo: 5457

Tempo decorrido: 0 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...
```

Figura 20- Evidência do teste com vetor de 1000 posições.

```
Selecionar C:\Users\kamilasantos\Desktop\Sem Título1.exe

-----
Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 22620
Numero de comparacoes: 34984
Numero de Quebras do vetor: 7
Numero de voltas do primeiro laço: 12364
Numero de Trocas no pivo: 12364

Tempo decorrido: 15 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 0 milisegundos
```

Figura 21- Evidência do teste com vetor de 2000 posições.

```
C:\Users\kamilasantos\Desktop\Sem Título1.exe

-----
Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 35232
Numero de comparacoes: 54596
Numero de Quebras do vetor: 7
Numero de voltas do primeiro laço: 19364
Numero de Trocas no pivo: 19364

Tempo decorrido: 0 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 0 milisegundos
```

Figura 22- Evidência do teste com vetor de 3000 posições.

```
C:\Users\kamilasantos\Desktop\Sem Título1.cpp - [Executing] - Dev-C++ 5.11
Arqu
C:\Users\kamilasantos\Desktop\Sem Título1.exe
Tempo decorrido: 47 milisegundos
-----
Vetor organizado com sucesso (Metodo ShellSort)...
Numero de trocas: 49247
Numero de comparacoes: 76331
Numero de Quebras do vetor: 8
Numero de voltas do primeiro laco: 27084
Numero de Trocas no pivo: 27084
Tempo decorrido: 16 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
```

Figura 23- Evidência do teste com vetor de 4000 posições

```
C:\Users\kamilasantos\Desktop\Sem Título1.exe
-----
Vetor organizado com sucesso (Metodo ShellSort)...
Numero de trocas: 70157
Numero de comparacoes: 105241
Numero de Quebras do vetor: 8
Numero de voltas do primeiro laco: 35084
Numero de Trocas no pivo: 35084
Tempo decorrido: 31 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 0 milisegundos
```

Figura 24- Evidência do teste com vetor de 5000 posições.

```
C:\Users\kamilasantos\Desktop\Sem T tulo1.exe

-----
Vetor organizado com sucesso (Metodo ShellSort)...

Numero de trocas: 81909
Numero de comparacoes: 124993
Numero de Quebras do vetor: 8
Numero de voltas do primeiro laco: 43084
Numero de Trocas no pivo: 43084

Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 0 milisegundos
```

Figura 25- Evid ncia do teste com vetor de 6000 posi  es.

## RESULTADOS E DISCUSSÃO

-Gráficos sobre o rendimento do programa

-Bubble sort

Entrada	Trocas
1000	196059
2000	732607
3000	1700498
4000	3005422
5000	4692287
6000	6778308

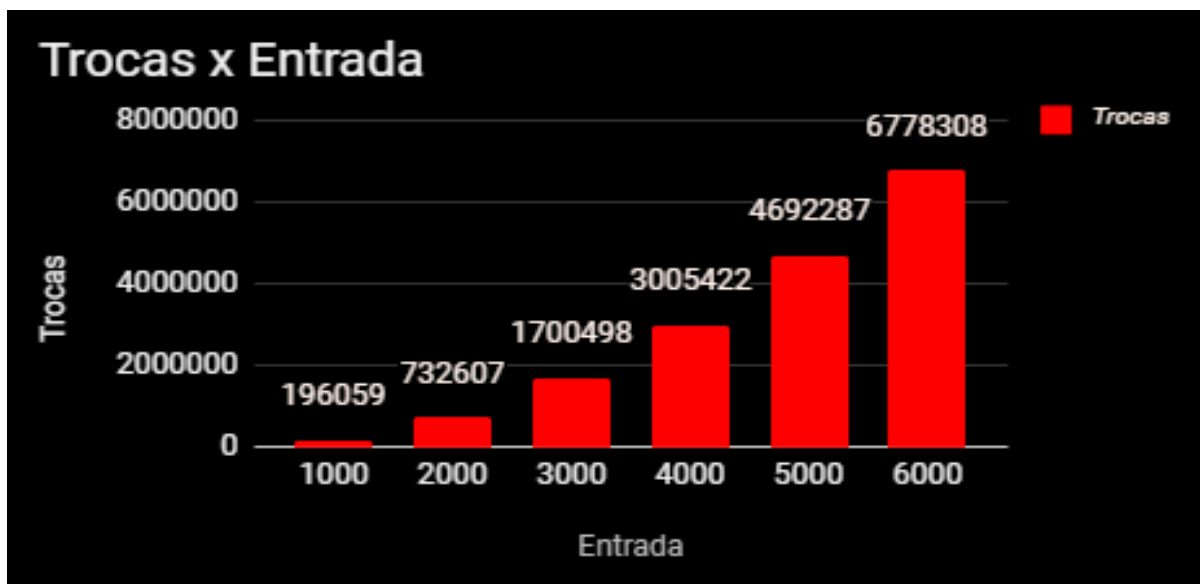


Figura 26- Gráfico com a relação entre o volume de entrada de dados e as trocas realizadas até a ordenação.



Entrada	Comparações
1000	499500
2000	1999000
3000	4498500
4000	7998000
5000	12497500
6000	17997000



Figura 27- Gráfico com a relação entre o volume de entrada de dados e as comparações realizadas até a ordenação.

Entrada	Voltas no primeiro laço
1000	1000
2000	2000
3000	3000
4000	4000
5000	5000
6000	6000



Figura 28- Gráfico com a relação entre o volume de entrada de dados e as voltas efetuadas pelo primeiro laço até a ordenação.

Entrada	voltas no segundo laço
1000	499500
2000	1999000
3000	4498500
4000	7998000
5000	12497500
6000	17997000



Figura 29- Gráfico com a relação entre o volume de entrada de dados e as voltas efetuadas pelo segundo laço até a ordenação.

Entrada	tempo
1000	16
2000	16
3000	47
4000	93
5000	141
6000	203



Figura 30 - Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Quicksort

Entrada	tempo
35000	16
40000	16
45000	16
50000	15
60000	16
80000	16
100000	31

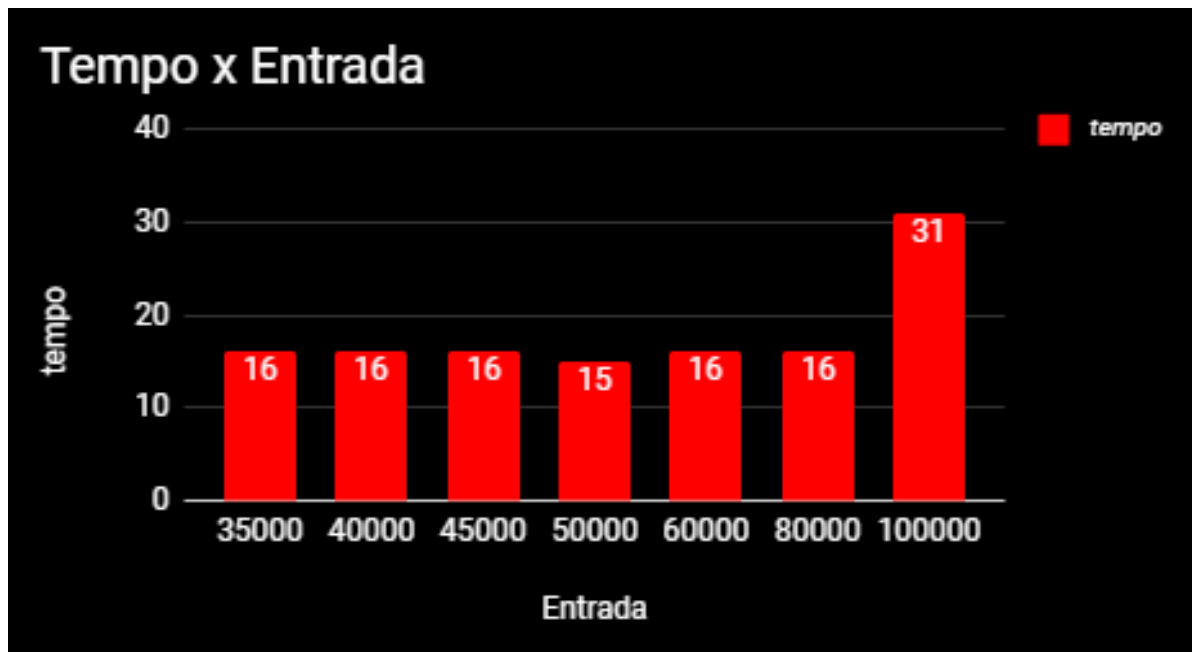


Figura 31- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Insertion sort

Entrada	tempo
35000	1781
40000	2532
45000	2891
50000	4000
60000	5782
80000	9953
100000	15000

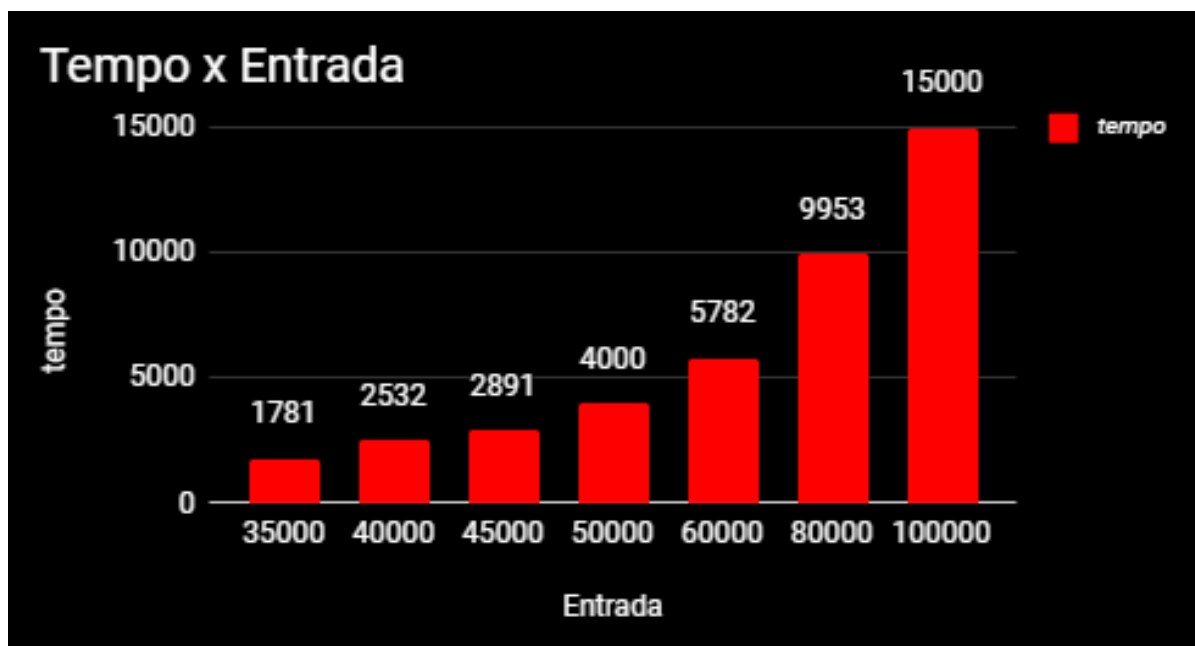


Figura 32- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

Entrada	tempo
35000	15
40000	16
45000	15
50000	32
60000	31
80000	32
100000	31

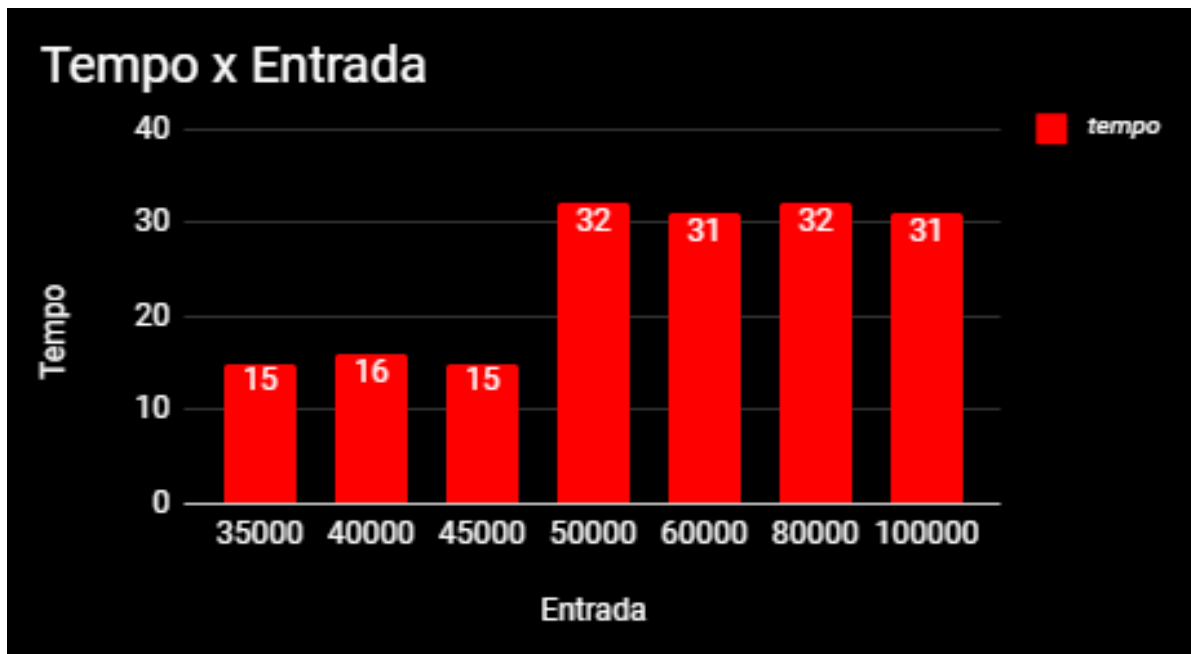


Figura 33- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Selectionsort

Entrada	Trocas
1000	993
2000	1989
3000	2991
4000	3987
5000	4988
6000	5988

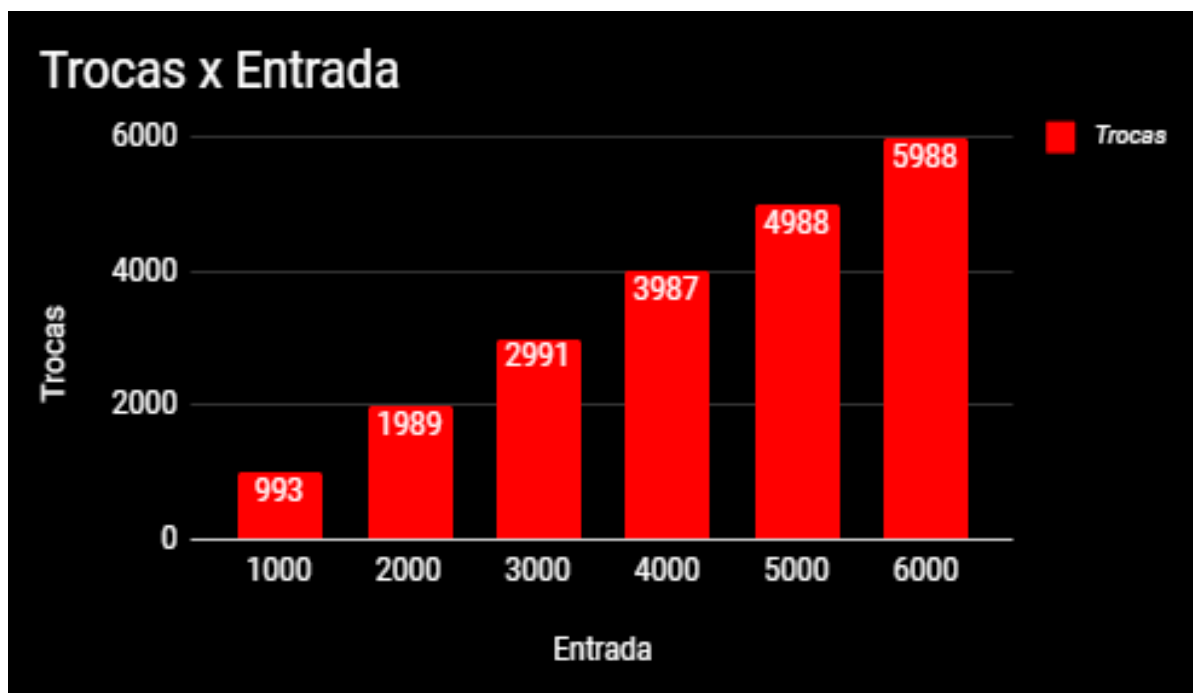


Figura 34- Gráfico com a relação entre o volume de entrada de dados e as trocas efetuadas até a ordenação.

Entrada	Comparações
1000	500499
2000	2000999
3000	4501499
4000	8001999
5000	12502499
6000	18002999



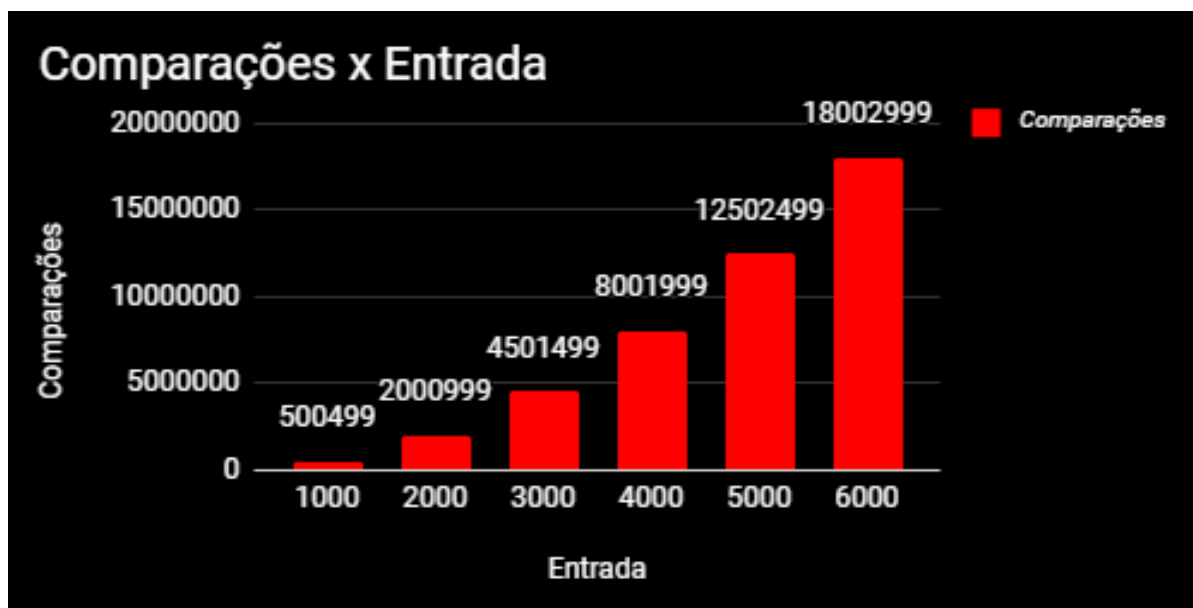


Figura 35- Gráfico com a relação entre o volume de entrada de dados e as comparações realizadas até a ordenação.

Entrada	Voltas no primeiro laço
1000	999
2000	1999
3000	2999
4000	3999
5000	4999
6000	5999



Figura 36- Gráfico com a relação entre o volume de entrada de dados e as voltas efetuadas pelo primeiro laço até a ordenação.

Entrada	voltas no segundo laço
1000	499500
2000	1999000
3000	4498500
4000	7998000
5000	12497500
6000	17997000



Figura 37- Gráfico com a relação entre o volume de entrada de dados e as voltas efetuadas pelo segundo laço até a ordenação.

Entrada	tempo
1000	15
2000	31
3000	31
4000	47
5000	47
6000	94

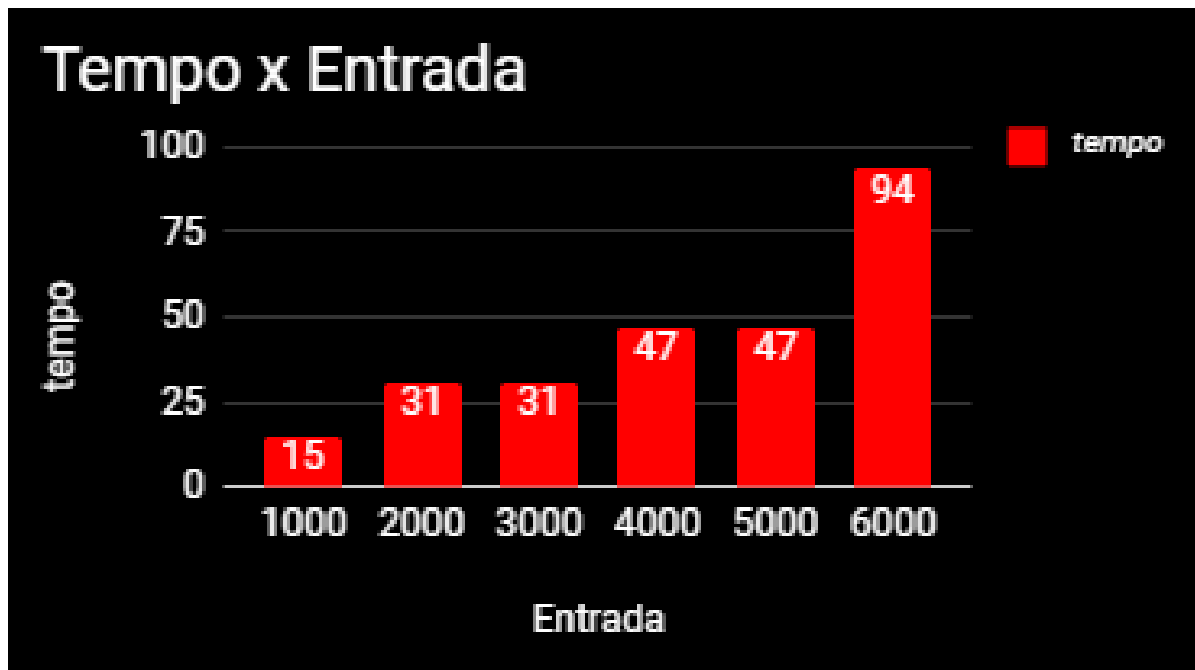


Figura 38- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Shellsort

Entrada	Trocas
1000	8464
2000	22620
3000	35232
4000	49247
5000	70157
6000	81909

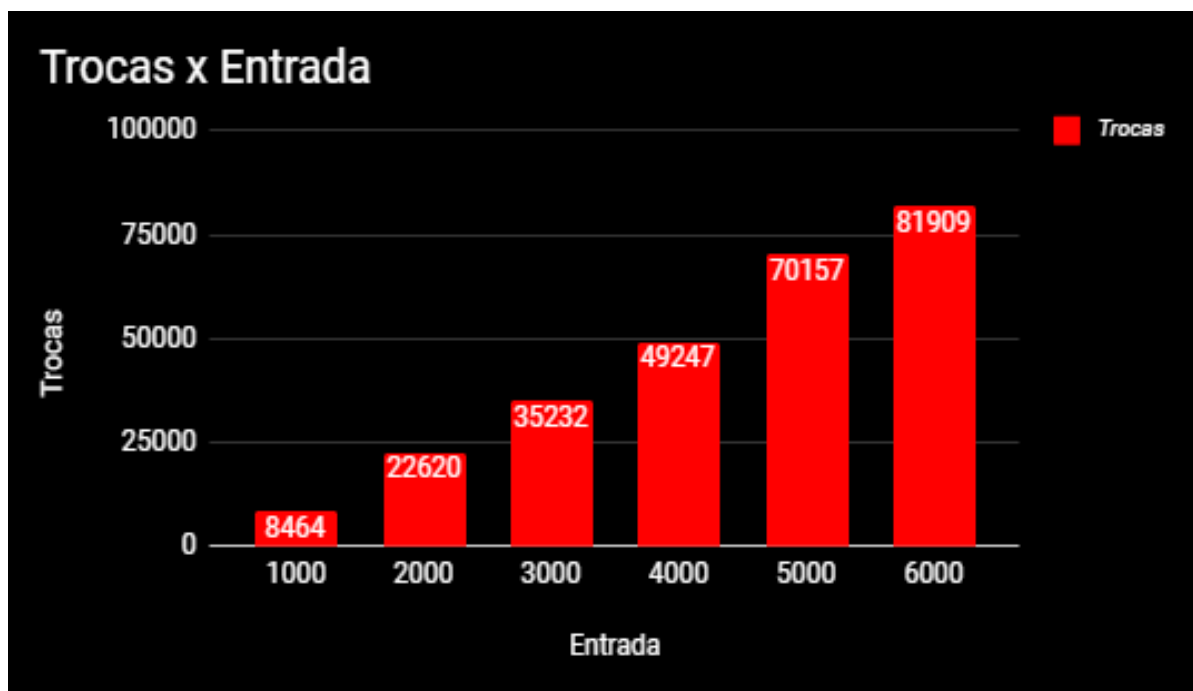


Figura 39- Gráfico com a relação entre o volume de entrada de dados e as trocas efetuadas até a ordenação.

Entrada	Comparações
1000	13921
2000	34984
3000	54596
4000	76331
5000	105241
6000	124993

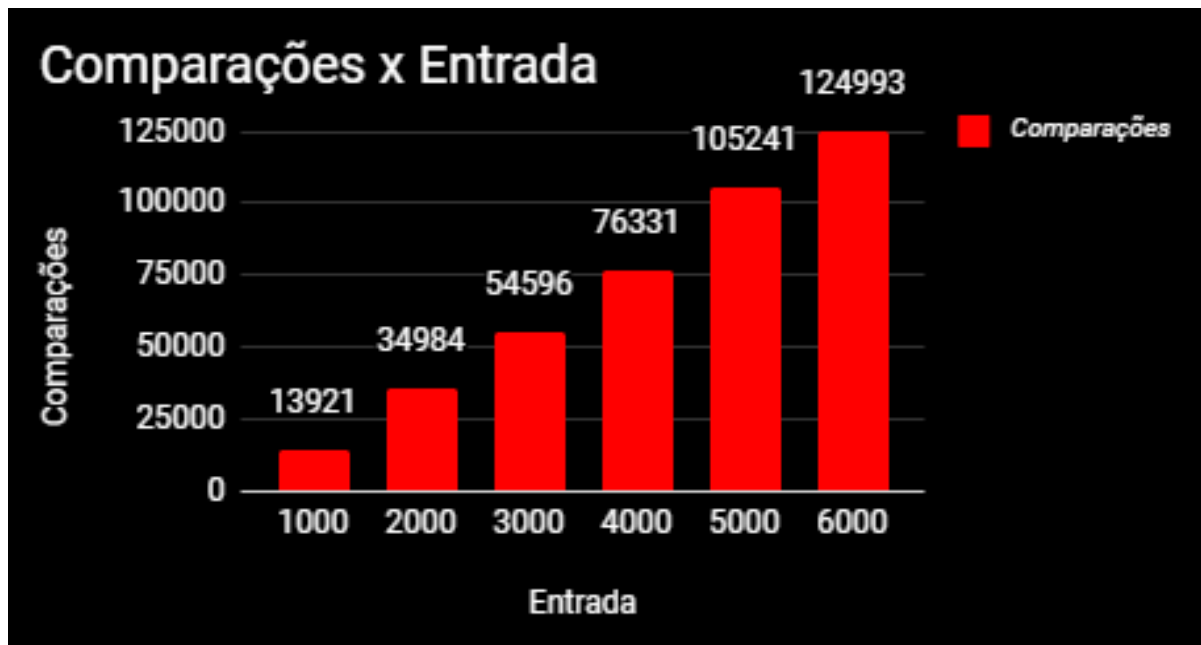


Figura 40- Gráfico com a relação entre o volume de entrada de dados e as comparações efetuadas até a ordenação.

Entrada	quebras do vetor
1000	6
2000	7
3000	7
4000	8
5000	8
6000	8

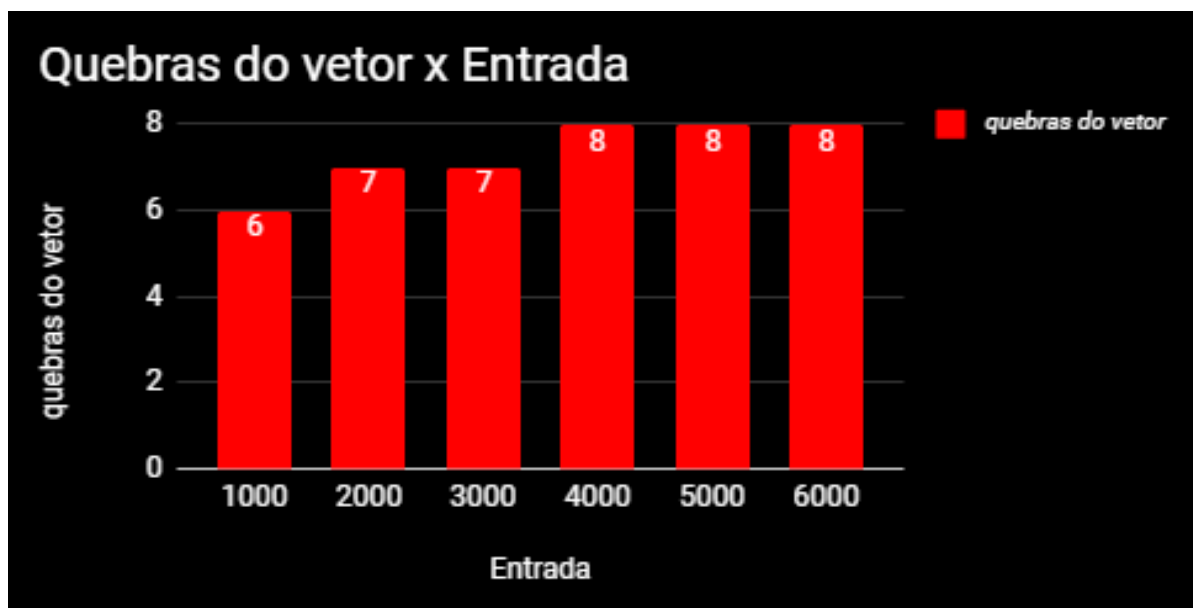


Figura 41- Gráfico com a relação entre o volume de entrada de dados e as quebras realizadas até a ordenação.

Entrada	voltas do primeiro laço
1000	5457
2000	12364
3000	19364
4000	27084
5000	35084
6000	43084



Figura 42- Gráfico com a relação entre o volume de entrada de dados e as voltas efetuadas pelo primeiro laço.

Entrada	trocas no pivo
1000	5457
2000	12364
3000	19364
4000	27084
5000	35084
6000	43084



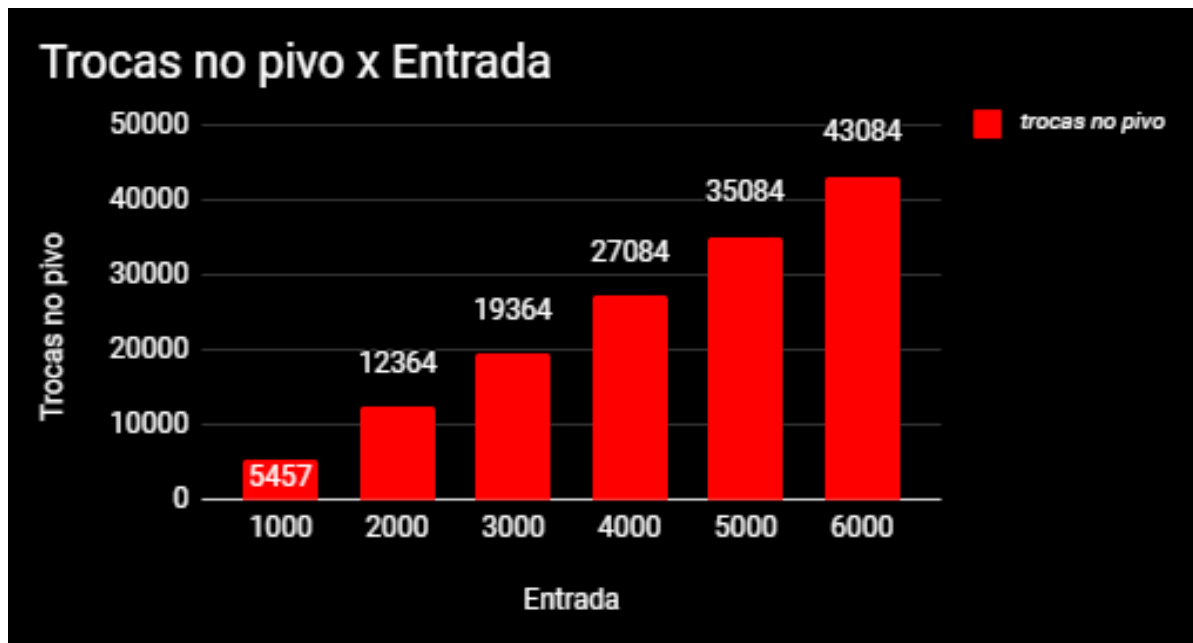


Figura 43- Gráfico com a relação entre o volume de entrada de dados e as trocas efetuadas pelo pivô até a ordenação.

Entrada	tempo
1000	16
2000	15
3000	0
4000	16
5000	31
6000	16

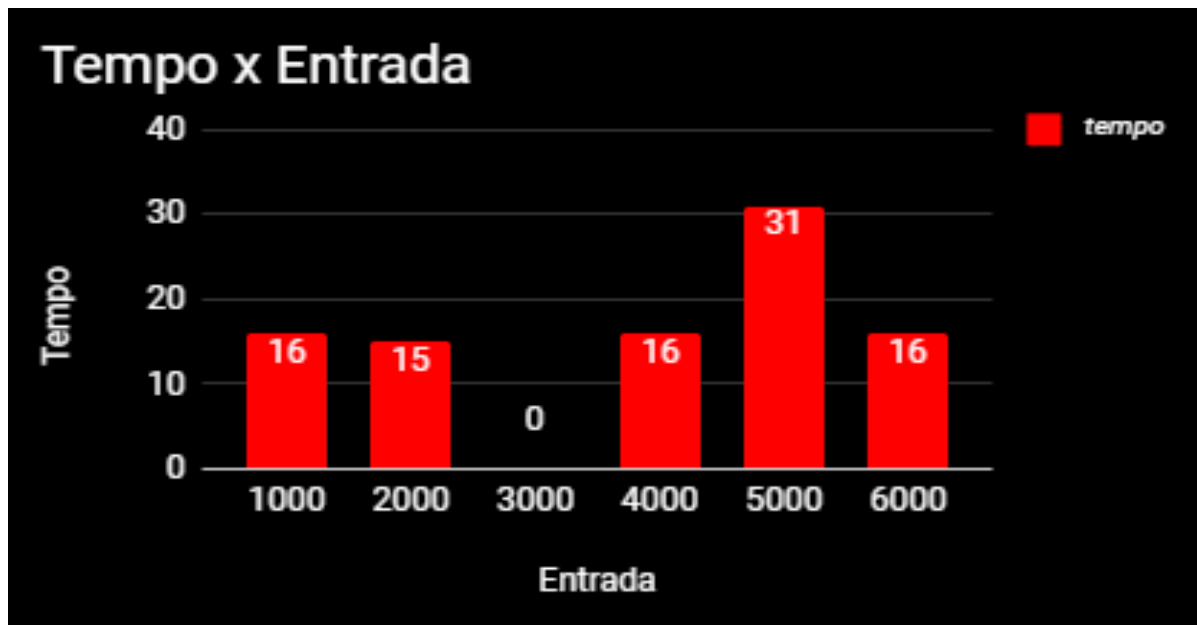


Figura 44- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

Testes com vetor semi ordenado

-Evidências

-Bubble sort

```
C:\Users\kamilasantos\Downloads\aps3.exe

990 || 990 |
991 || 992 |
992 || 993 |
993 || 993 |
994 || 994 |
995 || 995 |
996 || 996 |
997 || 997 |
998 || 998 |
999 || 999 |

Vetor organizado com sucesso...

Tempo decorrido: 15 milisegundosPressione qualquer tecla para continuar. . .
```

Figura 45- Evidência do teste com vetor de 1000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.exe

1988 || 1990 |
1989 || 1990 |
1990 || 1991 |
1991 || 1992 |
1992 || 1993 |
1993 || 1993 |
1994 || 1994 |
1995 || 1996 |
1996 || 1996 |
1997 || 1997 |
1998 || 1998 |
1999 || 1999 |

Vetor organizado com sucesso...

Tempo decorrido: 31 milisegundosPressione qualquer tecla para continuar. . .
```

Figura 46- Evidencia do teste com vetor de 2000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Downloads\aps3.exe
2988    2986
2989    2989
2990    2990
2991    2992
2992    2993
2993    2994
2994    2995
2995    2995
2996    2996
2997    2997
2998    2997
2999    2998

Vetor organizado com sucesso...
Tempo decorrido: 46 milisegundosPressione qualquer tecla para continuar...
```

Figura 47- Evidência do teste com vetor de 3000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.exe
3988    3987
3989    3989
3990    3990
3991    3991
3992    3992
3993    3993
3994    3993
3995    3993
3996    3994
3997    3995
3998    3998
3999    3999

Vetor organizado com sucesso...
Tempo decorrido: 79 milisegundosPressione qualquer tecla para continuar...
```

Figura 48- Evidência do teste com vetor de 4000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.exe
4988      4986      |
4989      ||      4987      |
4990      ||      4988      |
4991      ||      4989      |
4992      ||      4990      |
4993      ||      4990      |
4994      ||      4991      |
4995      ||      4993      |
4996      ||      4996      |
4997      ||      4997      |
4998      ||      4998      |
4999      ||      4998      |

Vetor organizado com sucesso...

Tempo decorrido: 125 milisegundosPressione qualquer tecla para continuar. . .
```

Figura 49- Evidência do teste com vetor de 5000 posições.

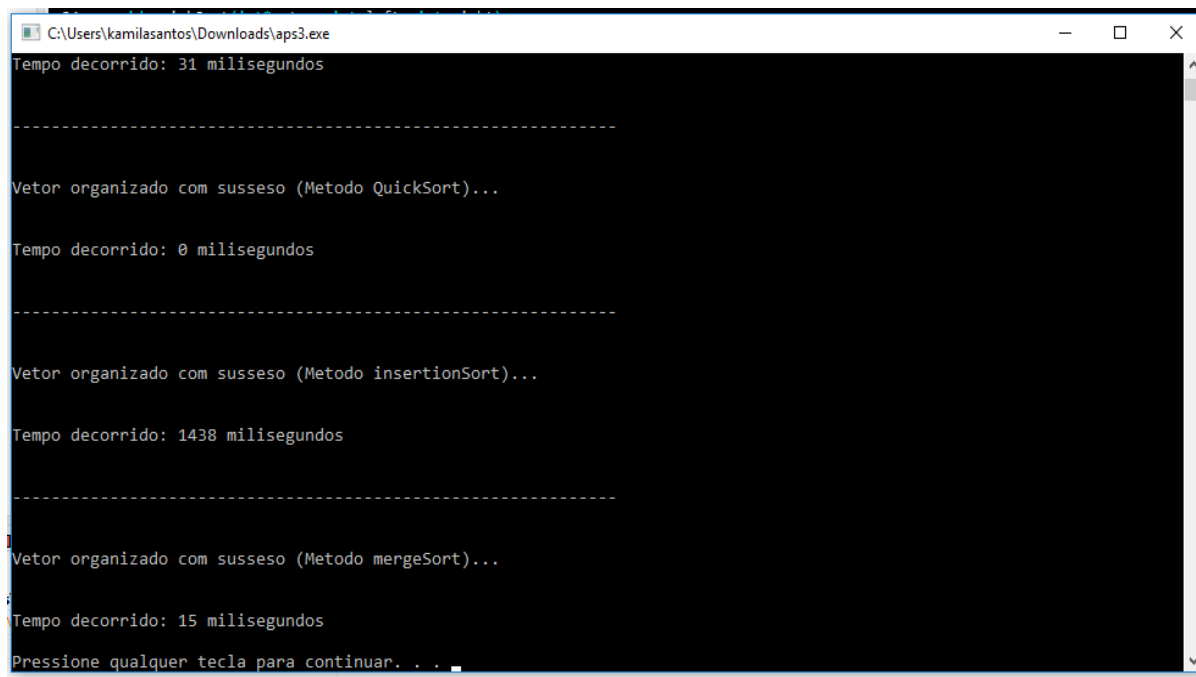
```
C:\Users\kamilasantos\Downloads\aps3.exe
5988      5987      |
5989      ||      5987      |
5990      ||      5990      |
5991      ||      5990      |
5992      ||      5991      |
5993      ||      5992      |
5994      ||      5993      |
5995      ||      5994      |
5996      ||      5994      |
5997      ||      5996      |
5998      ||      5997      |
5999      ||      5998      |

Vetor organizado com sucesso...

Tempo decorrido: 157 milisegundosPressione qualquer tecla para continuar. . .
```

Figura 50- Evidência do teste com vetor de 6000 posições.

- Quick sort, Insertion e merge



```
C:\Users\kamilasantos\Downloads\aps3.exe
Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 0 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 1438 milisegundos

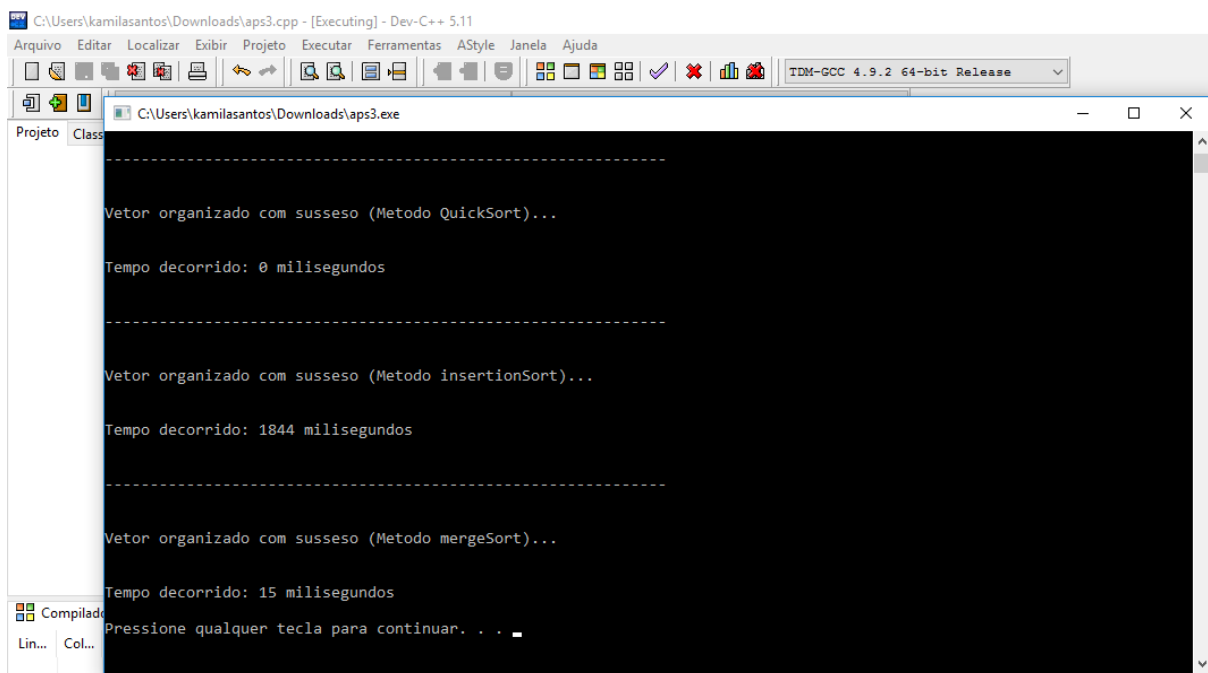
-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 15 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 51 - Evidência do teste com vetor de 35000 posições.



```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda
TDM-GCC 4.9.2 64-bit Release

Projeto Class
C:\Users\kamilasantos\Downloads\aps3.exe

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 0 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 1844 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 15 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 52- Evidência do teste com vetor de 40000 posições.

A screenshot of a console window titled "C:\Users\kamilasantos\Downloads\aps3.exe". The window displays the following text:

```
Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 15 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 2375 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 0 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 53- Evidência do teste com vetor de 45000 posições.

A screenshot of a console window titled "C:\Users\kamilasantos\Downloads\aps3.exe" overlaid on a Dev-C++ 5.11 window. The console displays the following text:

```
Tempo decorrido: 31 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 2953 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 16 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 54- Evidência do teste com vetor de 50000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda

C:\Users\kamilasantos\Downloads\aps3.exe
Tempo decorrido: 32 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 16 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 4188 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 15 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 55- Evidência do teste com vetor de 60000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda

(globale)
aps3.cpp
34 void quickSort(int*vetor, int left, int right);
35 void mergeSort();

C:\Users\kamilasantos\Downloads\aps3.exe
Tempo decorrido: 47 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 15 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 7578 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 31 milisegundos

Pressione qualquer tecla para continuar. . .
```

Figura 56- Evidência do teste com vetor de 80000 posições.



```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

C:\Users\kamilasantos\Downloads\aps3.exe
Tempo decorrido: 47 milisegundos

-----

Vetor organizado com sucesso (Metodo QuickSort)...

Tempo decorrido: 32 milisegundos

-----

Vetor organizado com sucesso (Metodo insertionSort)...

Tempo decorrido: 11703 milisegundos

-----

Vetor organizado com sucesso (Metodo mergeSort)...

Tempo decorrido: 32 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 57- Evidência do teste com vetor de 100000 posições.

## -Selection sort

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

C:\Users\kamilasantos\Downloads\aps3.exe
988 986
989 989
990 990
991 992
992 993
993 993
994 994
995 995
996 996
997 997
998 998
999 999

Vetor organizado com sucesso...

Tempo decorrido: 15 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 58- Evidência do teste com vetor de 1000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Downloads\aps3.exe
1988    1990
1989    ||    1990
1990    ||    1991
1991    ||    1992
1992    ||    1993
1993    ||    1993
1994    ||    1994
1995    ||    1996
1996    ||    1996
1997    ||    1997
1998    ||    1998
1999    ||    1999

Vetor organizado com sucesso...

Tempo decorrido: 0 milisegundosPressione qualquer tecla para continuar...
```

Figura 59- Evidência do teste com vetor de 2000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Downloads\aps3.exe
2988    2986
2989    ||    2989
2990    ||    2990
2991    ||    2992
2992    ||    2993
2993    ||    2994
2994    ||    2995
2995    ||    2995
2996    ||    2996
2997    ||    2997
2998    ||    2997
2999    ||    2998

Vetor organizado com sucesso...

Tempo decorrido: 31 milisegundosPressione qualquer tecla para continuar...
```

Figura 60- Evidência do teste com vetor de 3000 posições.

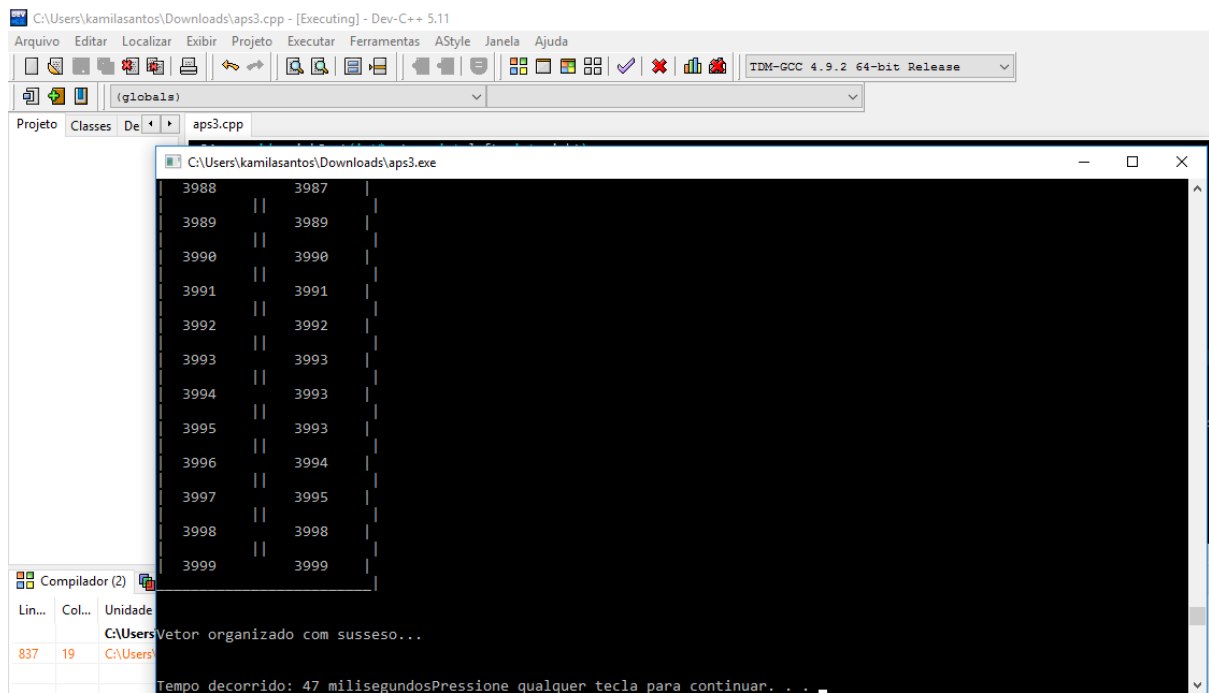


Figura 61- Evidência do teste com vetor de 4000 posições.

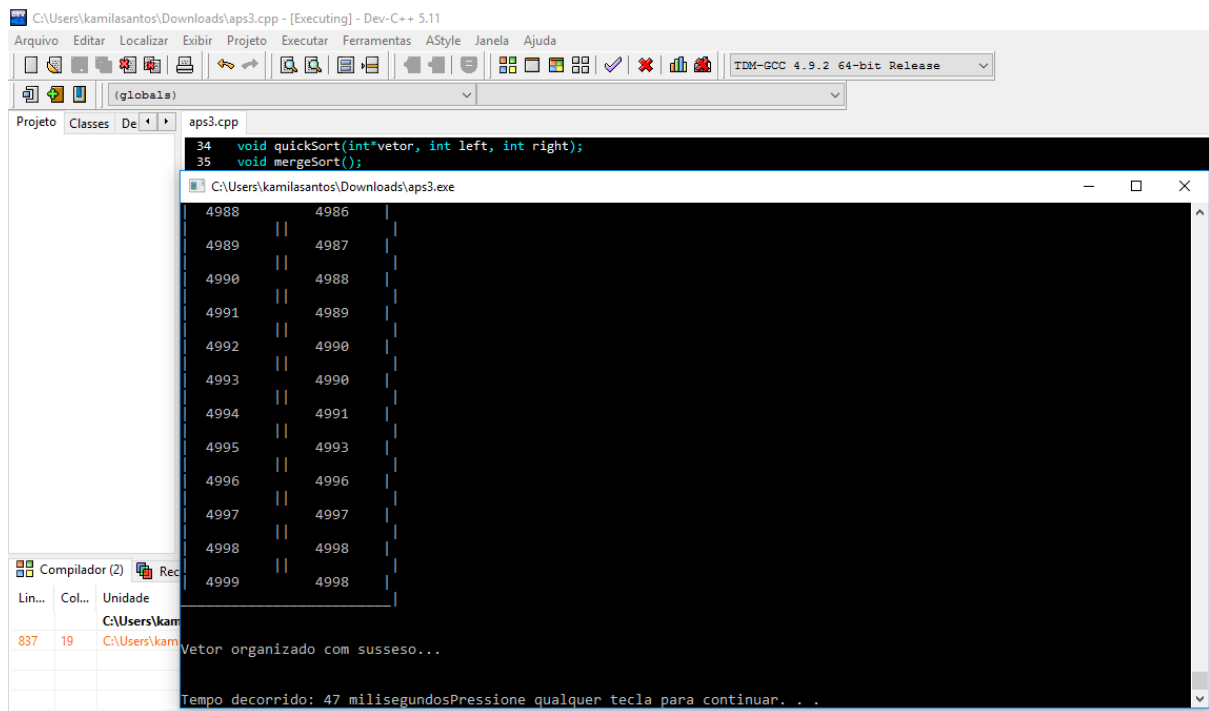


Figura 62- Evidência do teste com vetor de 5000 posições.

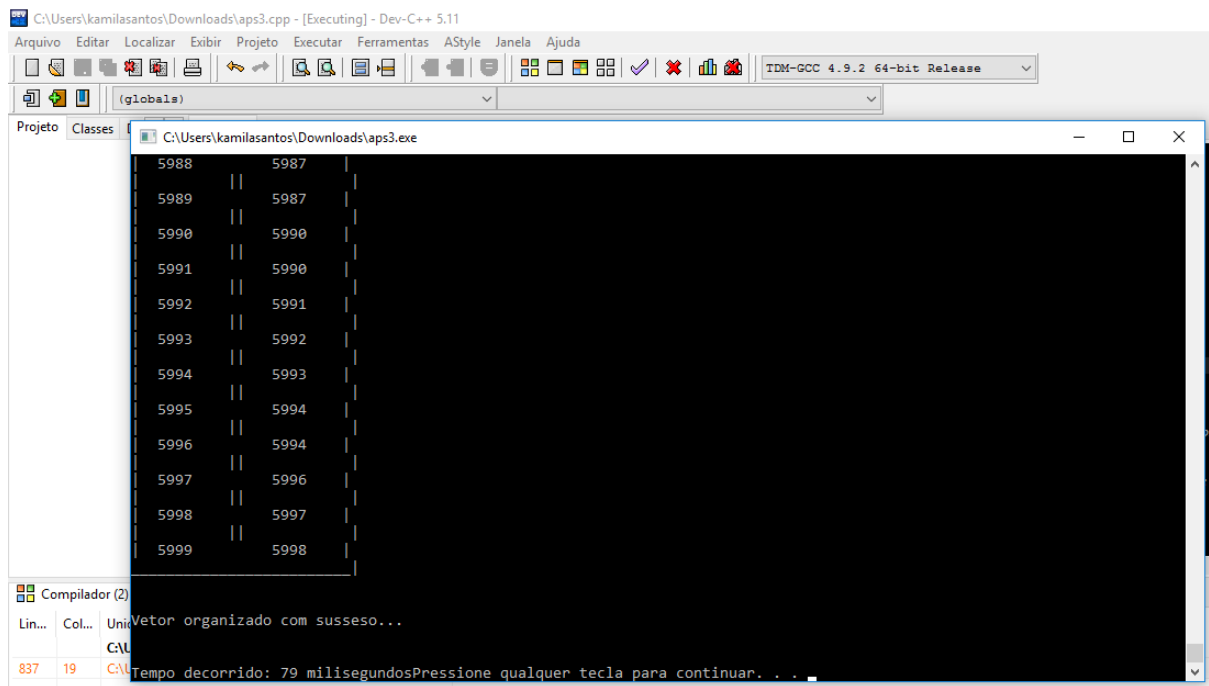


Figura 63- Evidência do teste com vetor de 6000 posições.

## -Shell sort

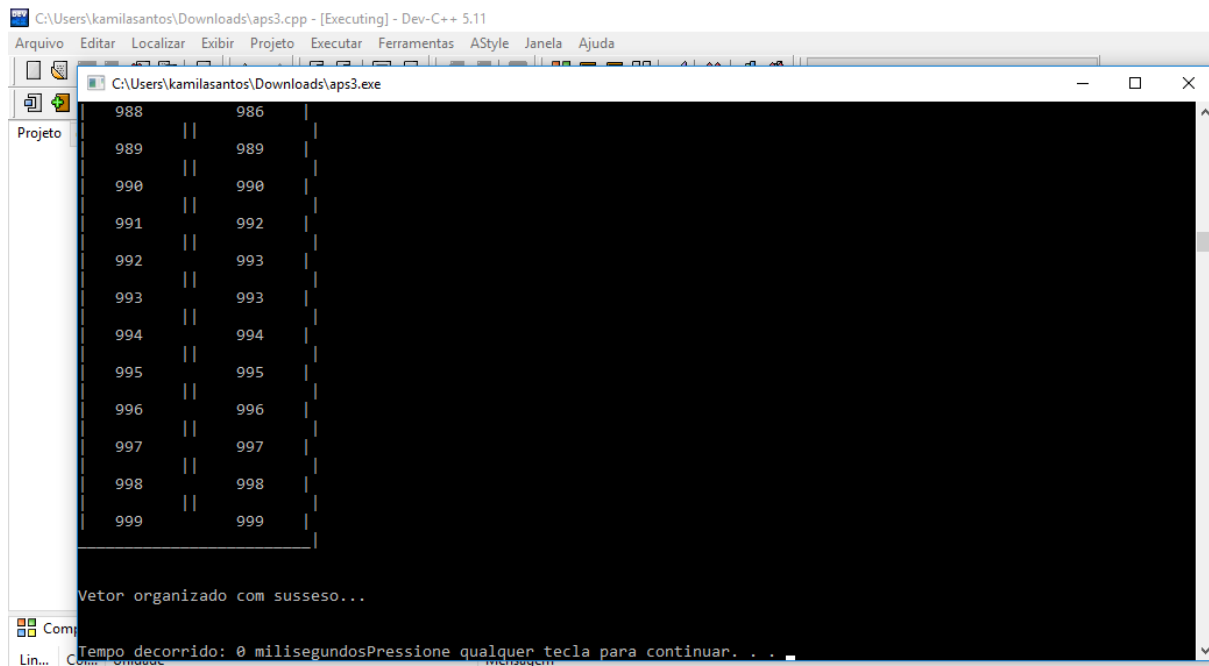


Figura 64- Evidência do teste com vetor de 1000 posições.

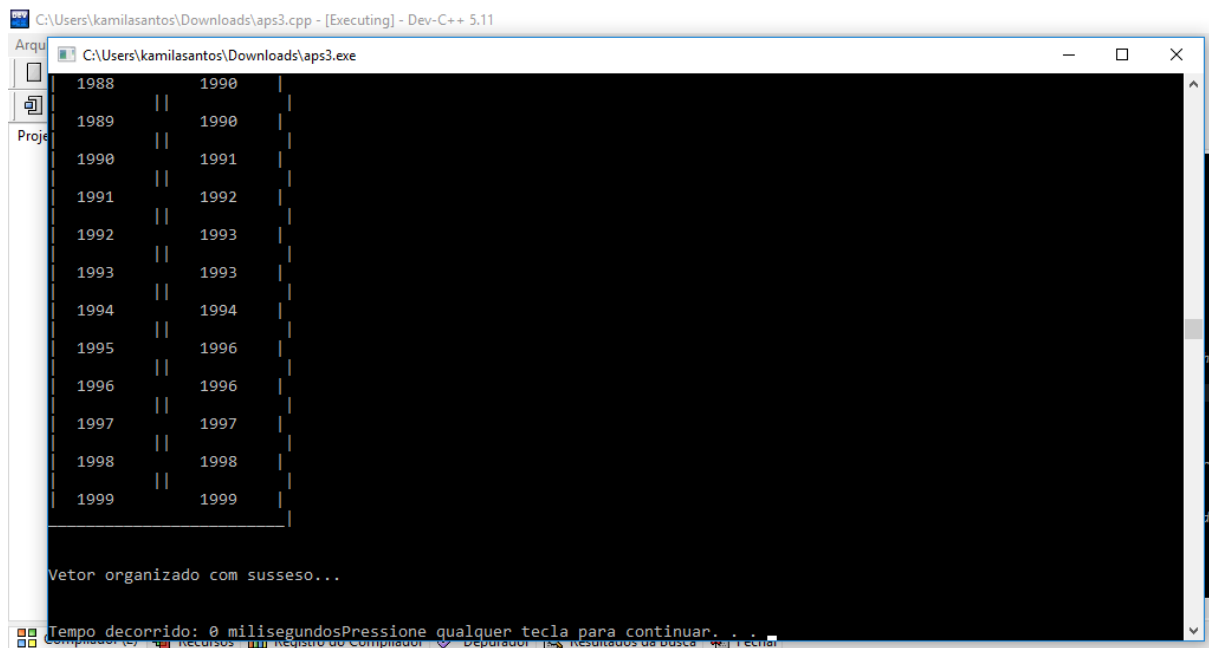


Figura 65- Evidência do teste com vetor de 2000 posições.

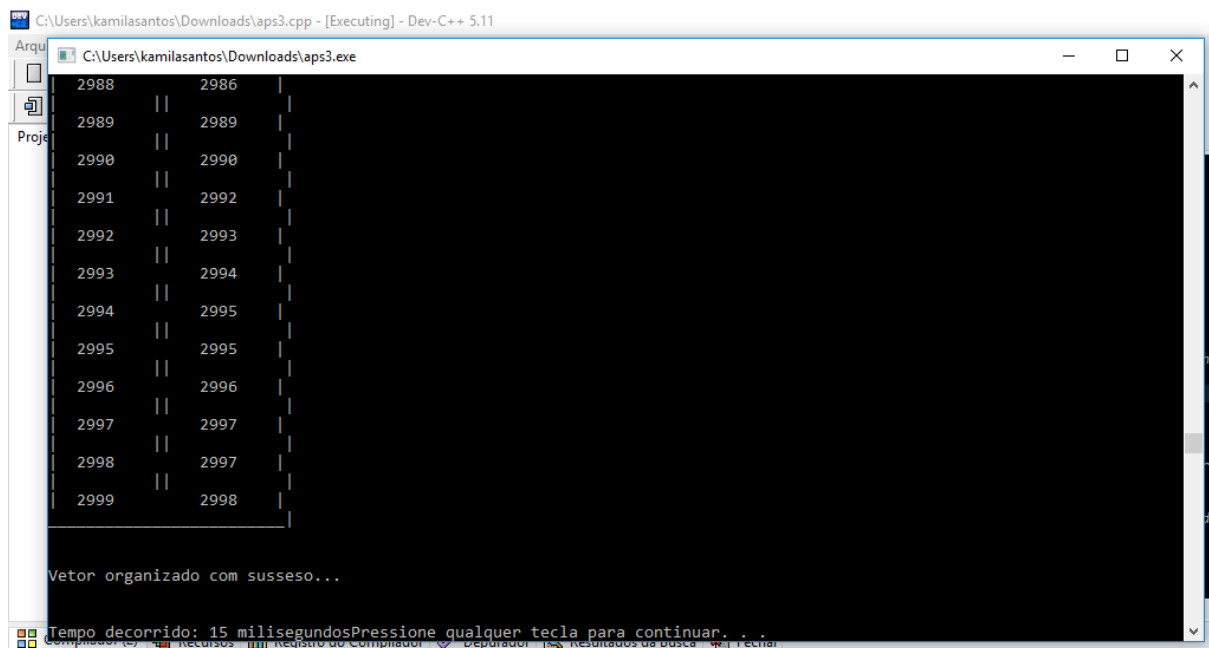


Figura 66- Evidência do teste com vetor de 3000 posições.

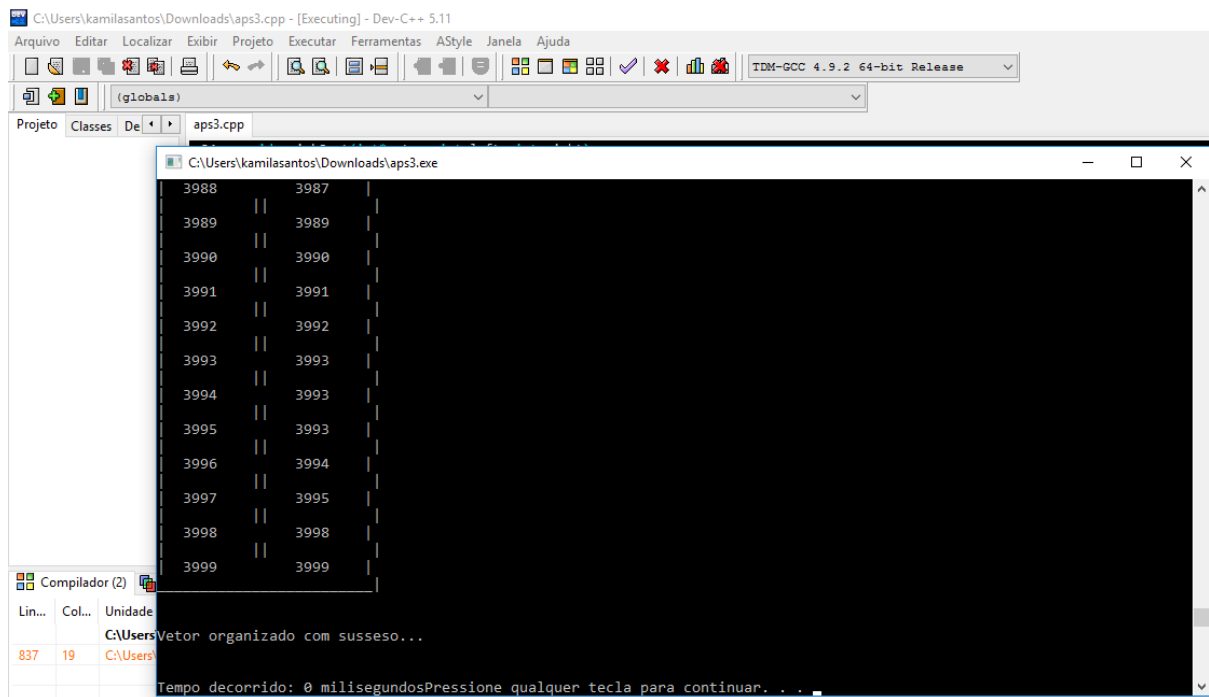


Figura 67- Evidência do teste com vetor de 4000 posições.

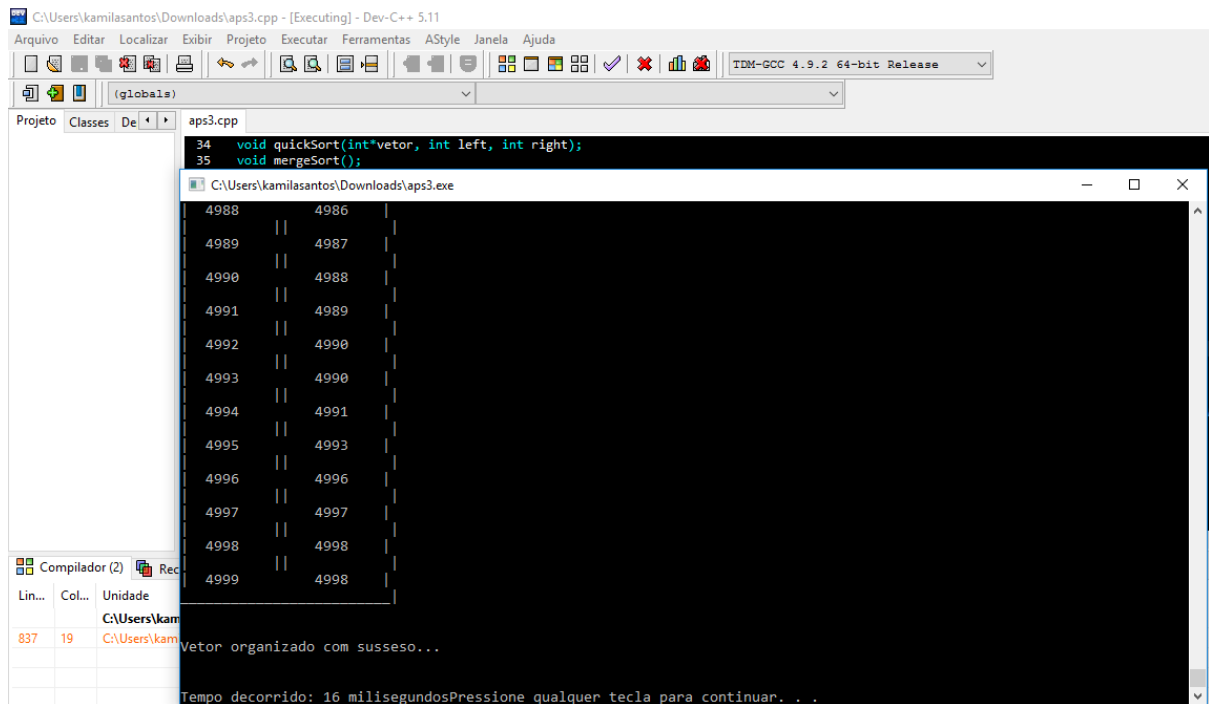


Figura 68- Evidência do teste com vetor de 5000 posições

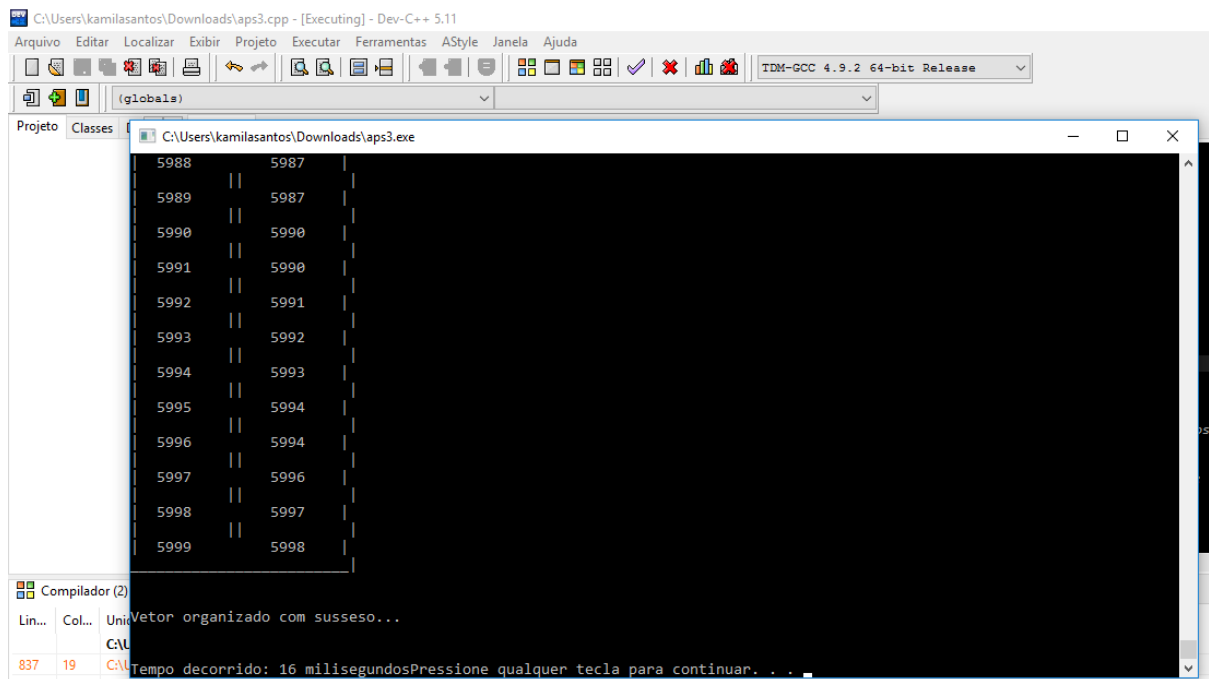


Figura 69- Evidência do teste com vetor de 6000 posições.

-Gráficos sobre o rendimento do programa

-Bubble sort

Entrada	tempo
1000	15
2000	31
3000	46
4000	79
5000	125
6000	157

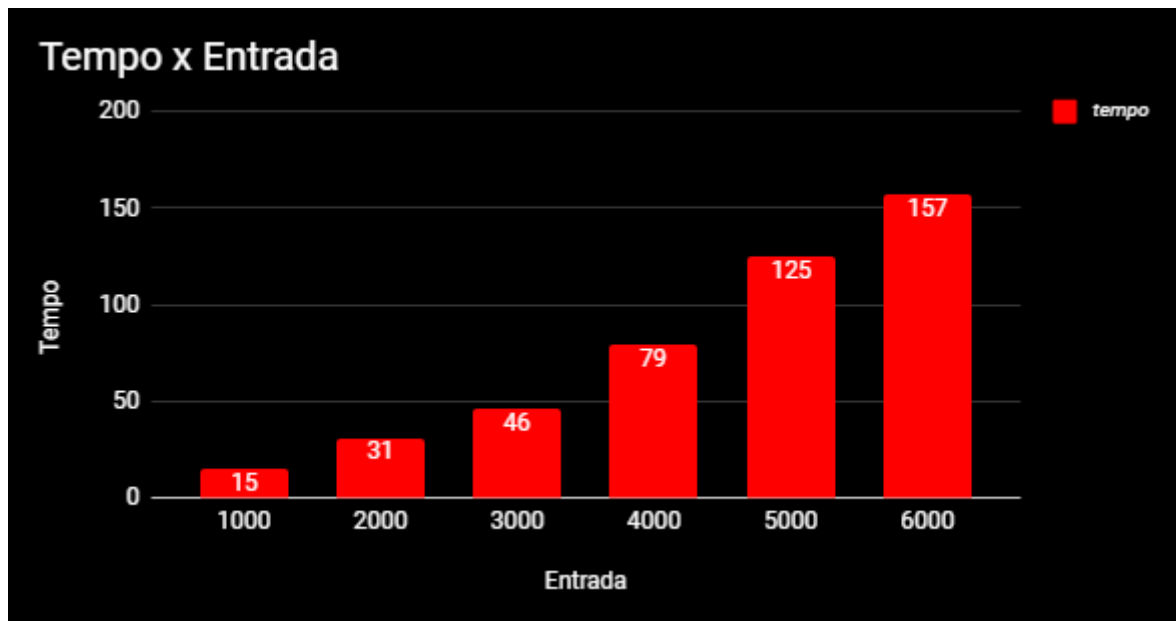


Figura 70- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Quick sort

Entrada	tempo
35000	0
40000	0
45000	15
50000	16
60000	16
80000	15
100000	32



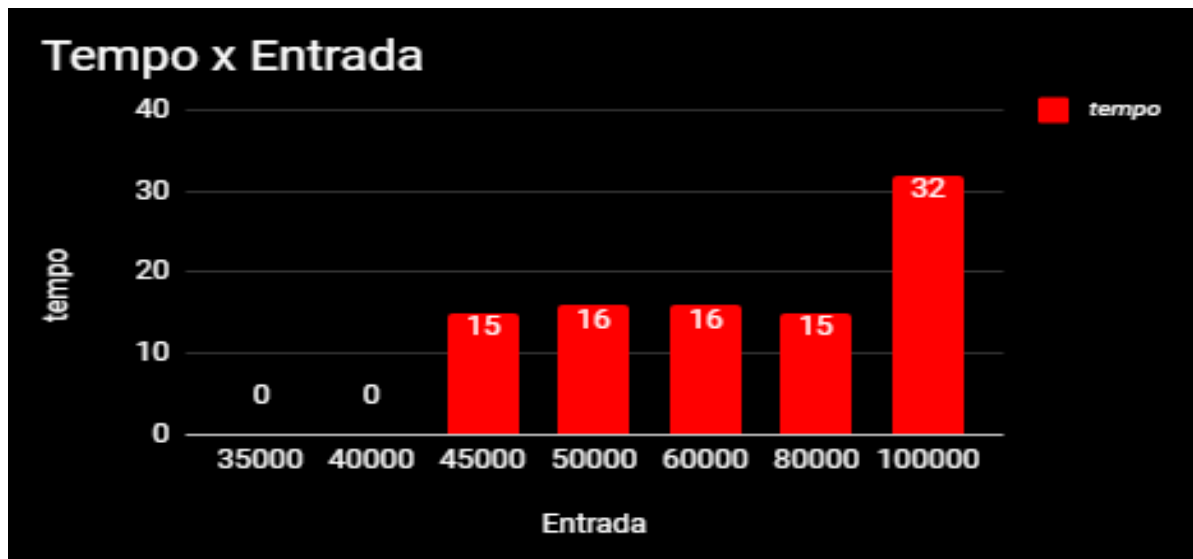


Figura 71- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Insertion sort

Entrada	tempo
35000	1438
40000	1844
45000	2375
50000	2953
60000	4188
80000	7578

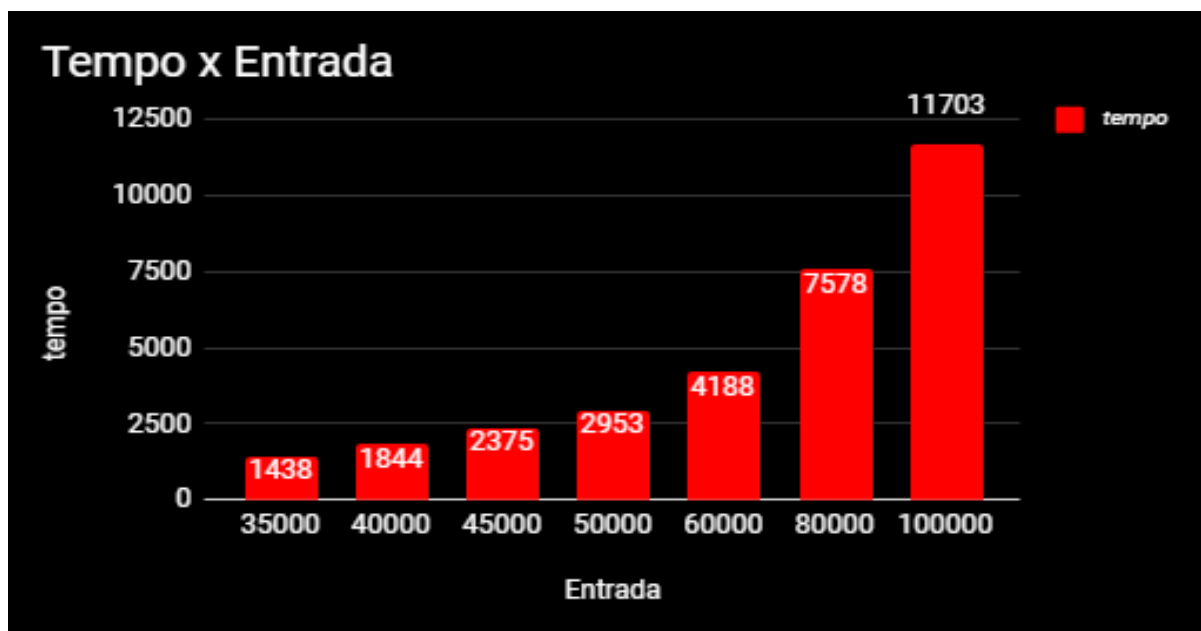


Figura 72- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Merge sort

Entrada	tempo
35000	15
40000	15
45000	0
50000	16
60000	15
80000	31
100000	32

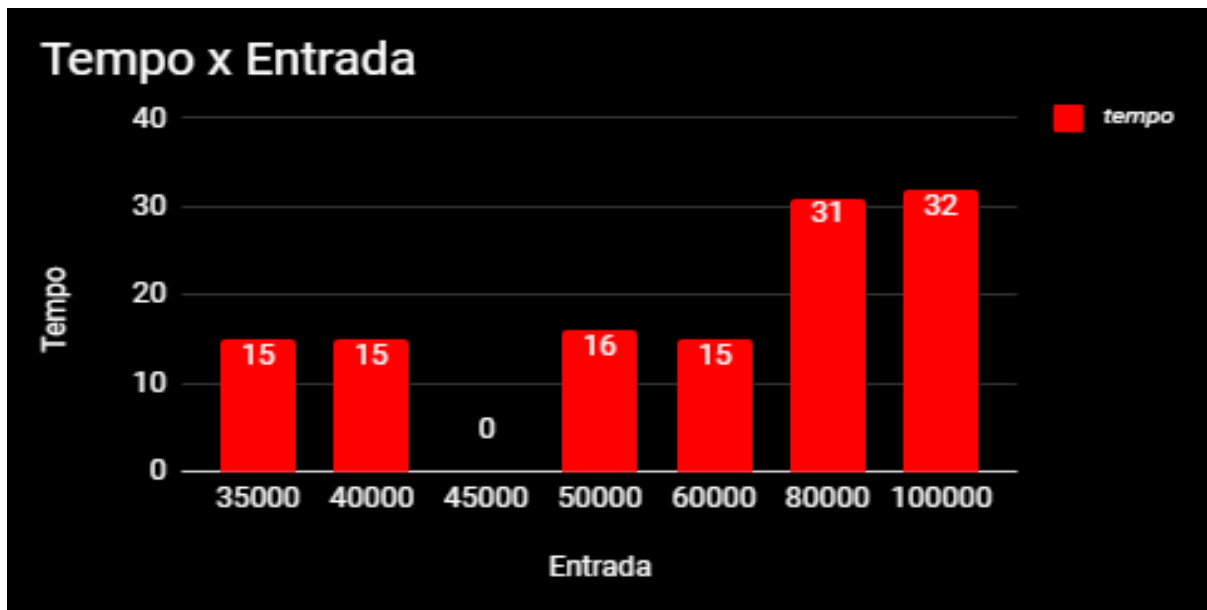


Figura 73- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Selection sort

Entrada	tempo
1000	15
2000	0
3000	31
4000	47
5000	47
6000	79

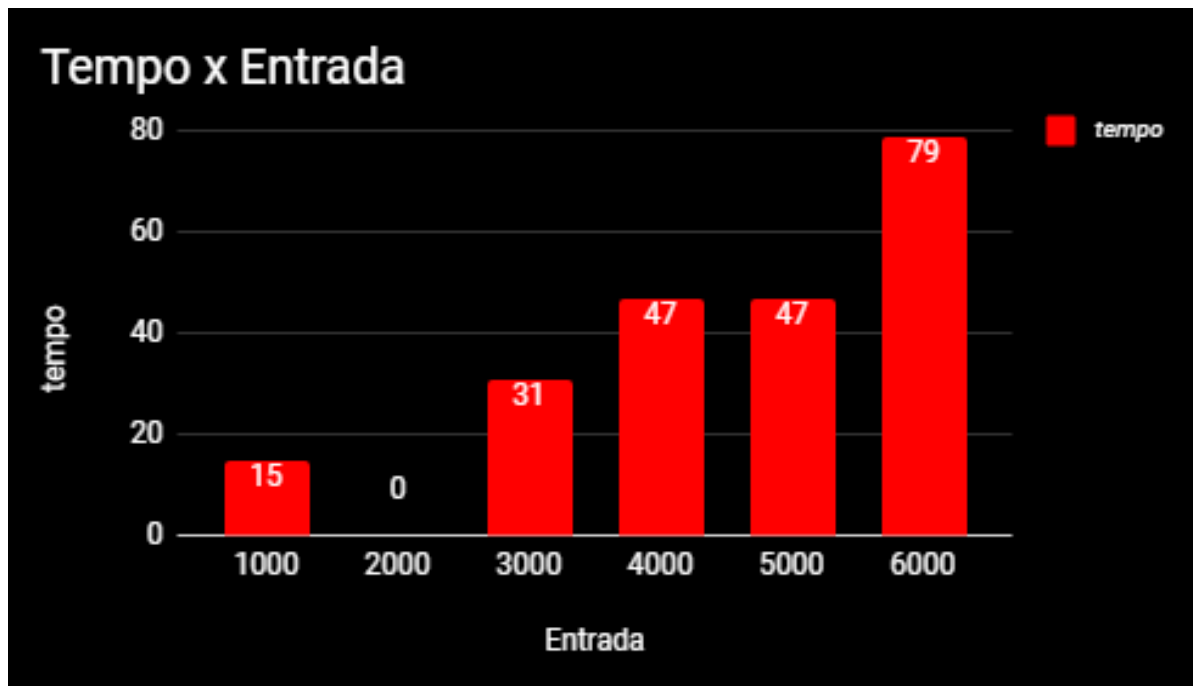


Figura 74- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Shellsort

Entrada	tempo
1000	0
2000	0
3000	15
4000	0
5000	16
6000	16

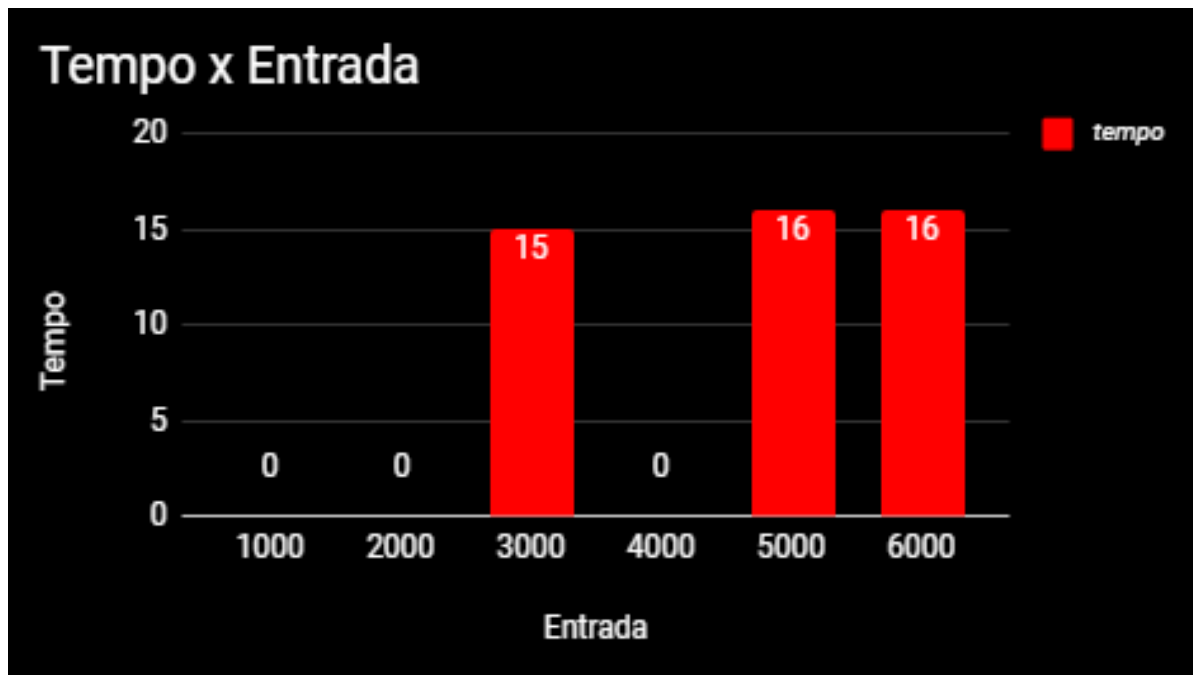


Figura 75- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

Testes com vetor ordenado

-Evidências

-Bubble sort

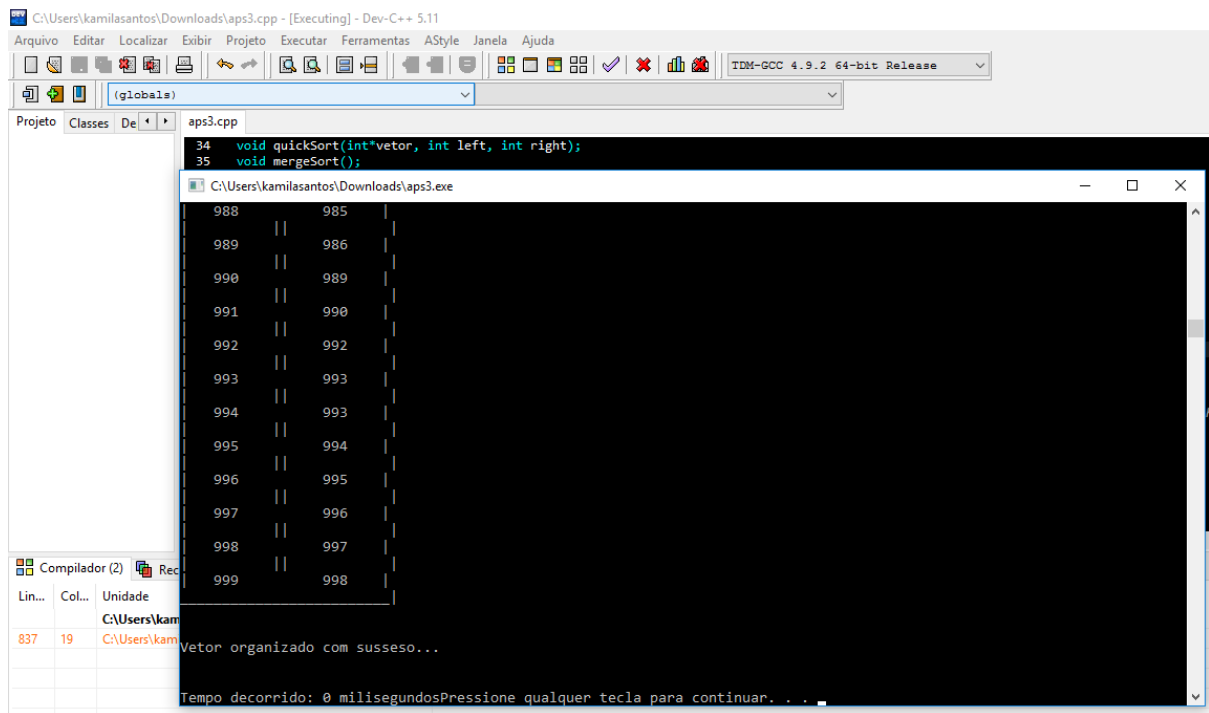


Figura 76- Evidência do teste com vetor de 1000 posições.

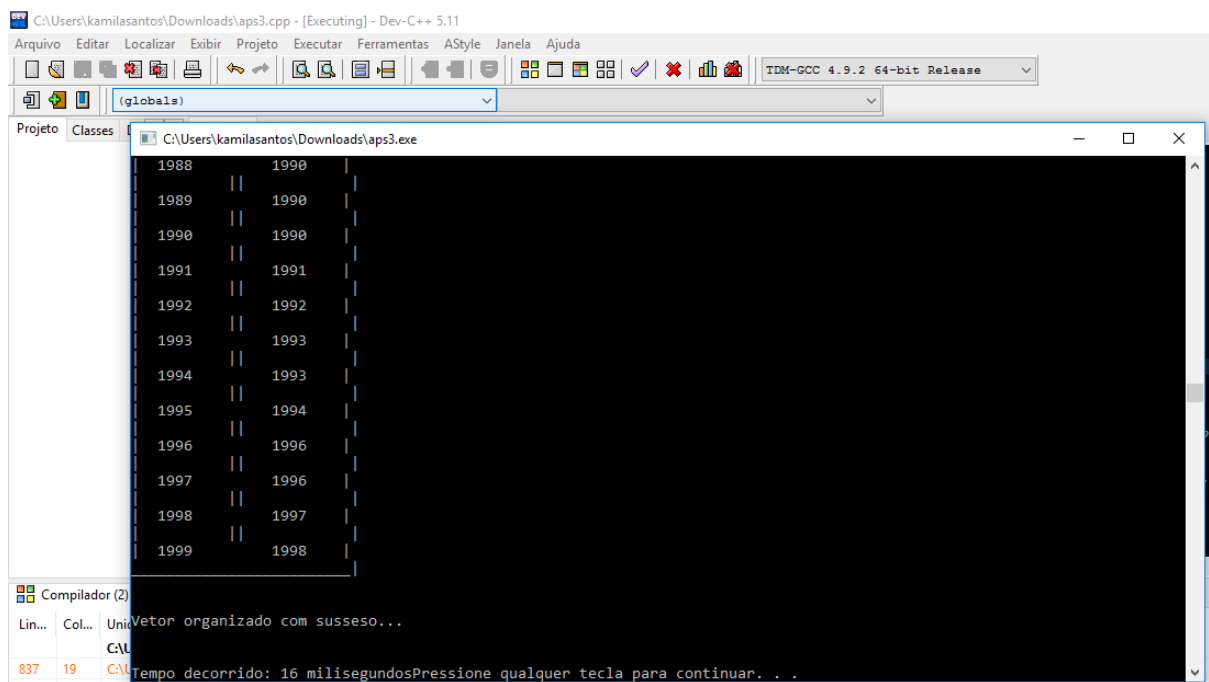


Figura 77- Evidência do teste com vetor de 2000 posições.

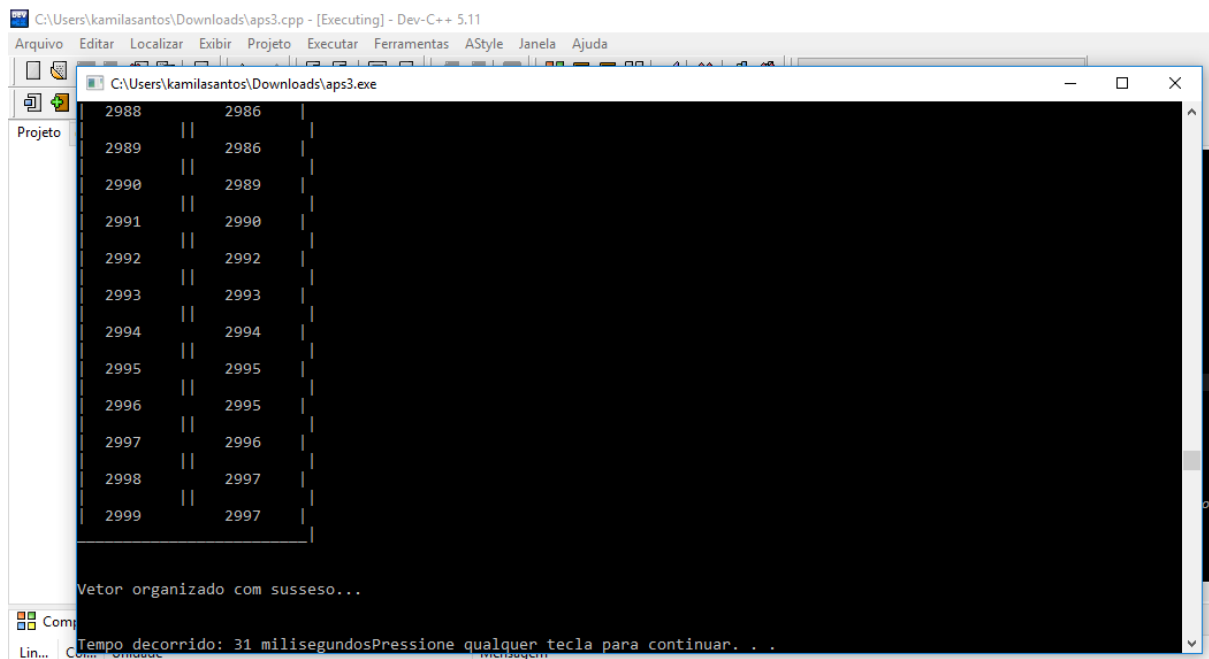


Figura 78- Evidência do teste com vetor de 3000 posições

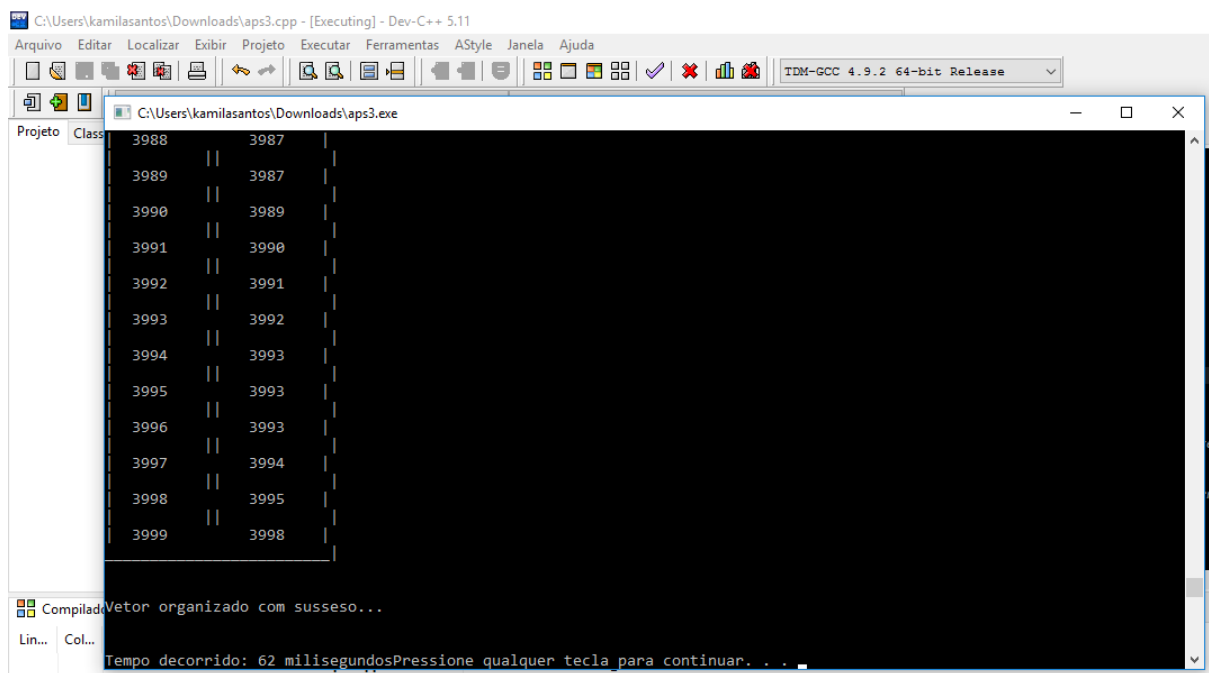


Figura 79- Evidência do teste com vetor de 4000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
C:\Users\kamilasantos\Downloads\aps3.exe
4988      4984
4989      ||      4986
4990      ||      4987
4991      ||      4988
4992      ||      4989
4993      ||      4990
4994      ||      4990
4995      ||      4991
4996      ||      4993
4997      ||      4996
4998      ||      4997
4999      ||      4998

Vetor organizado com sucesso...
Tempo decorrido: 78 milisegundosPressione qualquer tecla para continuar...
```

Figura 80- Evidência do teste com vetor de 5000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda
(globals)
Projeto  Classes  C:\Users\kamilasantos\Downloads\aps3.exe
5988      5986
5989      ||      5987
5990      ||      5987
5991      ||      5990
5992      ||      5990
5993      ||      5991
5994      ||      5992
5995      ||      5993
5996      ||      5994
5997      ||      5994
5998      ||      5996
5999      ||      5997

Vetor organizado com sucesso...
Tempo decorrido: 109 milisegundosPressione qualquer tecla para continuar...
```

Figura 81- Evidência do teste com vetor de 6000 posições.

-Quick sort, Merge sort e Insertion sort



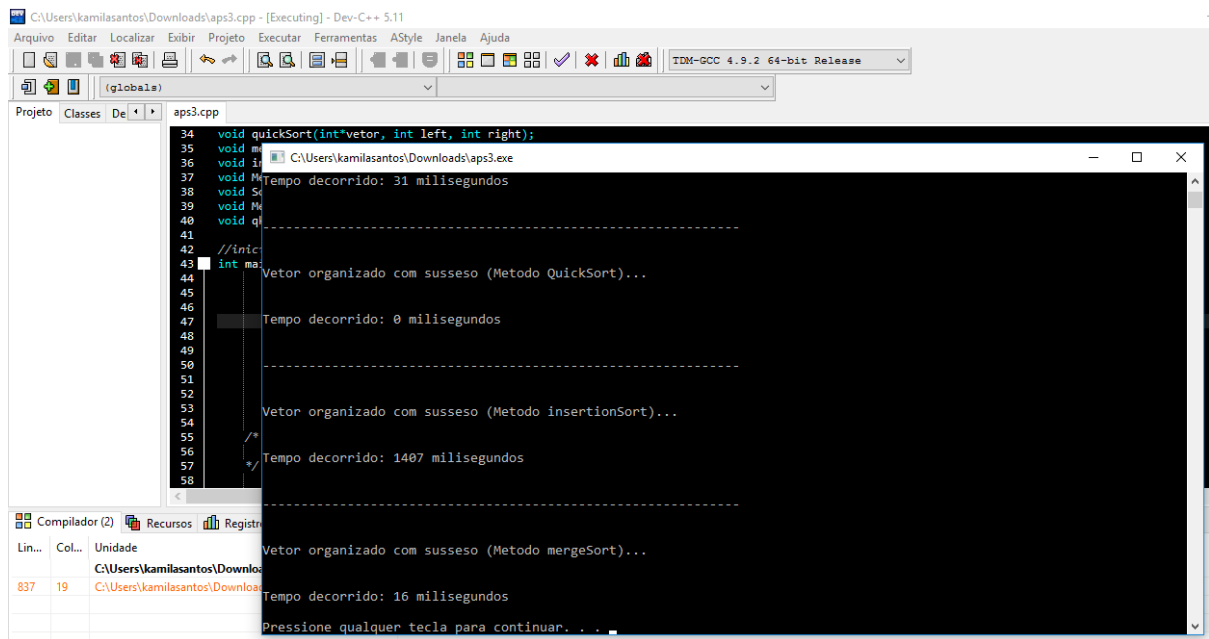


Figura 82- Evidência do teste com vetor de 35000 posições.

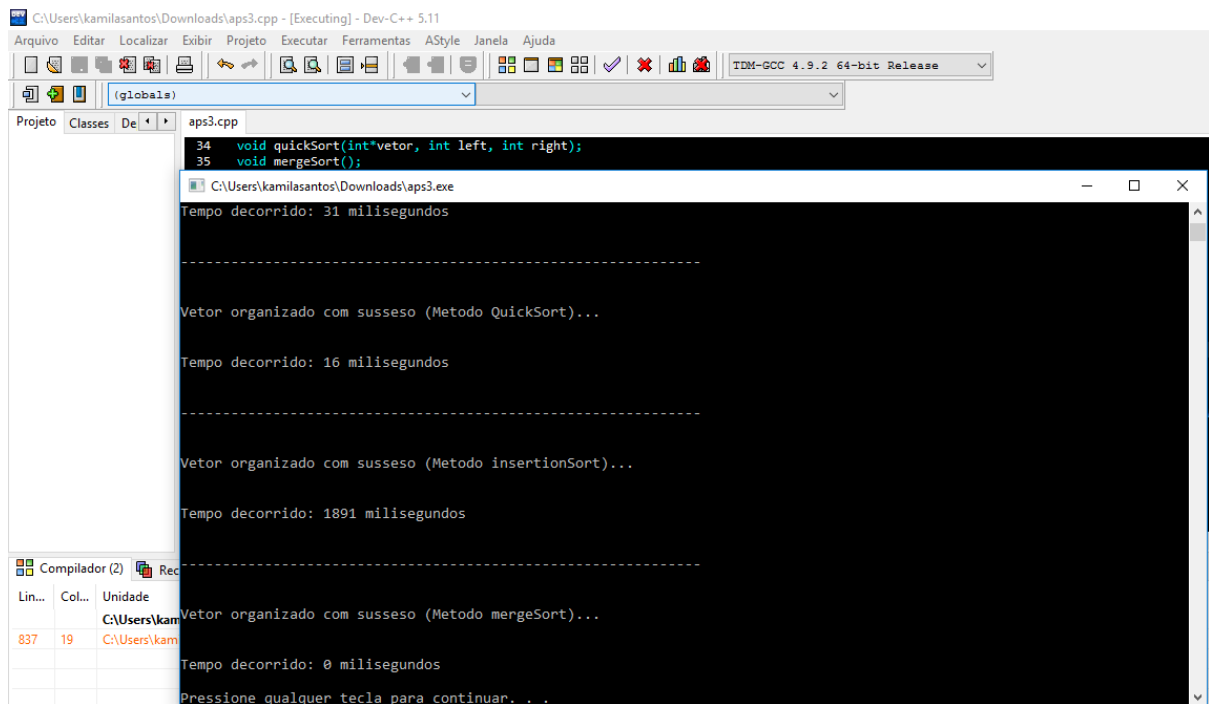


Figura 83- Evidência do teste com vetor de 40000 posições.

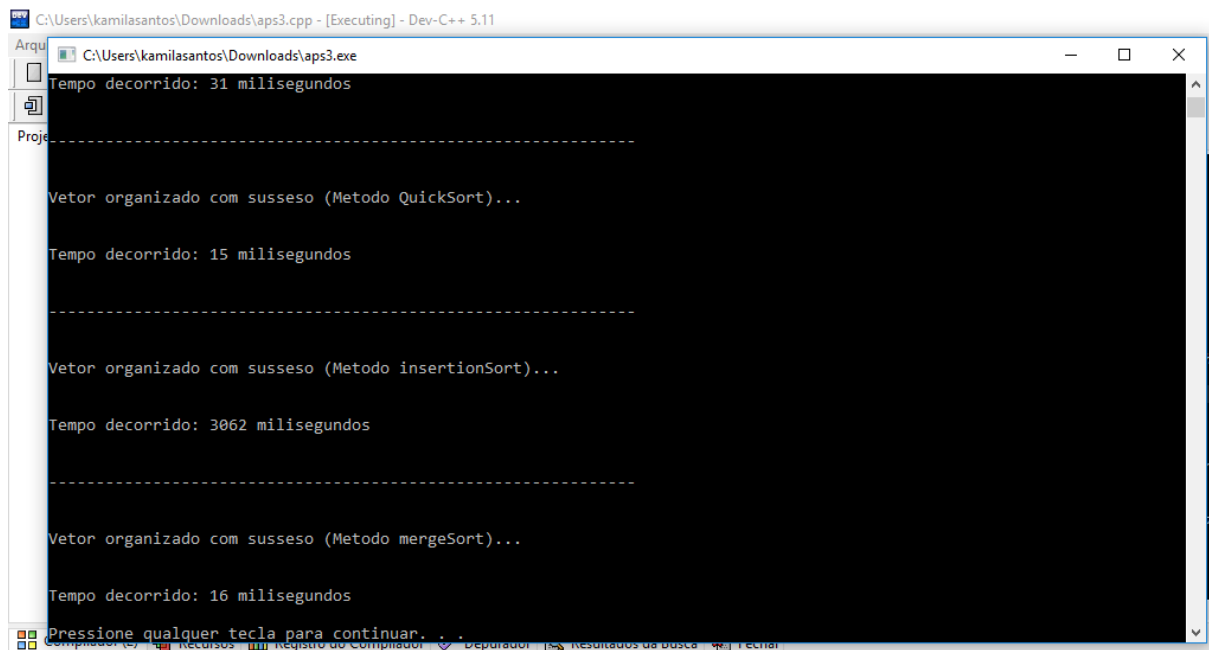


Figura 84- Evidência do teste com vetor de 45000 posições.

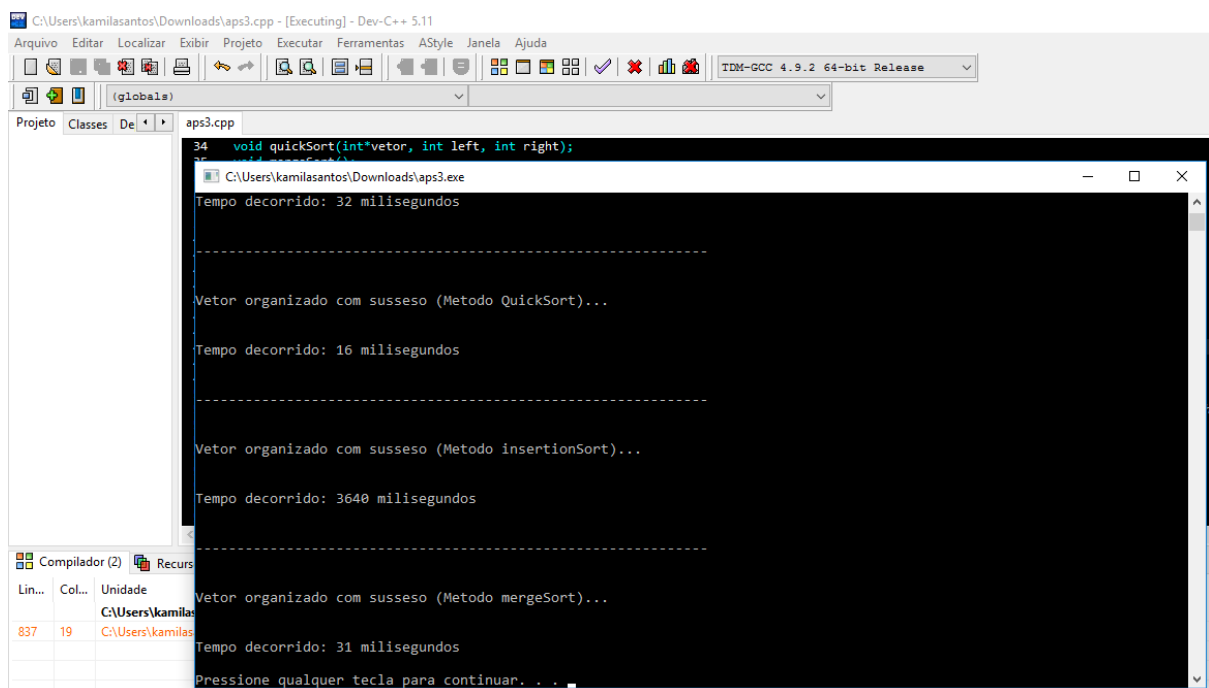


Figura 85- Evidência do teste com vetor de 50000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda
(globals)
Projeto  Classes  Del  aps3.cpp
C:\Users\kamilasantos\Downloads\aps3.exe
Tempo decorrido: 47 milisegundos
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 16 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 5281 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 15 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 86- Evidência do teste com vetor de 60000 posições.

```
C:\Users\kamilasantos\Downloads\aps3.cpp - [Executing] - Dev-C++ 5.11
Arqu...
C:\Users\kamilasantos\Downloads\aps3.exe
-----
Vetor organizado com sucesso (Metodo QuickSort)...
Tempo decorrido: 15 milisegundos
-----
Vetor organizado com sucesso (Metodo insertionSort)...
Tempo decorrido: 7547 milisegundos
-----
Vetor organizado com sucesso (Metodo mergeSort)...
Tempo decorrido: 31 milisegundos
Pressione qualquer tecla para continuar. . .
```

Figura 87- Evidência do teste com vetor de 80000 posições.

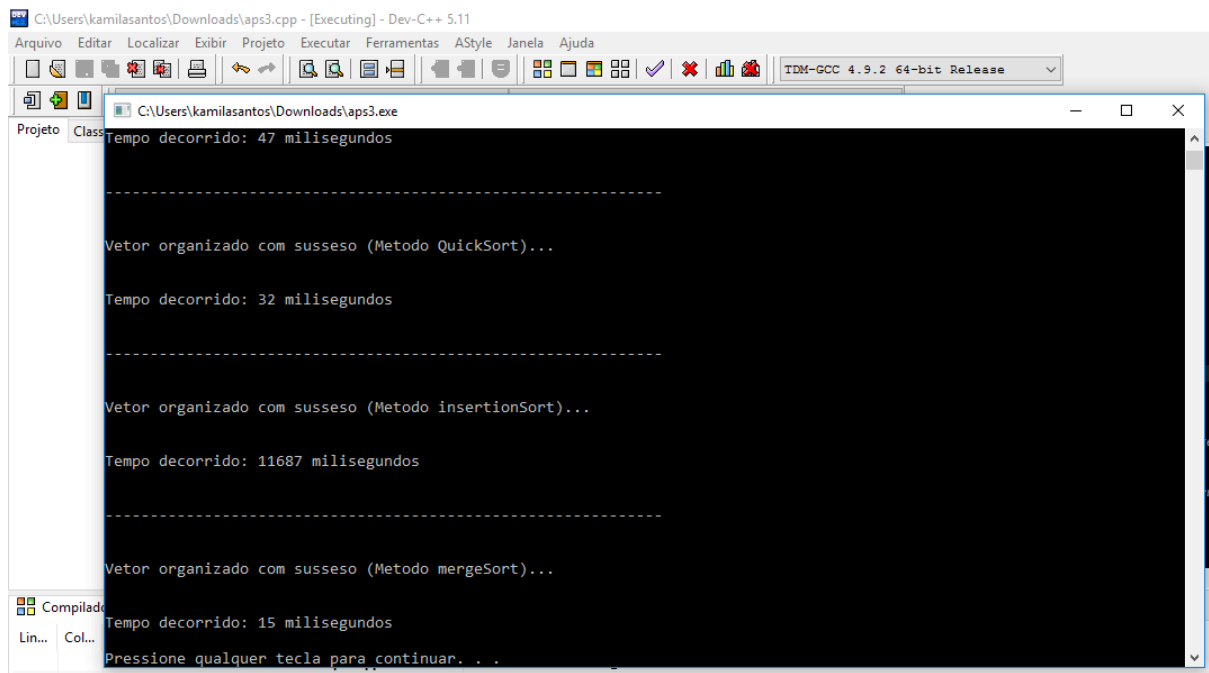


Figura 88- Evidência do teste com vetor de 100000 posições.

## -Selectionsort

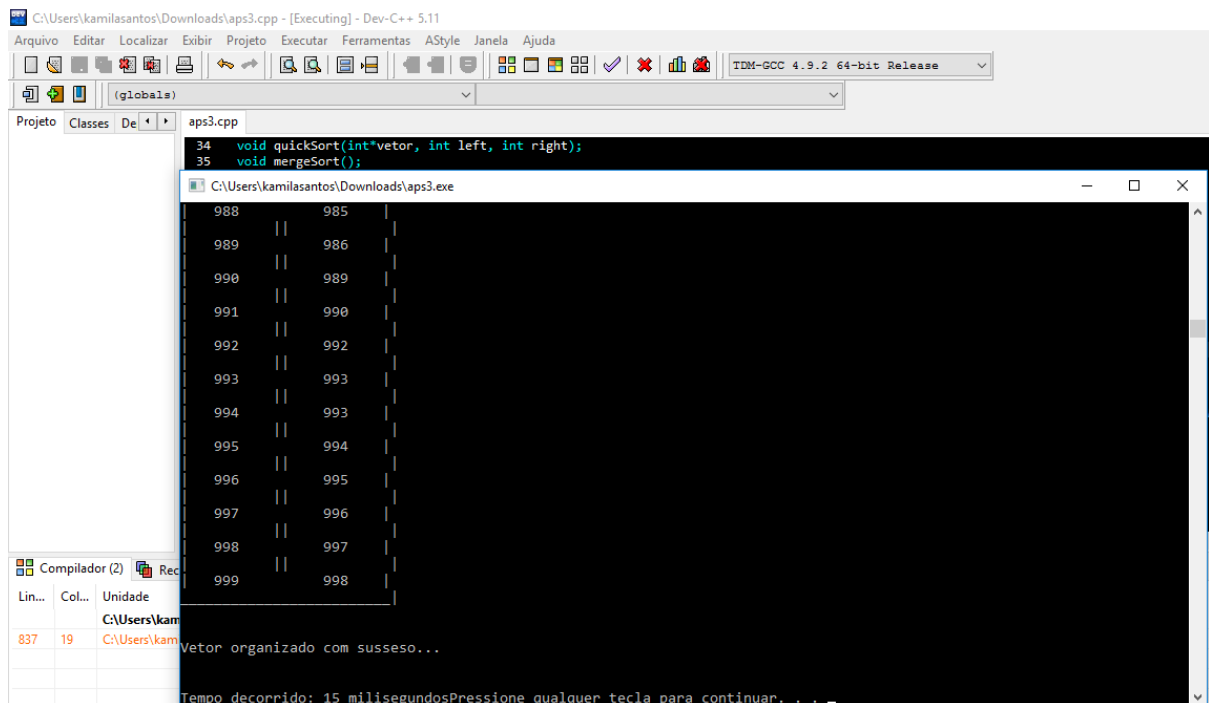


Figura 89- Evidência do teste com vetor de 1000 posições.

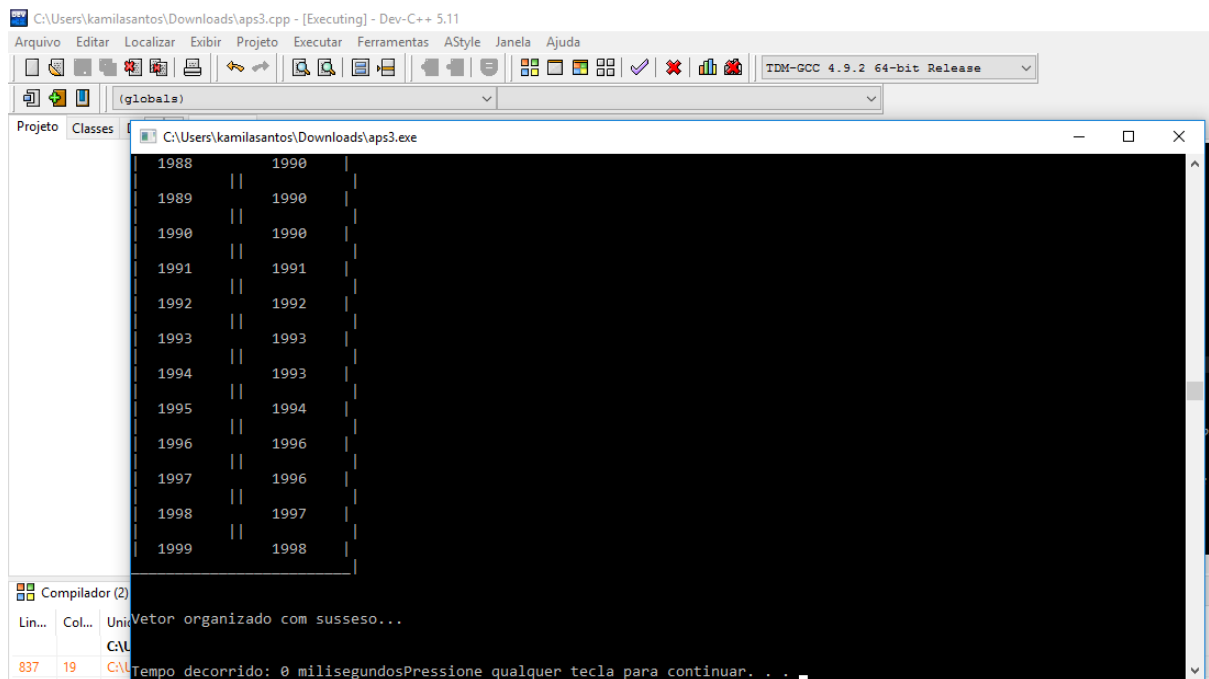


Figura 90- Evidência do teste com vetor de 2000 posições.

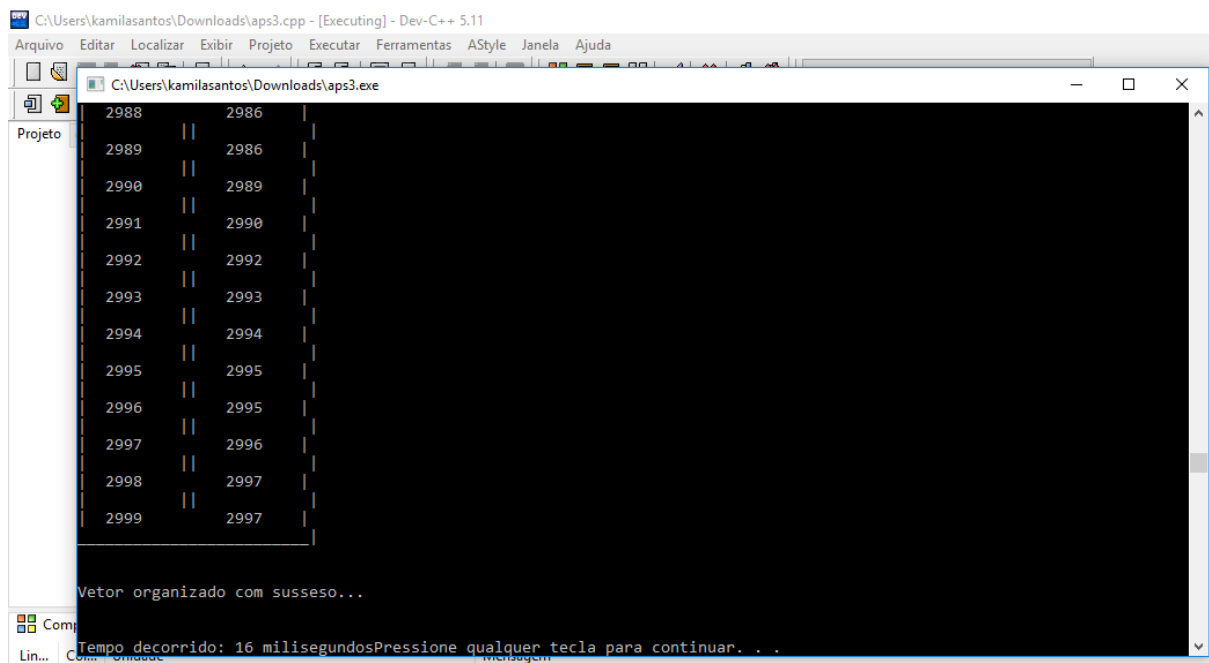


Figura 91- Evidência do teste com vetor de 3000 posições.

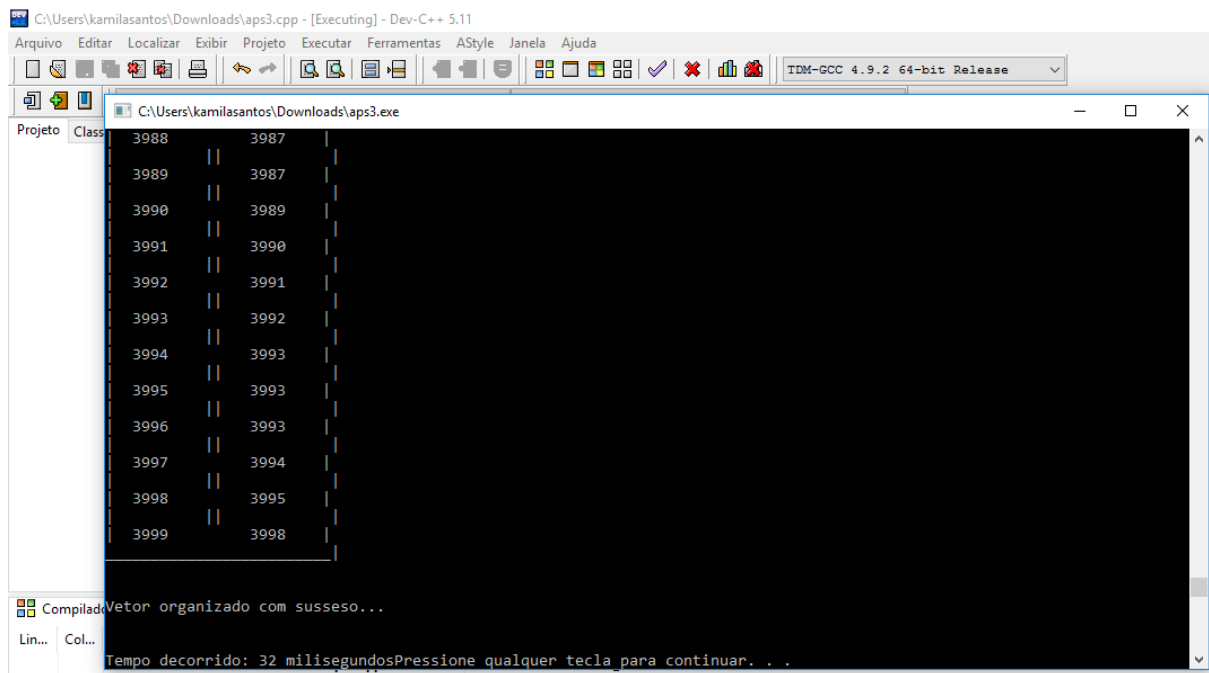


Figura 92- Evidência do teste com vetor de 4000 posições.

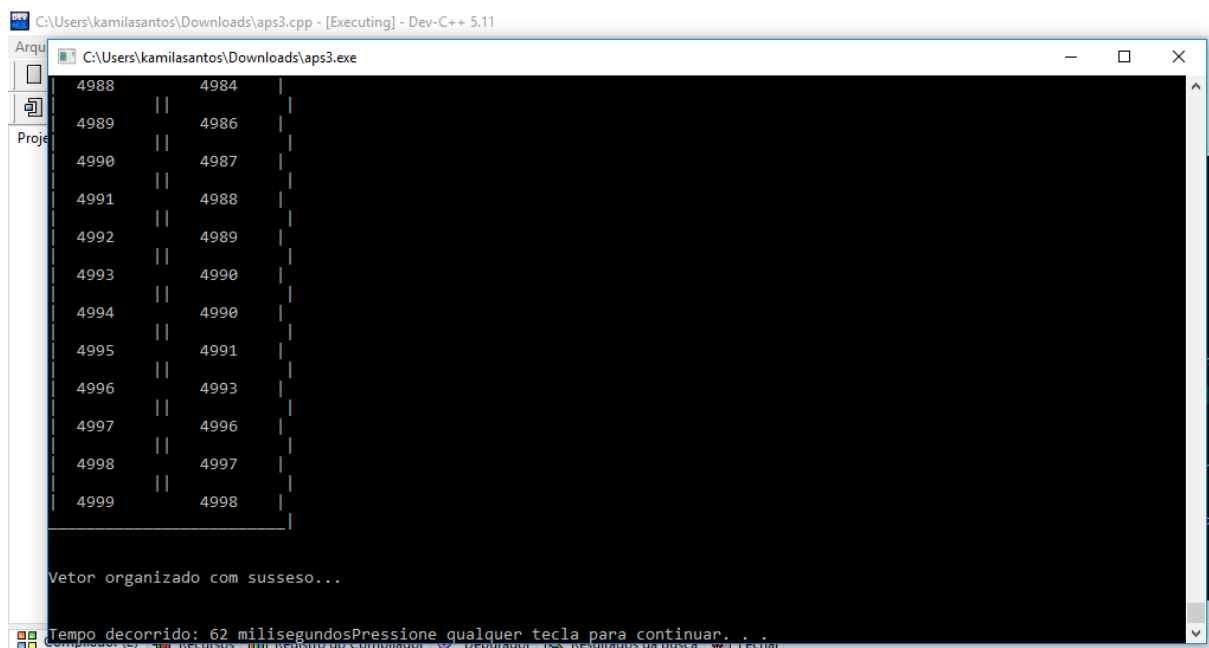


Figura 93- Evidência do teste com vetor de 5000 posições.

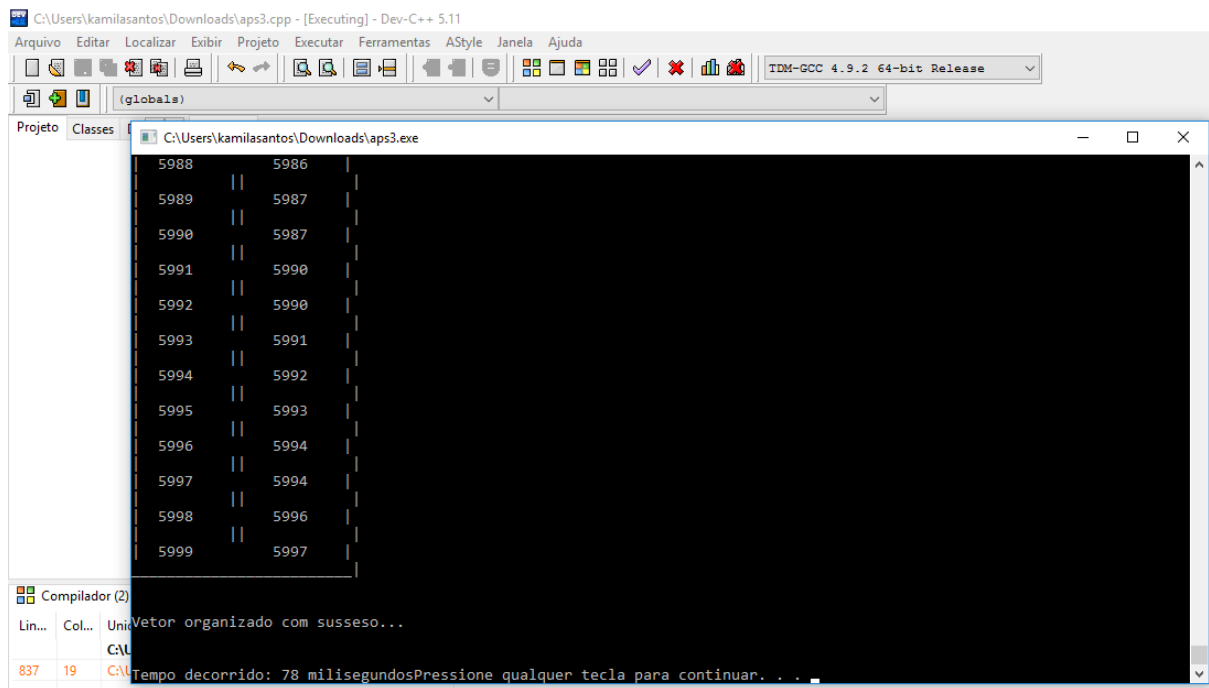


Figura 94- Evidência do teste com vetor de 6000 posições.

## -Shell sort

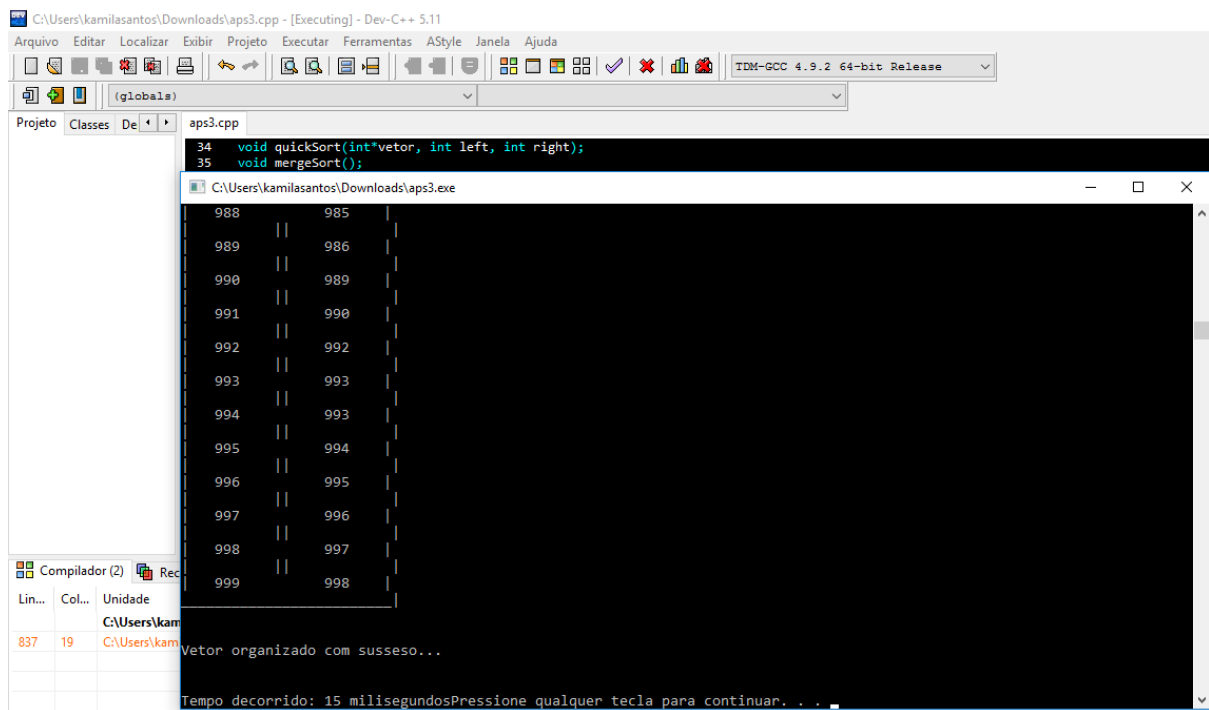


Figura 95- Evidência do teste com vetor de 1000 posições.

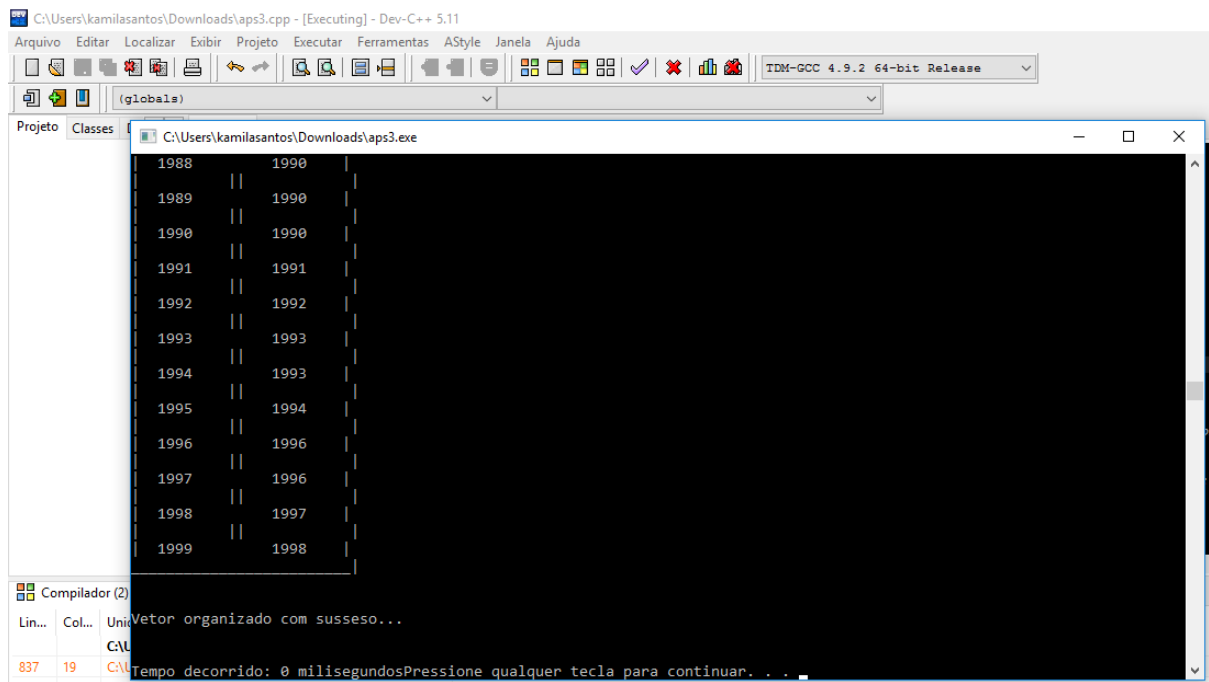


Figura 96- Evidência do teste com vetor de 2000 posições.

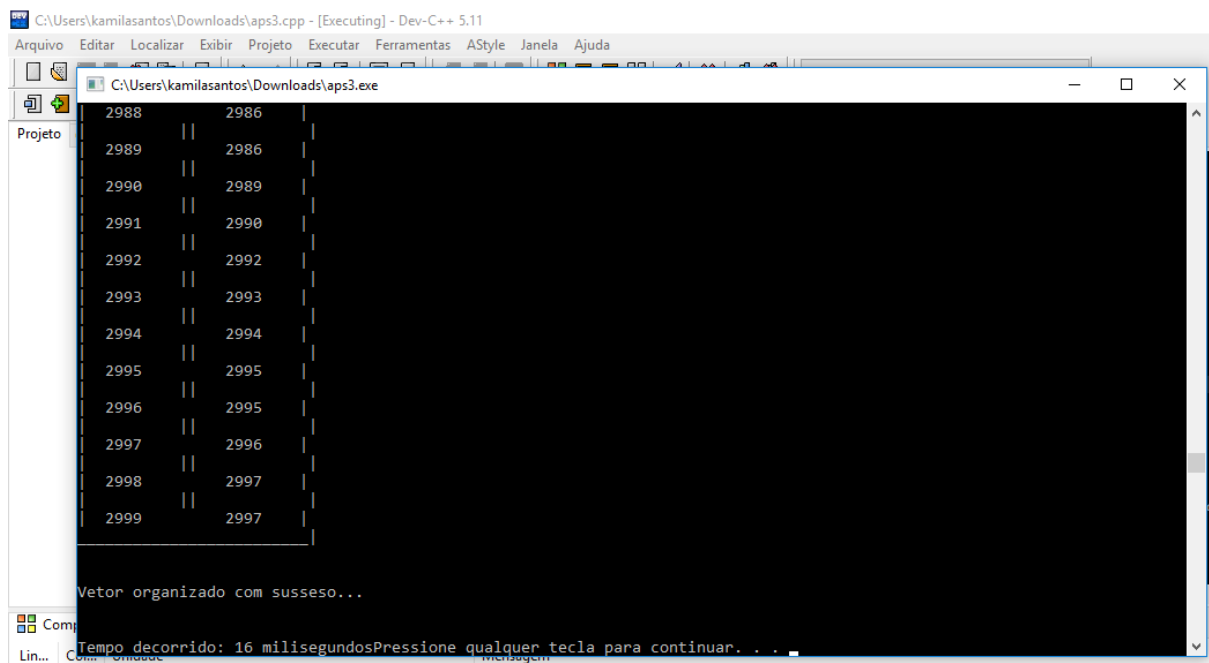


Figura 97- Evidência do teste com vetor de 3000 posições.



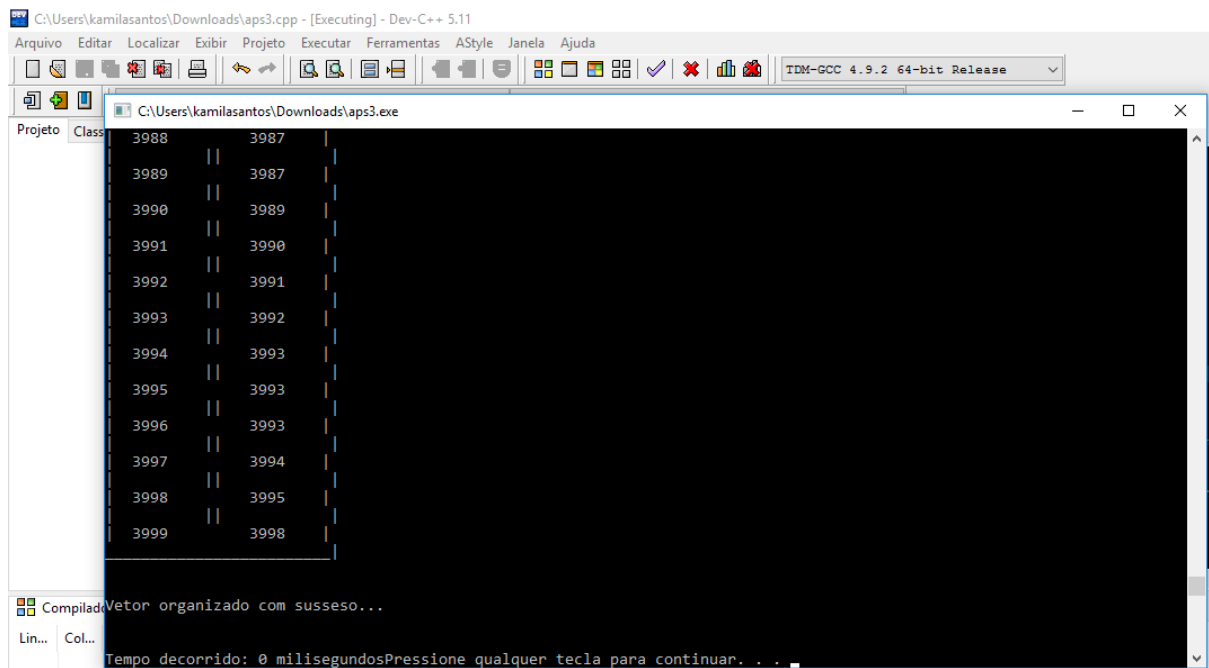


Figura 98- Evidência do teste com vetor de 4000 posições.

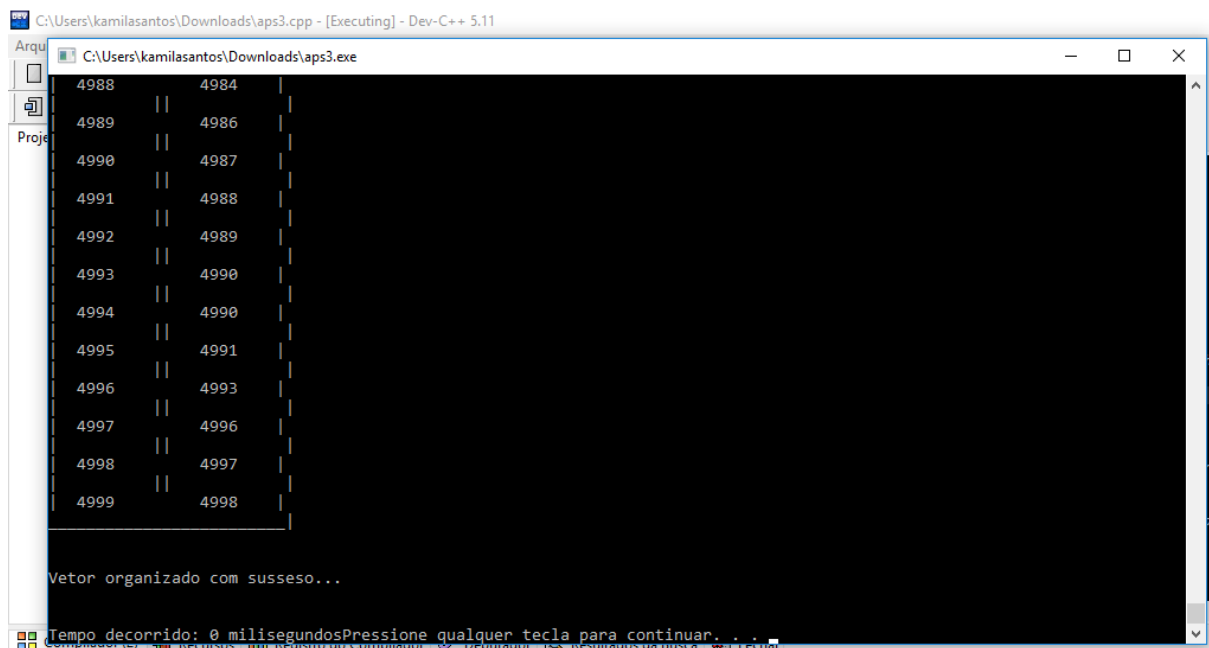


Figura 99- Evidência do teste com vetor de 5000 posições.

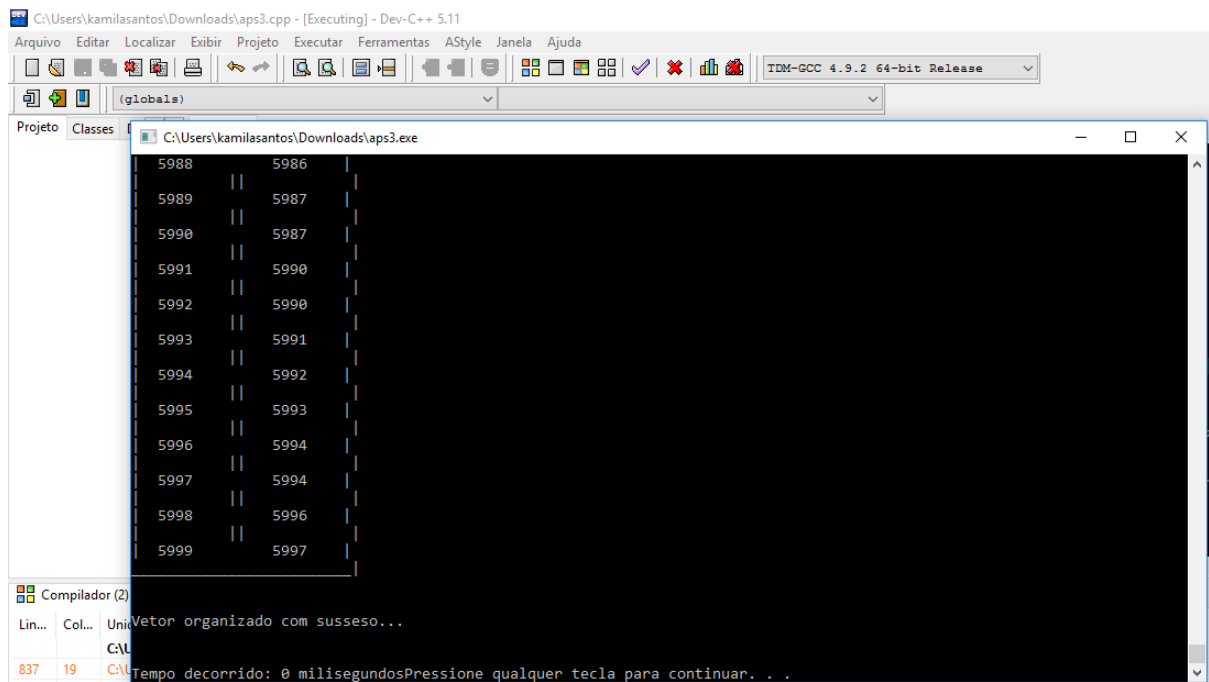


Figura 100- Evidência do teste com vetor de 6000 posições.

-Gráficos sobre o rendimento do programa

-Bubblesort

Entrada	tempo
1000	0
2000	16
3000	31
4000	62
5000	78
6000	109

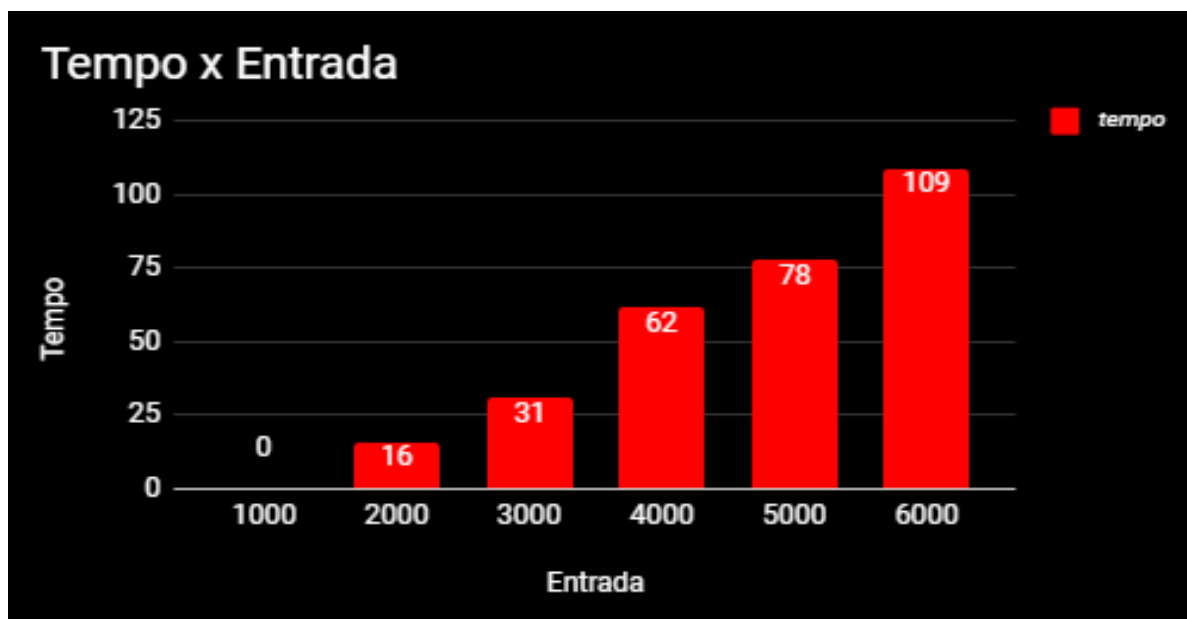


Figura 101- Gráfico com relação entre o volume de entrada de dados e o tempo de ordenação.

-Quick sort

Entrada	tempo
35000	0
40000	16
45000	15
50000	16
60000	16
80000	15
100000	32

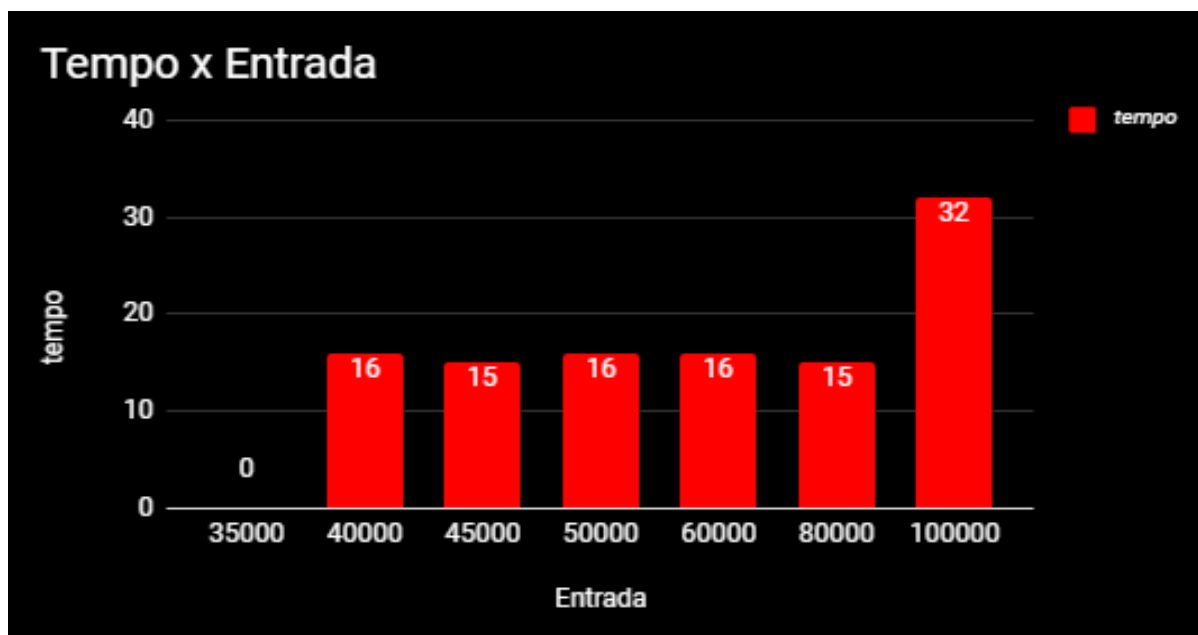


Figura 102- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Insertion sort

Entrada	tempo
35000	1407
40000	1891
45000	3062
50000	3640
60000	5281
80000	7547
100000	11687

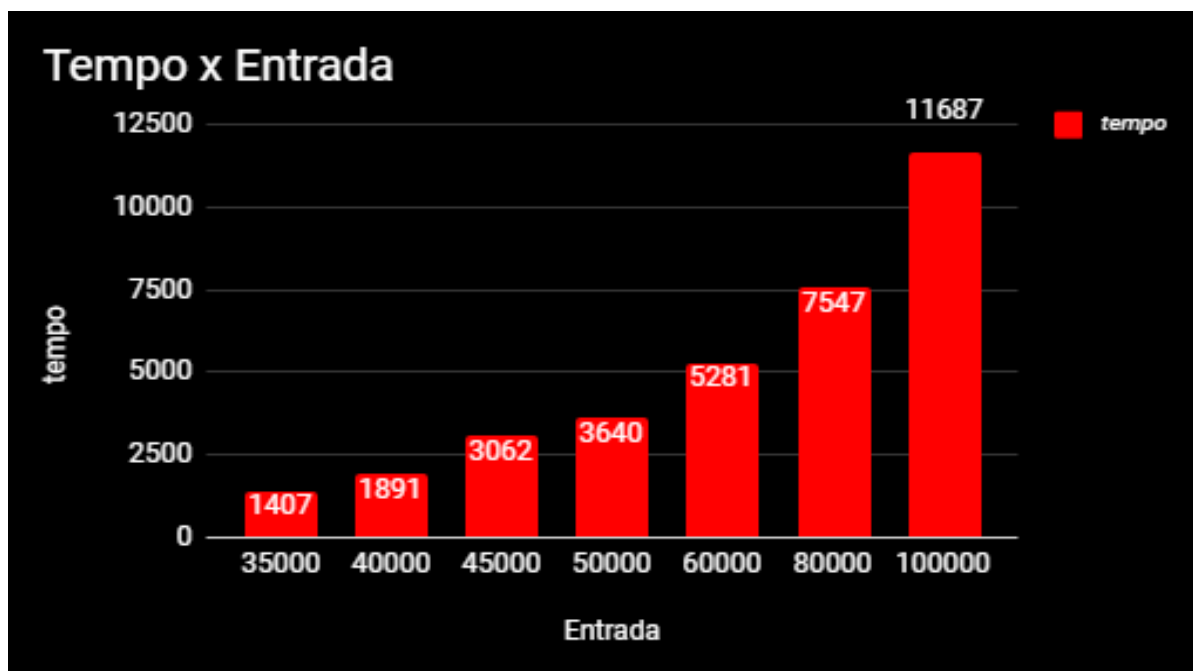


Figura 103- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Merge sort

Entrada	tempo
35000	16
40000	0
45000	16
50000	31
60000	15
80000	31
100000	15

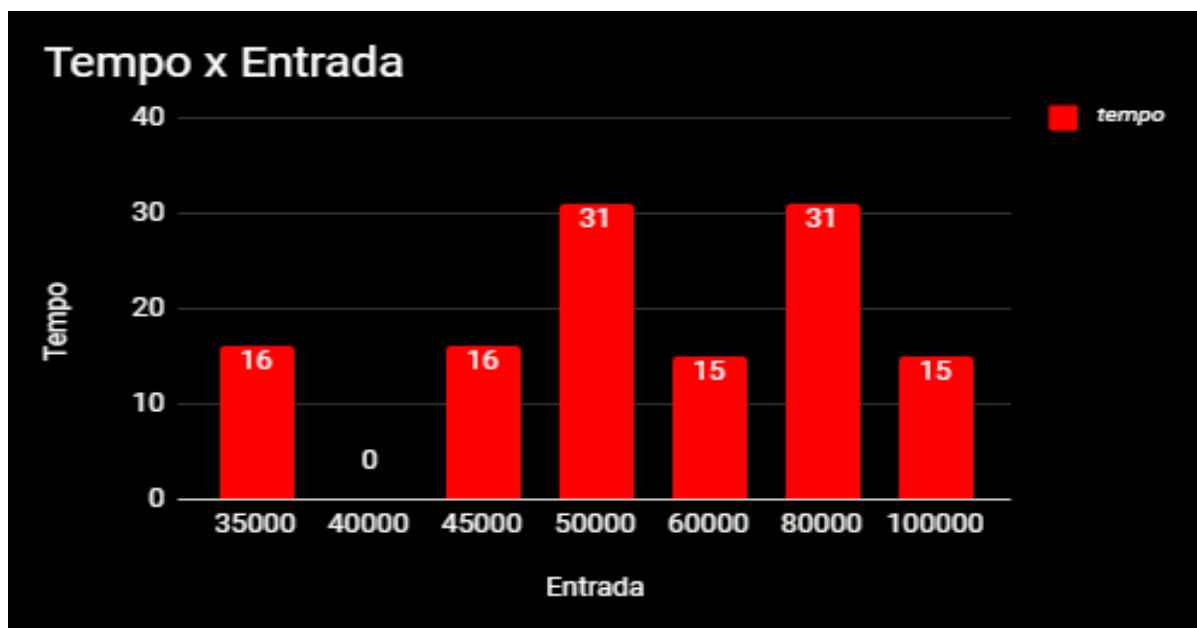


Figura 104- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Selectionsort

Entrada	tempo
1000	15
2000	0
3000	16
4000	32
5000	62
6000	78

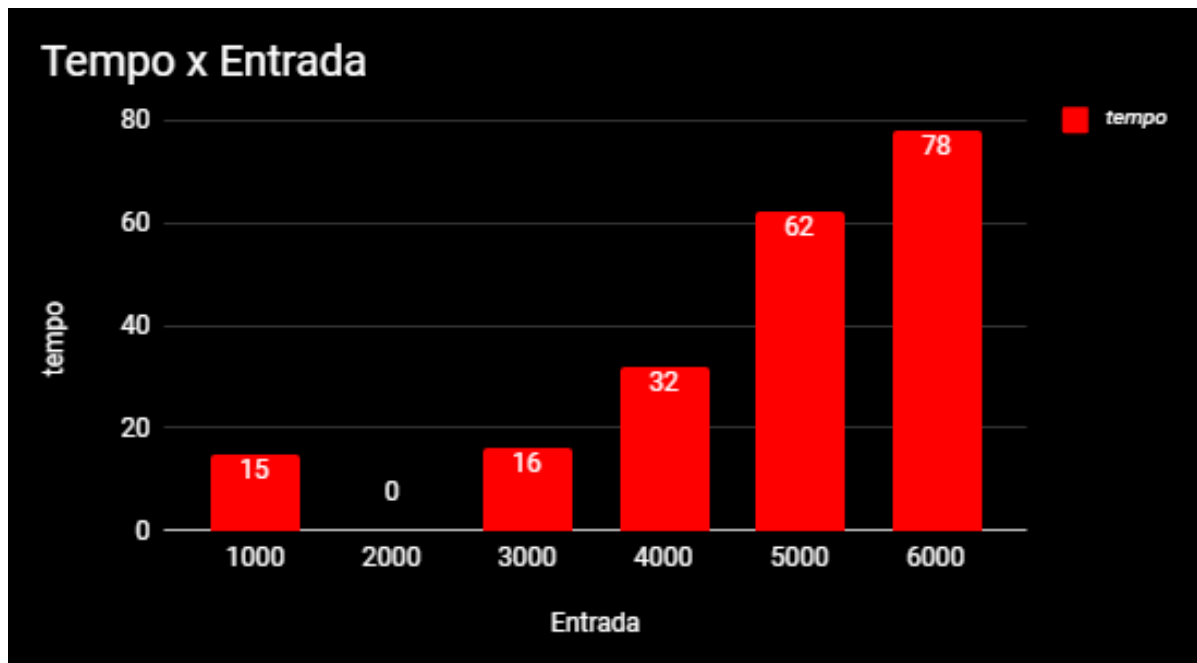


Figura 105- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.

-Shellsort

Entrada	tempo
1000	15
2000	0
3000	16
4000	0
5000	0
6000	0

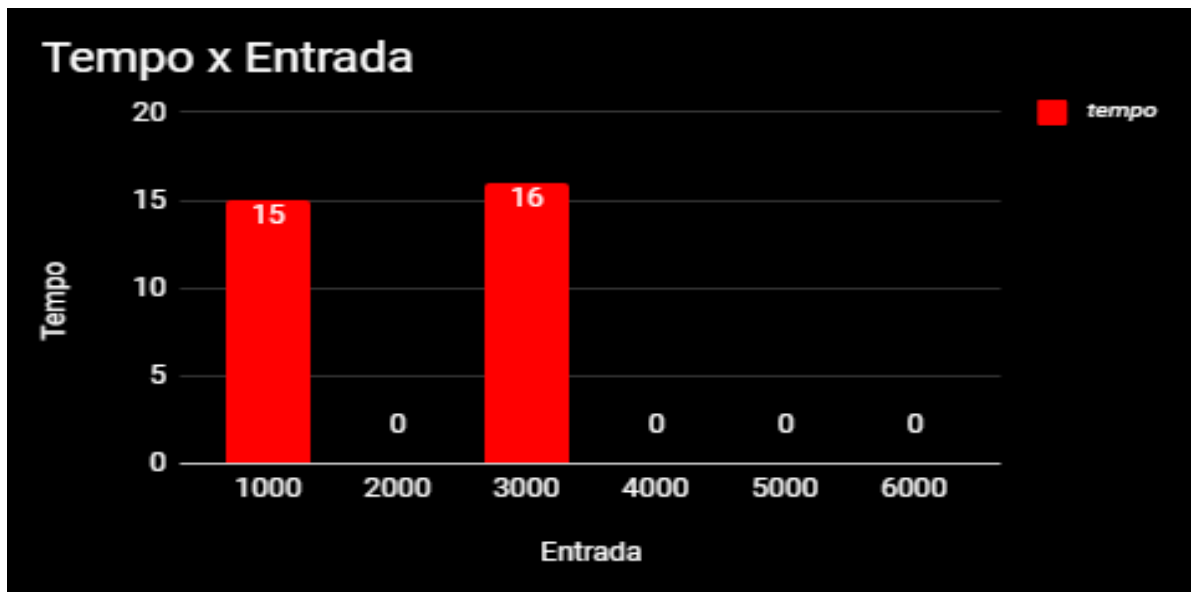


Figura 106- Gráfico com a relação entre o volume de entrada de dados e o tempo de ordenação.



## DOCUMENTO DO PROGRAMA (MANUAL DO USUÁRIO)

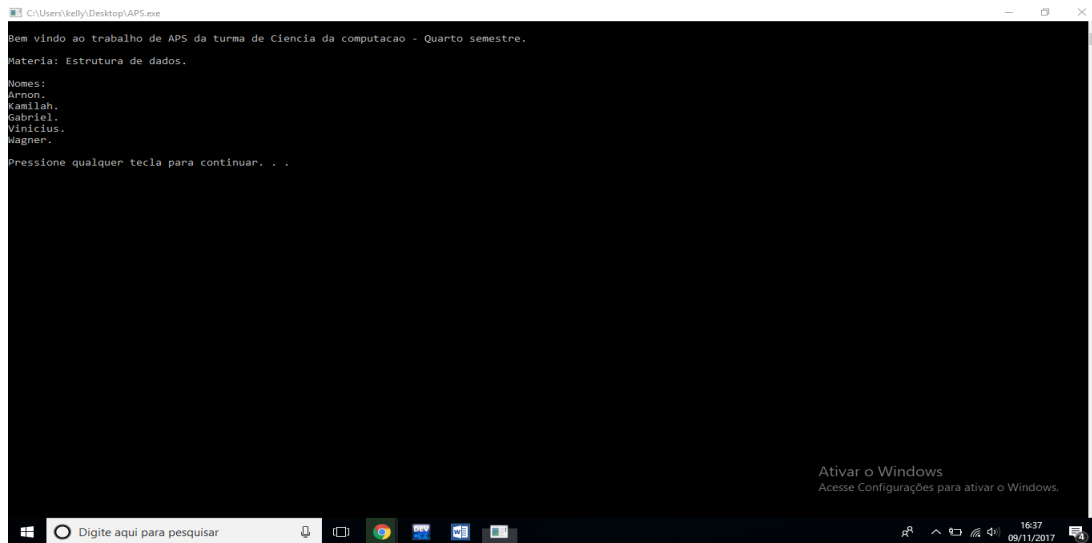


Imagem 01 – Tela inicial do programa.

Nessa tela exibimos uma mensagem de boas vindas ao usuário do programa.

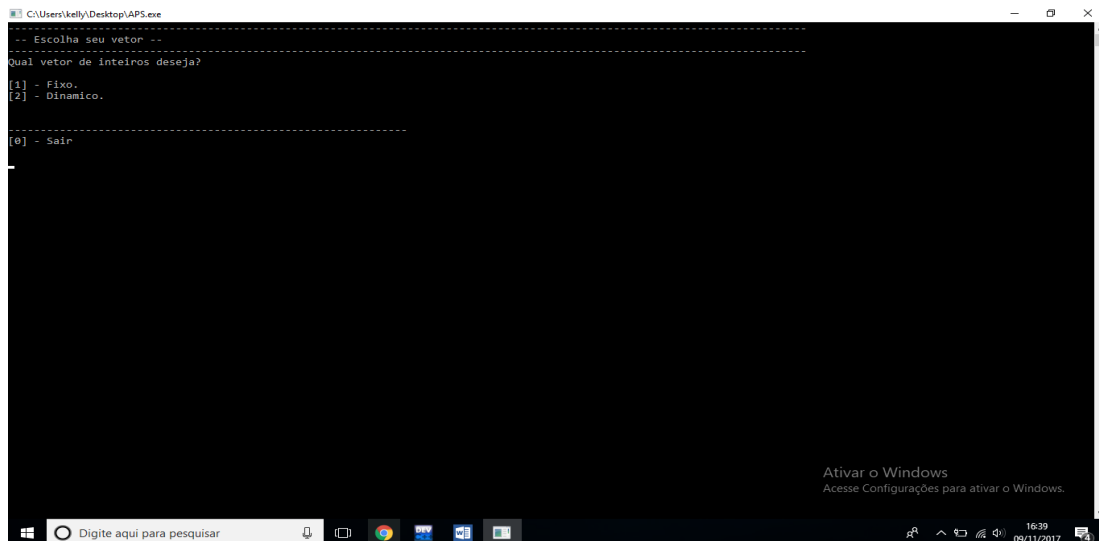


Imagem 02 – Tela escolha seu vetor.

Nessa tela o usuário escolhe que tipo de vetor deseja, fixo ou dinâmico.

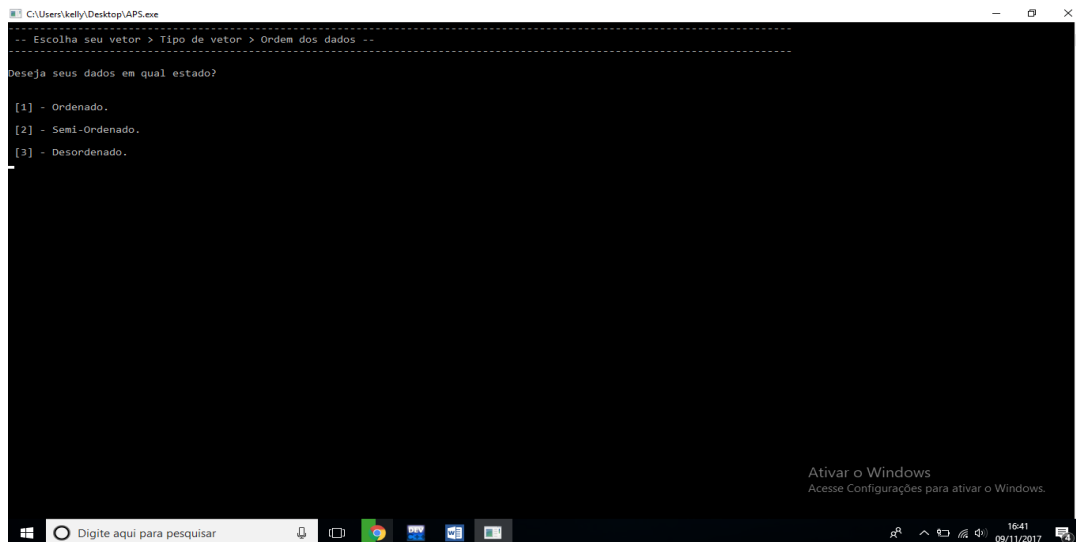


Imagem 03 – Tela estado.

Nessa tela o usuário escolhe como deseja que seus dados fiquem, ordenados, semi-ordenados ou desordenados.

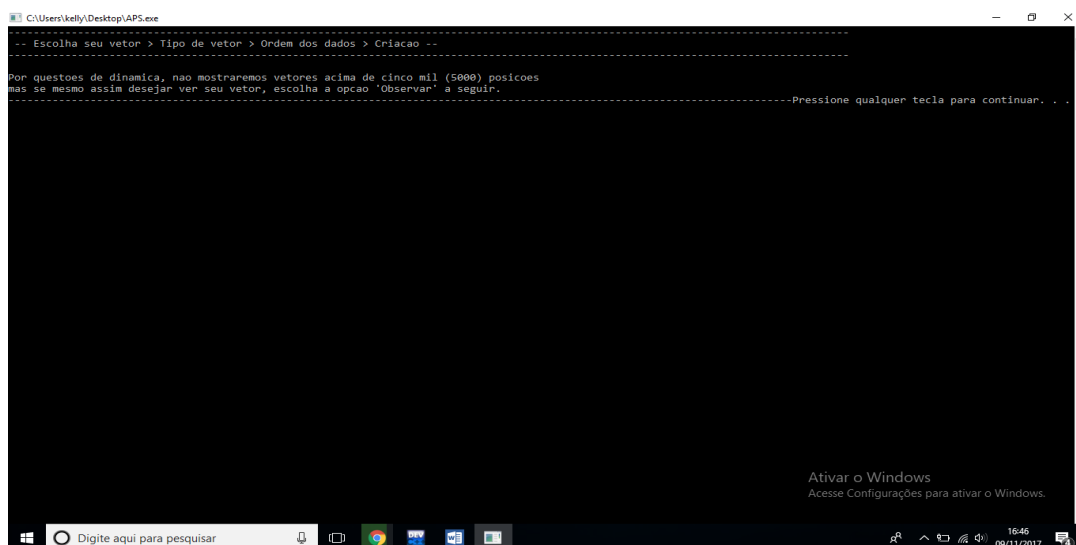


Imagem 04 – Exibe uma mensagem onde indica ao usuário que por questões de dinâmica não mostraremos vetores acima de 5000 posições, porém se o mesmo insistir em ver é só escolher a opção "observar" na tela a seguir.

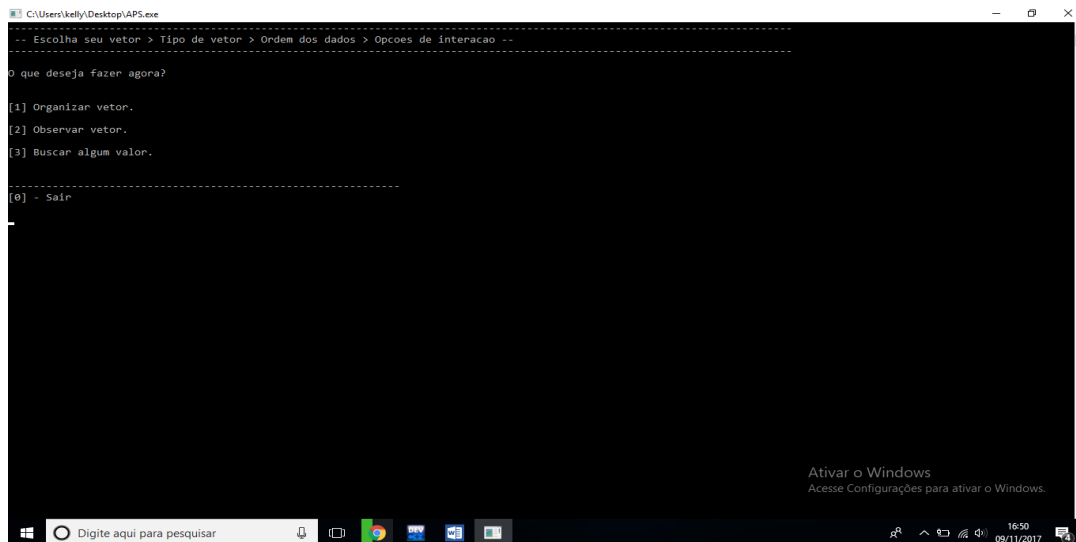


Imagem 05 – Tela o que deseja fazer agora?

Nessa tela o usuário tem a opção de escolher se deseja organizar o vetor, observar o vetor ou buscar algum valor dentro do vetor.

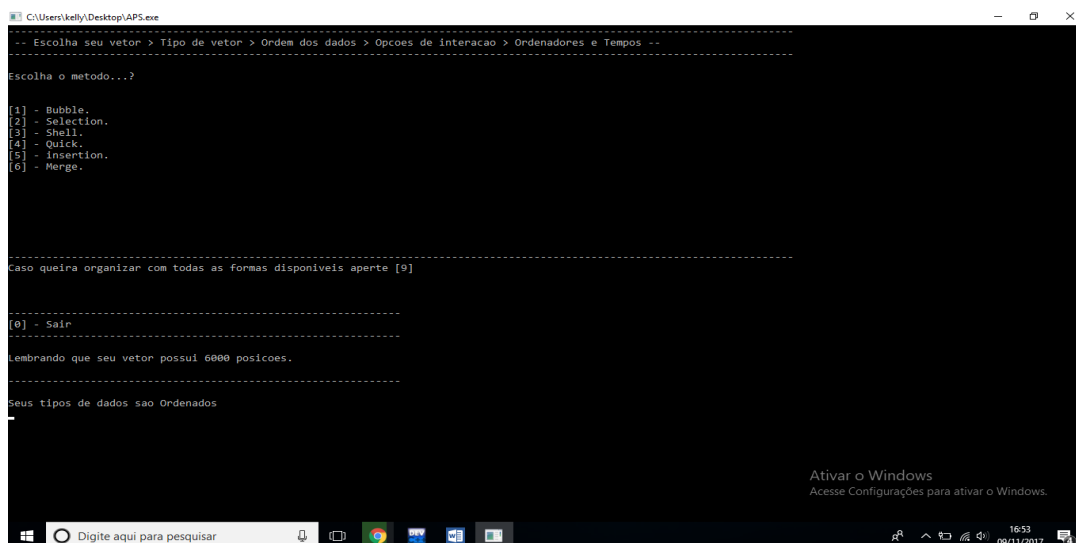


Imagem 06- Tela Organizar vetor

Nessa tela o usuário escolhe qual tipo de método quer usar para organizar o seu vetor.

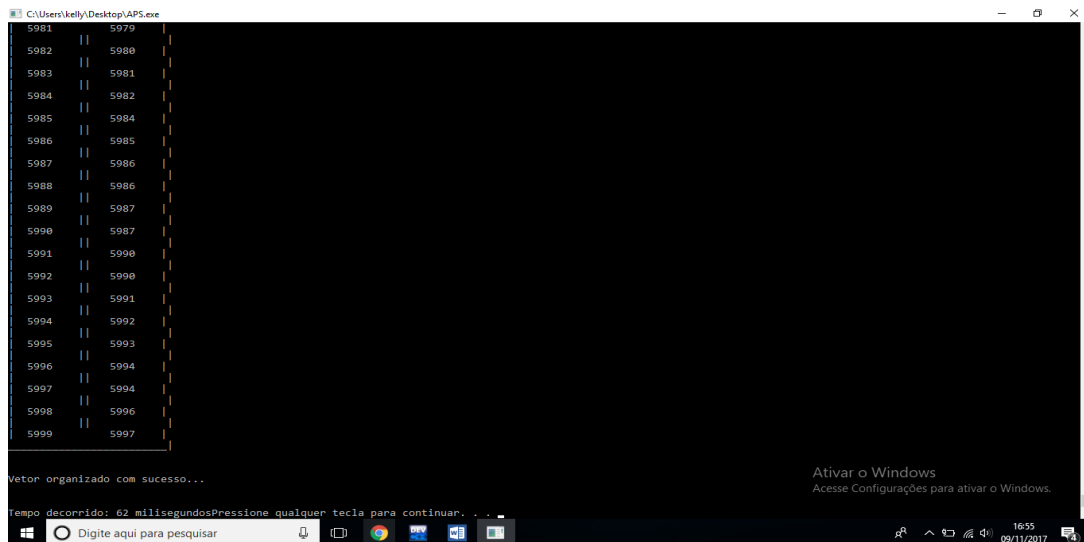


Imagem 07 – Mostra a organização do vetor com o método escolhido e o tempo percorrido para a organização dos dados.

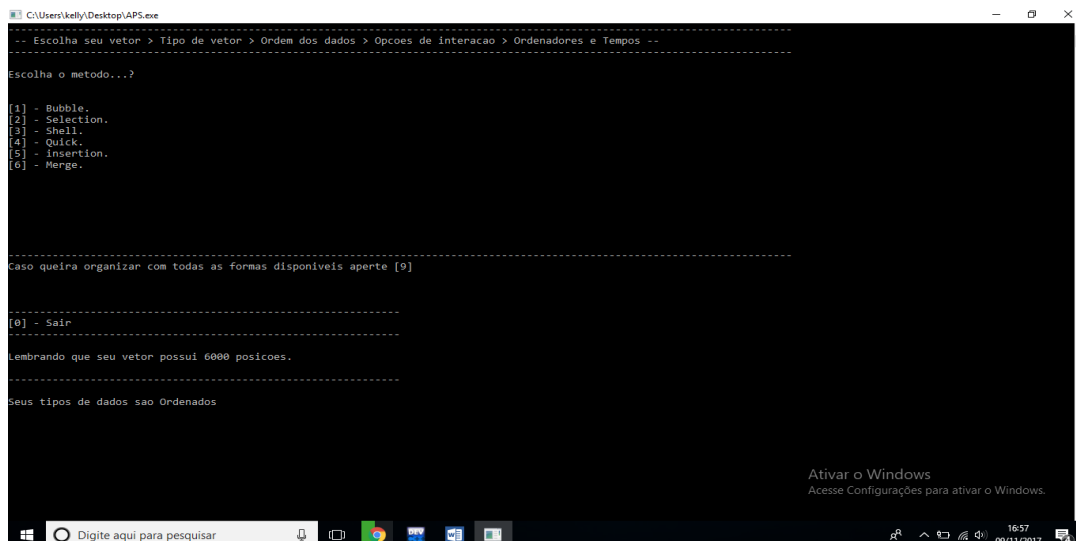


Imagem 08 – Tela anterior (escolha seu método)

Logo após a organização dos dados com determinado método escolhido, o programa dá a opção ao usuário dele reorganizar os seus dados com todos os métodos disponíveis para fazer uma comparação entre eles e verificar qual método é o mais eficiente para o volume de dados escolhido. Para isso o usuário tem que escolher a opção 9.

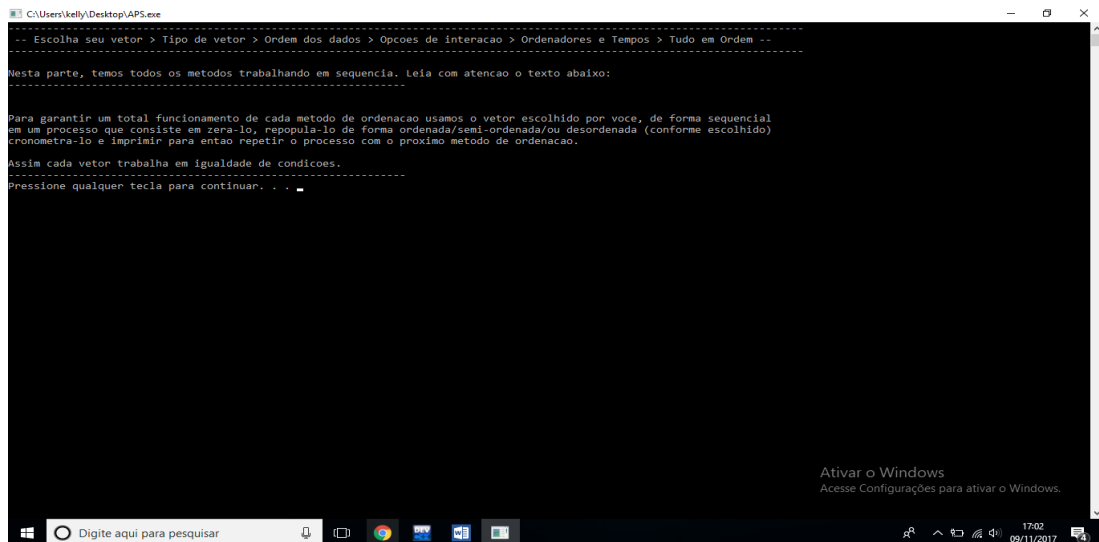


Imagem 09 – Mensagem onde indica ao usuário o funcionamento do programa e que todos os métodos trabalham em igualdade de condições.

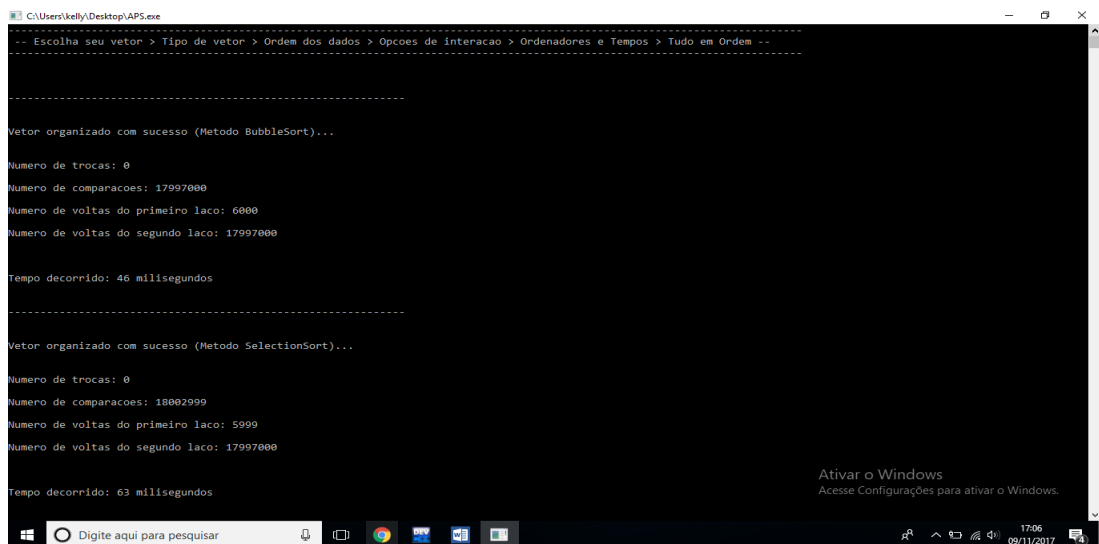


Imagem 10 – Tela de comparações entre os métodos.

Esta tela mostra e compara os métodos de ordenação em relação ao número de trocas feitas, número de comparações feitas, números voltas do primeiro laço do vetor, números de voltas do segundo laço do vetor e tempo decorrido de cada método.

```
C:\Users\kelly\Desktop\APS.exe
-- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes de interacao > Ordenadores e Tempos --
-----
Escolha o metodo...?
[1] - Bubble.
[2] - Selection.
[3] - Shell.
[4] - Quick.
[5] - Insertion.
[6] - Merge.

-----
Caso queira organizar com todas as formas disponiveis aperte [9]
-----
[0] - Sair
-----
Lembrando que seu vetor possui 6000 posicoes.
-----
Seus tipos de dados sao Ordenados

-----
Ativar o Windows
Acesse Configuracoes para ativar o Windows.

17:10
09/11/2017
```

Imagem 11 – Tela de escolha dos métodos (tela anterior)

Ao clicar enter o usuário é levado a tela anterior para escolher se quer reorganizar os dados com um método diferente ou se quer sair do programa.

## CONSIDERAÇÕES FINAIS

Após aproximadamente quatro meses de pesquisas e aplicações neste trabalho, podemos chegar a conclusão de que, a maneira de se utilizar um método de ordenação seja ele qual for deve-se levar em consideração muitos fatores para que se possa ter um modelo de ordenação único que atenda a todas as necessidades da melhor maneira possível. Então o que podemos concluir é que não existe um método de ordenação que seja melhor ou mais rápido que outro método, existem recursos que analisando uma determinada situação podem ser mais rápidos e eficazes que outros. O que devemos perceber então é que a definição e a utilização da ferramenta é o primeiro passo e o mais importante para a escolha de uma melhor solução.

## **REFERÊNCIAS BIBLIOGRÁFICAS**

Estrutura de dados usando C – (A.Tenenbaum, Y. Langsam, M.J.Augenstein, 1995)

Linguagens de programação – (A.B.Tucker, R.E.Noonan, 2008)

Fundamentos de Estrutura de dados – (Horowitz, E.Sahni, 1987)



## PROGRAMA EXECUTÁVEL E CÓDIGO FONTE (COMENTADO)

```
// importações;

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <locale.h>


//declaração de esc -> variavel global para auxilio da opção 9 da ordenação de
vetores, que, para funcionar corretamente, precisa saber qual forma de dados o

//usuario escolheu.

int esc = 0;


//relatorios (Declarações);

void relatorio (int comp, int cont);

void relatorio (int comp, int cont, int lac1);

void relatorio (int comp, int cont, int lac1, int lac2);

void relatorio (int comp, int cont, int lac1, int lac2, int lac3);

void relatorioShell (int comp, int cont, int lac1, int lac2, int lac3);


//Trabalhadores (Declarações);

void pop_ordm (int*vetor,int tam);

void pop_semi (int*vetor,int tam);

void pop_deso (int*vetor,int tam);
```

```
void mostrar (int*vet, int tam);
```

```
void zerar (int*veto, int tam);
```

```
//Buscadores(Declarações);
```

```
void busca_Binaria (int *veto, int tam);
```

```
void busca_Sequencial(int*veto, int tam);
```

```
//Auxiliadores (Declarações);
```

```
int semi();
```

```
int dinamico();
```

```
int menu();
```

```
void vetOrdem(int*veto, int tam);
```

```
// Ordenadores (Declarações);
```

```
void bubblesort(int*veto,int tam);
```

```
void shellSort(int*veto,int tam);
```

```
void selectionSort(int*veto,int tam);
```

```
void quickSort(int*veto, int left, int right);
```

```
void mergeSort();
```

```
void insertionSort(int*veto,int tam);
```

```
void Merge(int *veto, int ini, int meio, int fim, int *vetAux);
```

```
void Sort(int *veto, int inicio, int fim, int *vetAux);
```

```
void MergeSort(int *veto, int tam);
```

```
void qkSt(int*veto, int left, int right); // <- ordenador especifico para uso da função de  
semi-ordenação!
```

```
//inicio.
```

```
int main() {
```

```
    int gat = 0; // como o nome sugere, "gat" é o gatilho para 'navegação/verificavel  
de escolha' dos Switch;
```

```
    int tam = 6000; // valor de tam, define o tamanho do vetor no caso do "FIXO";
```

```
    //iniciando o vetor;
```

```
    int vetor[tam]; // criação do vetor, usando o tam como tamanho (tam logicamente  
e int, o vetor? um vetor de inteiros.);
```

```
    //iniciando controladores de contagem de tempo (padronizar e agilizar);
```

```
    int inicio,fim,tempo; //(Declaração das variaveis de controle de tempo;
```

```
    /*
```

```
        Apresentação do trabalho, de forma resumida, com os nomes dos e integrantes  
do grupo, o curso e o semestre;
```

```
    */
```

```
    printf("\nBem vindo ao trabalho de APS da turma de Ciencia da computacao -  
Quarto semestre. \n\n");
```

```
    printf("Materia: Estrutura de dados.\n\n");
```

```
    printf("Nomes: \nArnon.\nKamilah.\nGabriel.\nVinicius.\nWagner.\n\n");
```

```
    system("\n\n\n pause");
```

```
    system("cls");
```

```
do{ //começo da implementação de estrutura de escolhas;
```

No começo tínhamos um grande problema de implementar os Switch's como estrutura de escolha, sem o devido tratamento

eles se tornam uma fonte confiável de bugs e erros no código, como meta de primeira etapa de refinamento,

implementamos testes logicos em alguns pontos para restringir o usuario a escolhas dentro da proposta do programa

assim, em um menu com "Escola [1], [2] ou [3] o usuario não sera capaz de escolher 27 e continuar normalmente.

a propria aplicação, o indicara que este numero não e aceito, e mostrara novamente a ultima tela/menu que

o usuario estava;

**\* /**

```
//começo da aplicação...;
```

```
printf("-----  
-----");
```

```
printf("\n -- Escolha seu vetor --\n");
```

```
printf("-----\n");
```

```
printf("Qual vetor de inteiros deseja? \n\n[1] - Fixo.\n[2] - Dinamico.\n\n\n-----  
-----\n[0] - Sair\n\n");    // primeira escolha do  
programa;
```

```
scanf("%i",&gat); // usando o gat para a primeira escolha;
```

```
// sobre escolhas por switch.;
```

/\*

As melhores para agirem como menus, claro, se tratadas cuidadosamente, e bem aplicadas;

```
*/
```

```
switch (gat) //usando valor do 'gat' para escolha;
```

```
{
```

```
    case 1: // vetor fixo, nada mais faz que, pegar o valor de tam declarado na sua criação e usalo (puro) como tamanho do vetor;
```

```
        zerar(vetor,tam);
```

```
    /*
```

Aqui, passamos por parametro o vetor e o seu tamanho, para a função que, percorre o vetor, setando o valor 0 em

cada posição, motivo disso? eliminar lixo de memoria. podiamos simplesmente chamar a proxima função tambem

ela ja colocaria os valores aleatorios em cada posição, assim, subistituindo o lixo ou valores ja existentes,

maaasss... zerar, e a forma mais segura de evitar erros, e preparar o vetor para uso;

```
*/
```

```
vetOrdem(vetor,tam); // chamndno o metodo que, dara ao usuario a liberdade de escolher qual tipo de dados deseja dentro do seu vetor;
```

```
if (tam < 5000) // em cima!! vetOrdem como teste...
```

```
{ //Teste logico para exibição de vetores...
```

```
    //quando criamos um vetor muito longo, podemos perder minutos so "imprimindo" o mesmo apos sua criação
```

```
    //para evitar isso, foi retirada a obrigatoriedade; o usuario so imprimira seu vetor se quiser por meio da opção "Observar vetor";
```

```
//isto deixa as coisas mais dinamicas;
```

```
    mostrar(vetor,tam);    // aqui, temos uma função simples composta  
de prnt's.f's - para informações, de uma olhada na função;
```

```
    system("pause");
```

```
    system("cls");
```

```
    } else {
```

```
        system("cls");
```

```
        printf("-----  
-----");
```

```
        printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >  
Criacao --\n");
```

```
        printf("-----  
-----\n");
```

```
        printf("\nPor questoes de dinamica, nao mostraremos vetores acima de  
cinco mil (5000) posicoes \nmas se mesmo assim desejar ver seu vetor, escolha a  
opcao 'Observar' a seguir. \n");
```

```
        printf("-----  
-----");
```

```
        system("\n\npause");    // pausando a aplicação, para que o usuario  
possa ver oque esta acontecendo;
```

```
        system("cls");
```

```
    }
```

```
    gat = 0;
```

```

        break;

//-----

    case 2:

        tam = 0;

        // parte do vetor dinamico...

        /*

            Aqui, damos ao usuario a escolha de selecionar o
            tamanho do tam, assim, escolhendo o tamanho do vetor com qual

            o mesmo queira interagir, logicamente, não se trata
            de um vetor dinamico, esta mais para

            um vetor customizavel.

        */

        tam = (int)dinamico(); // para evitar algum tipo de erro, convertemos
        o valor recebido da função dinamico em int;

        vetor[tam]; // vetor recebe tam, com o tamanho especifico;

        zerar(vetor,tam); // zera o mesmo;

        vetOrdem(vetor,tam); // e pergunta ao usuario qual tipo de dados
        o mesmo deseja;

        if (tam < 5000)

        {

            mostrar(vetor,tam); // aqui, temos uma função simples composta de
            prnt's.f's - para informações, de uma olhada na função;

            system("pause");

            system("cls");

```

```

        } else {

            system("cls");

            printf("-----\n");

            printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Criacao --\n");

            printf("-----\n");

            printf("\nPor questoes de dinamica, nao mostraremos vetores acima de
cinco mil (5000) posicoes \nmas se mesmo assim desejar ver seu vetor, escolha a
opcao 'Observar' a seguir. \n");

            printf("-----\n");

            system("\n\npause"); // pausando a aplicação, para que o usuario
possa ver oque esta acontecendo;

            system("cls");

            }

            gat = 0;

            break;

//-----

            case 0:

                exit(1); // Opção para fechar a aplicação;

                break;

//-----

            default: // Aqui, casso o usuario tente digitar uma opção difente das dadas
pela aplicação, a mensagem abaixo sera exibida, e o codigo sera repetido

```



```

        // Pelo While.

        printf("Opcao invalida.");

        system("Pause");

        system("cls");

        break;

//-----

    }

}while(gat != 0);

do{

    // Proximo passo da interaçõa com o usuario, aqui, mostramos para
    que nosso programa foi feito! (literalmente :V)

    gat = 0;

    printf("-----
-----");

    printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao --\n");

    printf("-----
-----\n");

    printf("\nO que deseja fazer agora? \n\n[1] Organizar vetor. \n\n[2]
Observar vetor. \n\n[3] Buscar algum valor.\n\n-----
-----\n[0] - Sair\n\n");

    scanf("%i", &gat);

    switch (gat) //usando valor do 'gat' para escolha.

    {

        case 1:

            goto protect; // usando goto para avançar a aplicacao;

```

```

        break;

//-----

    case 2:

        system("cls");

        printf("-----
-----");

        printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes
de interacao > Observador --\n");

        printf("-----
-----\n\n");

        mostrar(vetor,tam); // chamando a função para imprimir o vetor, passamos
para o mesmo, o proprio vetor, e seu tamanho;

        system("pause");

        system("cls");

        break;

//-----

    case 3: // Opções de busca no vetor;

        system("cls");

        printf("-----
-----");

        printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes
de interacao > Buscador --\n");

        printf("-----
-----\n\n");

        printf("Deseja qual tipo de busca?\n\n");

        printf("\n[1] - Binaria. \n[2] - Sequencial.\n");

        scanf("%i", &gat);

```

```

switch (gat)
{
    case 1:
        busca_Binaria(vetor,tam); // opção binaria, (mais infos no metodo);
        break;

    case 2:
        busca_Sequencial(vetor,tam); // opção sequencial, (mais infos no
metodo);

        break;

    default:
        printf("\nOpcao invalida. ");
        system("Pause");
        system("cls");
}

break;

//-----

case 0:
    exit(1);
    break;

//-----

default:
    printf("Opcao invalida, ");
    system("Pause");
    system("cls");

```

```

        break;

//-----

    }

}while(gat != 0);

        gat = 0; // zerando o gat... importante para o uso dessa "tecnica" de
usar ele como cursor e sempre zera-lo apos o uso.

    do{
protect:

        system("cls"); // limpando o console.

        printf("-----
-----");

        printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos --\n");

        printf("-----
-----\n");

        printf("\nEscolha o metodo...? \n\n[1] - Bubble.\n[2] - Selection.\n[3]
- Shell.\n[4] - Quick.\n[5] - insertion.\n[6] - Merge.\n\n");

        printf("\n\n\n\n\n-----
-----\n");

        printf("Caso queira organizar com todas as formas disponiveis
aperte [9]\n\n");

        printf("\n\n-----\n[0]      -
Sair");

        printf("\n-----\n");

        printf("\nLembrando que seu vetor possui %i posicoes.",tam); //
mostrando quantas posicoes o vetor possui.

        printf("\n\n-----\n");

```

```
switch (esc) // mostrando qual tipo de dados foi escolhido;
```

```
{
```

```
case 1:
```

```
    printf("\nSeus tipos de dados sao Ordenados\n");
```

```
    break;
```

```
case 2:
```

```
    printf("\nSeus tipos de dados sao Semi-ordenados\n");
```

```
    break;
```

```
case 3:
```

```
    printf("\nSeus tipos de dados sao Desordenados\n");
```

```
    break;
```

```
}
```

```
scanf("%i",&gat);
```

```
switch (gat)
```

```
{
```

```
//-----
```

```
-----
```

```
case 1:
```

```
system("cls");
```

```
printf("-----  
-----");
```

```
printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >  
Opcoes de interacao > Ordenadores e Tempos > Bubble Sort --\n");
```

```

printf("-----\n");
-----\n");

inicio = GetTickCount(); // inicio do cronometro

bubblesort(vetor,tam);    // chamando o metodo de ordenação

Bubblesort

fim = GetTickCount();    // fim do cronometro;

mostrar(vetor,tam);      // imprimindo o vetor.

tempo = fim-inicio;

printf("\n\nVetor organizado com sucesso...\n");

printf("\n\nTempo decorrido: %d milisegundos",tempo);

system("pause");

system("cls");

break;

//-----
-----

case 2:

system("cls");

printf("-----\n");
-----");

printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Selection Sort --\n");

```

```

printf("-----\n");

-----\n");

    inicio = GetTickCount();

    selectionSort(vetor,tam);

    fim = GetTickCount();

    mostrar(vetor,tam);

    tempo = fim-inicio;

    printf("\n\nVetor organizado com sucesso...\n");

    printf("\n\nTempo decorrido: %d milisegundos",tempo);

    system("pause");

    system("cls");

    break;

//-----

-----

    case 3:

        system("cls");

        printf("-----\n");

        -----");

        printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Shell Sort --\n");

```

```

printf("-----\n");
-----\n");

inicio = GetTickCount();

shellSort(vetor,tam);

fim = GetTickCount();


mostrar(vetor,tam);


tempo = fim-inicio;


printf("\n\nVetor organizado com sucesso...\n");

printf("\n\nTempo decorrido: %d milisegundos",tempo);

system("pause");


system("cls");

break;

//-----
-----

case 4:

system("cls");

printf("-----\n");
-----");

printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Quick Sort --\n");

printf("-----\n");
-----\n");

```



```

        inicio = GetTickCount();

        quickSort(vetor,0,tam-1);

        fim = GetTickCount();


        mostrar(vetor,tam);


        tempo = fim-inicio;


        printf("\n\nVetor organizado com sucesso...\n");

        printf("\n\nTempo decorrido: %d milisegundos",tempo);

        system("pause");


        system("cls");

        break;

//-----
-----

        case 5:

            system("cls");

            printf("-----
            -----");

            printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Insertion Sort --\n");

            printf("-----
            -----\n");

            inicio = GetTickCount();

            insertionSort(vetor,tam);

```

```

        fim = GetTickCount();

        mostrar(vetor,tam);

        tempo = fim-inicio;

        printf("\n\nVetor organizado com sucesso...\n");
        printf("\n\nTempo decorrido: %d milisegundos",tempo);
        system("pause");

        system("cls");
        break;

//-----
-----

        case 6:

            system("cls");

            printf("-----
            -----");

            printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Merge Sort --\n");

            printf("-----
            -----\n");

        inicio = GetTickCount();

        MergeSort(vetor,tam);

        fim = GetTickCount();

```

```
mostrar(vetor,tam);
```

```
tempo = fim-inicio;
```

```
printf("\n\nVetor organizado com sucesso...\n");
```

```
printf("\n\nTempo decorrido: %d milisegundos",tempo);
```

```
system("pause");
```

```
system("cls");
```

```
break;
```

```
//-----
```

```
-----
```

```
case 9: // chamando todos os metodos de organizaçao em cadeia
```

```
system("cls");
```

```
printf("-----
```

```
-----");
```

```
printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >  
Opcoes de interacao > Ordenadores e Tempos > Tudo em Ordem --\n");
```

```
printf("-----
```

```
-----\n");
```

```
printf ("\nNesta parte, temos todos os metodos trabalhando em  
sequencia. Leia com atencao o texto abaixo:");
```

```
printf("\n-----\n");
```

```
printf("\n\nPara garantir um total funcionamento de cada metodo de  
ordenacao usamos o vetor escolhido por voce, de forma sequencial \nem um processo
```

que consiste em zera-lo, repopula-lo de forma ordenada/semi-ordenada/ou desordenada (conforme escolhido)\ncronometra-lo e imprimir para entao repetir o processo com o proximo metodo de ordenacao.");

```
printf("\n\nAssim cada vetor trabalha em igualdade de condicoes.");

printf("\n-----\n");

system("pause");


system("cls"); // limpando o console.

printf("-----
-----");

printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados >
Opcoes de interacao > Ordenadores e Tempos > Tudo em Ordem --\n");

printf("-----
-----\n");

// -----Organizando pelo BubbleSort (Usando vetor ja criado)-----
-----

printf("\n\n\n-----\n");

printf("\n\nVetor      organizado      com      sucesso      (Metodo
BubbleSort)...\n");

switch (esc)
{
    case 1:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_ordm(vetor,tam);            // re-populamos de forma aleatoria

        break;

    case 2:
```

```

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_semi(vetor,tam);            // re-populamos de forma semi-
ordenada

        break;

    case 3:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_deso(vetor,tam);            // re-populamos de forma
desordenada

        break;

    }

    // que poderiam haver nele, estando ordenados ou
não

    // Na sequencia re-populamos ele.

    inicio = GetTickCount();            // chamada inicial do cronometro.

    bubblesort(vetor,tam);              // metodo de organização.

    fim = GetTickCount();                // chamada final do vetor.

    tempo = fim-inicio;

    printf("\n\nTempo decorrido: %d milisegundos",tempo);

    // -----Organizando pelo SelectionSort (Zerando o vetor ja
organizado, e, re-preenchendo-o)-----

    printf("\n\n\n-----\n");

    printf("\n\nVetor    organizado    com    sucesso    (Metodo
SelectionSort)...\n");

    switch (esc)

```

```

{
    case 1:
        zerar(vetor,tam);                // Primeiro zeramos o vetor,
        assim eliminamos qualquer sequencia de numeros

        pop_ordm(vetor,tam);            // re-populamos de forma aleatoria

        break;

    case 2:
        zerar(vetor,tam);                // Primeiro zeramos o vetor,
        assim eliminamos qualquer sequencia de numeros

        pop_semi(vetor,tam);            // re-populamos de forma semi-
        ordenada

        break;

    case 3:
        zerar(vetor,tam);                // Primeiro zeramos o vetor,
        assim eliminamos qualquer sequencia de numeros

        pop_deso(vetor,tam);            // re-populamos de forma
        desordenada

        break;
}

```

```

inicio = GetTickCount();

```

```

selectionSort(vetor,tam);

```

```

fim = GetTickCount();

```

```

tempo = fim-inicio;

```

```

printf("\n\nTempo decorrido: %d milisegundos",tempo);

```

```
// -----Organizando pelo ShellSort (Zerando o vetor ja organizado,  
e, re-preenchendo-o)-----
```

```
printf("\n\n-----\n");
```

```
printf("\n\nVetor organizado com sucesso (Metodo ShellSort...\n");
```

```
switch (esc)
```

```
{
```

```
case 1:
```

```
    zerar(vetor,tam);                // Primeiro zeramos o vetor,  
assim eliminamos qualquer sequencia de numeros
```

```
    pop_ordm(vetor,tam);            // re-populamos de forma aleatoria
```

```
    break;
```

```
case 2:
```

```
    zerar(vetor,tam);                // Primeiro zeramos o vetor,  
assim eliminamos qualquer sequencia de numeros
```

```
    pop_semi(vetor,tam);            // re-populamos de forma semi-  
ordenada
```

```
    break;
```

```
case 3:
```

```
    zerar(vetor,tam);                // Primeiro zeramos o vetor,  
assim eliminamos qualquer sequencia de numeros
```

```
    pop_deso(vetor,tam);            // re-populamos de forma  
desordenada
```

```
    break;
```

```
}
```

```
inicio = GetTickCount();
```

```
shellSort(vetor,tam);
```

```

fim = GetTickCount();

tempo = fim-inicio;

printf("\n\nTempo decorrido: %d milisegundos",tempo);

// -----Organizando pelo QuickISort (Zerando o vetor ja
organizado, e, re-preenchendo-o)-----

printf("\n\n\n-----\n");

printf("\n\nVetor organizado com sucesso (Metodo QuickSort)...\n");

switch (esc)
{
    case 1:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_ordm(vetor,tam);            // re-populamos de forma aleatoria

        break;

    case 2:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_semi(vetor,tam);            // re-populamos de forma semi-
ordenada

        break;

    case 3:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_deso(vetor,tam);            // re-populamos de forma
desordenada

        break;
}

```



```

    }

    inicio = GetTickCount();

    quickSort(vetor,0,tam-1);

    fim = GetTickCount();

    tempo = fim-inicio;

    printf("\n\nTempo decorrido: %d milisegundos",tempo);

    // -----Organizando pelo QuickISort (Zerando o vetor ja
organizado, e, re-preenchendo-o)-----

    printf("\n\n\n-----\n");

    printf("\n\nVetor      organizado      com      sucesso      (Metodo
insertionSort)...\n");

    switch (esc)
    {
        case 1:

            zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

            pop_ordm(vetor,tam);            // re-populamos de forma aleatoria

            break;

        case 2:

            zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

            pop_semi(vetor,tam);            // re-populamos de forma semi-
ordenada

            break;

```

```

        case 3:

            zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

            pop_deso(vetor,tam);            // re-populamos de forma
desordenada

            break;

        }

        inicio = GetTickCount();

        insertionSort(vetor,tam);

        fim = GetTickCount();

        tempo = fim-inicio;

        printf("\n\nTempo decorrido: %d milisegundos",tempo);

        // -----Organizando pelo MergeSort (Zerando o vetor ja
organizado, e, re-preenchendo-o)-----

        printf("\n\n\n-----\n");

        printf("\n\nVetor organizado com sucesso (Metodo mergeSort)...\n");

        switch (esc)

        {

            case 1:

                zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

                pop_ordm(vetor,tam);            // re-populamos de forma aleatoria

                break;

            case 2:

```

```

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_semi(vetor,tam);            // re-populamos de forma semi-
ordenada

        break;

    case 3:

        zerar(vetor,tam);                // Primeiro zeramos o vetor,
assim eliminamos qualquer sequencia de numeros

        pop_deso(vetor,tam);            // re-populamos de forma
desordenada

        break;

    }

    inicio = GetTickCount();

    MergeSort(vetor,tam);

    fim = GetTickCount();

    tempo = fim-inicio;

    printf("\n\nTempo decorrido: %d milisegundos\n\n",tempo);

    system("\n\npause");

    break;

//-----
-----

    case 0: // Opcao para sair da aplicação

    exit(1);

```

```

        break;

//-----
-----

        default :

        printf("\nOpcao invalida.\n");

        system("\n\npause");

        system("cls");

goto protect;

        break;

    }

} while (gat != 0);

}

/*

```

Relatorios são usados diretamente e em conjunto com os metodos de ordeção, tanto que, somente eles os chamam, aqui, temos, 5 metodos

mas, como ja visto em POO, cada qual possui uma assinatura, porque disso?

cada metodo de ordenação e unico, alguns simples, emglobam apenas um laço, uma troca, e uma serie do comparações

outros... aaah... outros não... e e por isso que temos cada qual com "comp" (Comparações) e "cont" (trocas) como parametros fixos,

e as unicas coisas que almentão são os parametros responsaveis por contar as voltas de cada laço.

Mais datalhes do funcionamento no BubbleSort.

```
*/
```

```
void relatorio (int comp, int cont)
```

```
{
```

```
    printf("\n\nNumero de trocas: %i",cont);
```

```
    printf("\n\nNumero de comparacoes: %i\n\n",comp);
```

```
}
```

```
void relatorio (int comp, int cont, int lac1)
```

```
{
```

```
    printf("\n\nNumero de trocas: %i",cont);
```

```
    printf("\n\nNumero de comparacoes: %i",comp);
```

```
    printf("\n\nNumero de voltas do primeiro laco: %i\n\n",lac1);
```

```
}
```

```
void relatorio (int comp, int cont, int lac1, int lac2)
```

```
{
```

```
    printf("\n\nNumero de trocas: %i",cont);
```

```
    printf("\n\nNumero de comparacoes: %i",comp);
```

```
    printf("\n\nNumero de voltas do primeiro laco: %i",lac1);
```

```
    printf("\n\nNumero de voltas do segundo laco: %i\n\n",lac2);
```

```
}
```

```
void relatorio (int comp, int cont, int lac1, int lac2, int lac3)
```

```
{
```

```
    printf("\n\nNumero de trocas: %i",cont);
```

```
    printf("\n\nNumero de comparacoes: %i",comp);
```

```
    printf("\n\nNumero de voltas do primeiro laço: %i",lac1);
```

```
    printf("\n\nNumero de voltas do segundo laço: %i",lac2);
```

```

printf("\n\nNumero de voltas do terceiro laço: %i\n\n",lac3);
}

void relatorioShell (int comp, int cont, int lac1, int lac2, int lac3) //<-teste
{
    printf("\n\nNumero de trocas: %i",cont);

    printf("\n\nNumero de comparacoes: %i",comp);

    printf("\n\nNumero de Quebras do vetor: %i",lac1);

    printf("\n\nNumero de voltas do primeiro laco: %i",lac2);

    printf("\n\nNumero de Trocas no pivo: %i\n\n",lac3);
}

//-----

// Ordenadores.

void bubblesort(int*vetor,int tam)
{

    int trocas = 0, lac1 = 0, lac2 = 0, comp = 0; // declaração local das variaveis
    usadas para os trabalhos do relatorio;

    int aux = 0;

    for (int a = 0 ; a < tam; a++, lac1++){ // aqui, implementamos o laço, dentro dele,
    nossa primeira variavel de contagem que, a cada volta do mesmo

        // sera implementada em +1;

        for (int b = a + 1 ; b < tam ; b++, comp++, lac2++){ // contagem do segundo laco;

            if(vetor[a] > vetor[b]){ // momento da troca de valores do metodo booble

```

```

        aux = vetor[a];

        vetor[a] = vetor[b];

        vetor[b] = aux;

        trocas++; // tambem e o momento que a variavel "Trocas" e implementada
+1;

    }

}

}

    relatorio(comp,trocas,lac1,lac2); // chamando o relatorio, e passando por referencia
os valores acumulados duranta a execusao do bubbleSort

}

//-----

void shellSort(int*vetor,int tam) // Shell simples...

{

    int trocas = 0, lac1 = 0, lac2 = 0, lac3 = 0, comp = 0; // declaração dos contadores


    int i , j , value; // declaração dos valores usados no vetor

    int gap = 1; // auxiliar


do {

    gap = 3*gap+1;

    lac1++;

    } while(gap < tam);


do {

```

```

gap /= 3;

for(i = gap; i < tam; i++, lac2++) {

    value = vetor[i];

    j = i - gap;

    lac3++;

    while (comp++, j >= 0 && value < vetor[j]) {

        vetor[j + gap] = vetor[j];

        j -= gap;

        trocas ++;

    }

    vetor[j + gap] = value;

}

}while(gap > 1);

```

```

    relatorioShell(comp,trocas,lac1,lac2,lac3);

}

```

```

//-----

```

```

void selectionSort(int*vetor,int tam)

```

```

{

```

```

    int trocas = 0, lac1 = 0, lac2 = 0, comp = 0;

```

```

    int i, j, k, tmp, tr;

```

```

for(i = 0; i < tam-1; i++, comp++, lac1++)

```

```

{

```



```

tr = 0;

k = i;

tmp = vetor[i];

for(j = i+1; j < tam; j++, comp++, lac2++)
{
    if(vetor[j] < tmp)
    {
        k = j;

        tmp = vetor[j];

        tr = 1;
    }
}

if(tr)
{
    vetor[k] = vetor[i];
    vetor[i] = tmp;

    trocas++;
}

}

relatorio(comp,trocas,lac1,lac2);

}

//-----

void quickSort(int*vetor, int left, int right)

    // (vetor,0,tam-1) (o 0 é passado para o left de inicio,

```

```
        //porque e e novamente chamado, e ai, o left tera outro  
valor)
```

```
{
```

// quick tem um serio problema com o metodo de relatorio atual... ele e recursivo, significado? ele se chama, e isso faz com que a precisão do relatorio (Ja duvidosa) va para outra galaxia...

```
int trocas = 0, lac1 = 0, lac2 = 0, lac3 = 0, comp = 0;
```

```
int i = 0, j = 0, x = 0, y = 0;
```

```
int teste;
```

```
i = left;
```

```
j = right;
```

```
x = vetor[(left + right) / 2];
```

```
while(i <= j) {
```

```
    while(vetor[i] < x && i < right) {
```

```
        i++;
```

```
    }
```

```
    while(vetor[j] > x && j > left) {
```

```
        j--;
```

```
    }
```

```
    if(i <= j) {
```

```
        y = vetor[i];
```

```
        vetor[i] = vetor[j];
```

```
        vetor[j] = y;
```

```
        i++;
```

```

        j--;

        teste += trocas++;

    }

}

if(j > left) {

    quickSort(vetor, left, j);

}

if(i < right) {

    quickSort(vetor, i, right);

}

}

//-----

// O metodo de ordenação Merge-sort e recursivo, aqui vemos ele dividido em 3
funções, onde cada uma chama a seguinte;

// 1ª

void Merge(int *vetor, int ini, int meio, int fim, int *vetAux)

{

    int esq = ini;

    int dir = meio;

    for (int i = ini; i < fim; ++i) {

        if ((esq < meio) and ((dir >= fim) or (vetor[esq] < vetor[dir]))) {

            vetAux[i] = vetor[esq];

            ++esq;

        }

    }

}

```

```

        else {

            vetAux[i] = vetor[dir];

            ++dir;

        }

    }

//copiando o vetor de volta

    for (int i = ini; i < fim; ++i) {

        vetor[i] = vetAux[i];

    }

}

//2ª

void Sort(int *vetor, int inicio, int fim, int *vetAux)

{

    if ((fim - inicio) < 2) return;

    int meio = ((inicio + fim)/2);

    Sort(vetor, inicio, meio, vetAux);

    Sort(vetor, meio, fim, vetAux);

    Merge(vetor, inicio, meio, fim, vetAux);

}

//3ª

void MergeSort(int *vetor, int tam) //função que o usuario realmente chama

{

//criando vetor auxiliar

    int vetAux[tam];

```

```

        Sort(vetor, 0, tam, vetAux);
    }

//-----

void insertionSort(int*vetor,int tam)
{
    int trocas = 0, lac1 = 0, lac2 = 0, lac3 = 0, comp = 0;

    int i, j, tmp;

    for(i = 1; i < tam; i++)
    {
        tmp = vetor[i];
        for(j = i-1; j >= 0 && tmp < vetor[j]; j--)
        {
            vetor[j+1] = vetor[j];
        }
        vetor[j+1] = tmp;
    }
}

//-----

void qkSt(int*vetor, int left, int right) //ordenador Customizado para uso da função de
semi-ordenação!
{
    int i = 0, x = 0, y = 0;

    int j = (right / 2);

```

```

int rg = j;

int lf = left;

i = left;

x = vetor[(lf + rg) / 2];

while(i <= j) {

    while(vetor[i] < x && i < rg) {

        i++;

    }

    while(vetor[j] > x && j > lf) {

        j--;

    }

    if(i <= j) {

        y = vetor[i];

        vetor[i] = vetor[j];

        vetor[j] = y;

        i++;

        j--;

    }

}

if(j > lf) {

    quickSort(vetor, lf, j);

}

if(i < rg) {

```

```

        quickSort(vetor, i, rg);

    }

}

//-----

//Trabalhadores


//Gerando os numeros aleatorios e ja colocando-os no vetor;

void pop_deso (int*vetor,int tam)

{
    for (int i = 0 ; i < tam ; i++)

    {
        vetor[i] = rand () % tam; // aqui temos o nosso vetor sendo percorrido de i ate
tam (valor total do vetor), a coda avanço,

        // "rand" gera um numero aleatorio que vai de 0 a tam, oque
significa?

        // se o vetor tem 100 posições, teremos numeros aleatorio de 0
a 100, se forem 2000... numeros de 0 a 2000;

    }

}

void pop_semi (int*vetor,int tam)

{
    for (int i = 0 ; i < tam ; i++)

    {
        vetor[i] = rand () % tam; // aqui temos o nosso vetor sendo percorrido de i ate
tam (valor total do vetor), a coda avanço,

```

```
        // "rand" gera um numero aleatorio que vai de 0 a tam, oque  
significa?
```

```
        // se o vetor tem 100 posições, teremos numeros aleatorio de 0  
a 100, se forem 2000... numeros de 0 a 2000;
```

```
    }
```

```
    qkSt(vetor,0,tam);
```

```
}
```

```
void pop_ordm (int*vetor,int tam)
```

```
{
```

```
    for (int i = 0 ; i < tam ; i++)
```

```
    {
```

```
        vetor[i] = rand () % tam; // aqui temos o nosso vetor sendo percorrido de i ate  
tam (valor total do vetor), a cada avanço,
```

```
        // "rand" gera um numero aleatorio que vai de 0 a tam, oque  
significa?
```

```
        // se o vetor tem 100 posições, teremos numeros aleatorio de 0  
a 100, se forem 2000... numeros de 0 a 2000;
```

```
    }
```

```
    quickSort(vetor,0,tam);
```

```
}
```

```
// so para imprimir o vetor;
```

```
void mostrar (int*vetor,int tam){
```

```
    // sobre...
```



printf("\_\_\_\_\_\n"); // aqui temos o começo da impressao, importante que, como podem notar esta parte esta fora do laço, afinal, so sera impressa 1 vez

```
printf("| Posicao || valor ||\n");
```

```
printf("-----\n");
```

```
for (int i = 0 ; i < tam ; i++){
```

```
printf("      ||      |\n", i, vetor[i]);
```

printf("| %4i %5d |\n", i, vetor[i]); // impressão dos valores, esses numeros antes de "I" e "D", deixam o valor fixos... como assim?

// Ex: reservamos para impressão 4 espaços e depois mais 5 | \_ \_ \_ \_ , \_ \_ \_ \_ \_ | enquanto nossa

// nossa sequencia numerica não estourar esse espaço, tudo que estiver na frente das reservas

// não sera deslocado para frente nem para trás, muito util para criação de mini tabelas...

```
}
```

```
printf("_____| \n");
```

```
}
```

// para lidar de forma simples (Talvez não a mais indicada) como o "lixo" de memoria;

```
void zerar (int*vetor,int tam)
```

```
{
```

```
int vtm = sizeof(vetor);
```

```
for(int i = 0 ; i < vtm; i++)
```

```
{
```

```
vetor[i] = 0;
```

```

    }

}

//-----

//Buscadores.


// Busca Binaria no vetor.

void busca_Binaria (int*vetor,int tam)

{

    bool achou = false;

    int inf = 0;    // limite inferior (o primeiro índice de vetor em C é zero    )

    int sup = tam-1; // limite superior (termina em um número a menos. 0 a 9 são 10
números)

    int meio;      // divisão dos valores.

    int var = 0;   //numero buscado.

    int aux;       // auxiliar.

    system("cls");


    printf("-----
-----");

    printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes de
interacao > Buscador > Busca Binaria --\n");

    printf("-----
-----\n");


    printf("\n\nDevemos lembra-los que para a realizacao de busca binaria em vetores,
o mesmo precisa estar obrigatoriamente ordenado. \nPor este motivo, caso o mesmo

```

nao tenha sido previamente ordenado, tomamos a liberdade de ordena-lo automaticamente a seguir.\n\n\n\n");

```
printf("-----\n");
```

```
system("pause");
```

```
system("cls");          // primeiro ordenamo...
```

```
for (int a = 0 ; a < tam; a++){ // primeiro laço, vai de 0 a tam
```

```
for (int b = a + 1 ; b < tam ; b++){ // segundo laço percorre o vetor de a+1 ate  
quando o tam for menor que b
```

```
if(vetor[a] > vetor[b]){ // se o valor na posicao A for menos que o na B
```

```
    aux = vetor[a];    //trocam.
```

```
    vetor[a] = vetor[b];
```

```
    vetor[b] = aux;
```

```
}
```

```
}
```

```
}
```

```
printf("-----\n");
```

```
printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes de  
interacao > Buscador > Busca Binaria --\n");
```

```
printf("-----\n");
```

```
printf ("Digite o valor que deseja buscar\n\n");
```

```
scanf("%i",&var); // guardando o valor escolhido pelo usuario;
```

while (inf <= sup) // primeiro laco while, pega o valor a esquerda do vetor, Ex: 0 ,  
o valor a direita Ex: 15, e somaos, para

```
{ // para obter o tamanho do vetorr.  
  
    meio = (inf + sup)/2; // logo apos, divide o mesmo por 2;  
  
    if (var == vetor[meio]) // se o valor pedido pelo usuario for exatamente o do  
meio, ja de primeira veremos a msg abaixo.  
  
        printf("\nValor desejado localizado | Posicao: %i | Valor:  %i |\n\n\n",meio  
,vetor[meio], achou = true);
```

if (var < vetor[meio]) //caso não, e o valor pedido for menor que o do meio, o  
codigo diminui o valor do meio.

sup = meio-1; //Ex vetor[meio] = 5; o valor pedido foi 3... meio que estava  
na posição 4, agora diminuira 1, indo para a 3;

```
    else  
  
        inf = meio+1;  
  
    }  
  
    if (achou == false)  
  
        {  
  
            printf("-----  
-----\n");  
  
            printf("Valor nao encontrado\n");  
  
        }  
  
        system("pause");  
  
        system("cls");  
  
    }
```

// Busca Sequencial no vetor.

void busca\_Sequencial(int\*vetor, int tam) // busca sequencial... e feita de forma parecida com a ordenadora Bubble sort.

```
{ // nossa aplicacao percorrera todo o vetor, de posicao em posicao conferindo os valores,

    int var; // toda vez que achar nosso valor, contara;

    bool achou = false; // variavel booleana de controle

    system("cls"); // limpando o console

    printf("-----\n");

    printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados > Opcoes de interacao > Buscador > Busca Sequencial --\n");

    printf("-----\n");

    printf ("Digite o valor que deseja buscar\n\n");

    scanf("%i",&var); // lendo o valor a ser buscado

    for (int i = 0 ; i < tam ; i++){ // laco responsavel por correr o vetor

        if (vetor[i] == var) // condicao : vetor na posição i e igual o valor digitado?

            printf("\nValor desejado localizado em: |Posicao: %i | \n\n", i+1, vetor[i],achou = true); // caso seja, é imprimido na tela, e a bool achou

    } // se torna TRUE

    if(achou == false) // caso ele percorra todo o vetor e não ache o valor solicitado;

    {

        printf("-----\n");

        printf("\nValor nao localizado.\n\n");
```

```

    }

    system("Pause");

    system("cls");
}

//-----

//Auxiliadores

int dinamico()
{ // responsavel pelo nosso vetor "dinamico"

    nozero:

    int temp, valor; // declaramos as variaveis que vamos usar;

    system("cls");

    printf("-----\n");

    printf("\n -- Escolha seu vetor > Dinamico --\n");

    printf("-----\n");

    printf("Nesta parte, voce tem a liberdade de escolher um tamanho especifico para o seu vetor de inteiros.\n");

    printf("\nQual tamanho deseja?\n");

    scanf("%i",&temp); // temp armazena o valor que o usuario deseja para o tamanho do seu vetor;

    if (temp <= 0)

```

```

    {

        printf("\nPor Favor digite algum valor aceitavel para o seu vetor.\n");

        system("pause\n\n");

        system("cls");

        goto nozero;

    }

    int *test; // processo para verificar se o computador possui memoria para o valor
    solicitado;

    test = (int*)malloc(temp*sizeof(int)); // tentativa de reserva

    if(test!=NULL)

    {

        valor = temp; // caso de tudo certo, temos nossa variavel valor com o tamanho
        escolhido pelo usuario

    }else{

        printf("Memoria  insuficiente;  hahaha!"); // caso não aja memoria, esta
        mensagem aparecera para o usuario;

        system("pause");

        exit(1); // fechando a aplicação;

    }

    return valor;

}

void vetOrdem(int*vetor, int tam){ // Função para escolher a forma dos dados.
(ordenado / semi-ordenado / desordenado);

```

nd:

```
int *p;    // declaração do ponteiro;

int gat = 0;

system("cls");

printf("-----
-----");

printf("\n -- Escolha seu vetor > Tipo de vetor > Ordem dos dados --\n");

printf("-----
-----\n");

printf("\nDeseja seus dados em qual estado?");

printf("\n\n [1] - Ordenado.\n\n [2] - Semi-Ordenado.\n\n [3] - Desordenado.\n");

scanf("%i", &gat);

switch (gat) //usando valor do 'gat' para escolha;
{
    case 1: // primeira condição do switch;

        pop_ordm (vetor,tam); // chamamos o pop_ordem

        p = &esc; // logo em seguida, o ponteiro recebe o endereço da variavel esc;

        *p = gat; // jogamos o valor do 'menu' escolhido pelo usuario, diretamente
na variavel esc atravez do seu endereço de memoria;

        break;

    case 2:

        pop_semi (vetor,tam); // chamamos o pop_semi;

        p = &esc;    // repetimos o mesmo processo de cima, para passagem de
valor para a variavel esc;

        *p = gat;

        break;
```



```
case 3:

    pop_deso(vetor,tam); //chamamos o pop_deso.

    p = &esc;

    *p = gat;

    break;

default :

    printf("\n\nOpção invalida!"); // caso nenhuma opção valida (ou seja, de 1 a
3) seja escolhida, caímos aqui, onde avisamos o usuario, limpamos a tela,

    system("pause"); // e voltamos para o começo da funcao;

    system("cls");

    goto nd; // comando para retorno 'brusco';

}

}
```