

UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

Laboratório de estrutura de dados I

Laboratório 5

Discente: Anderson Carlos da Silva Moraes

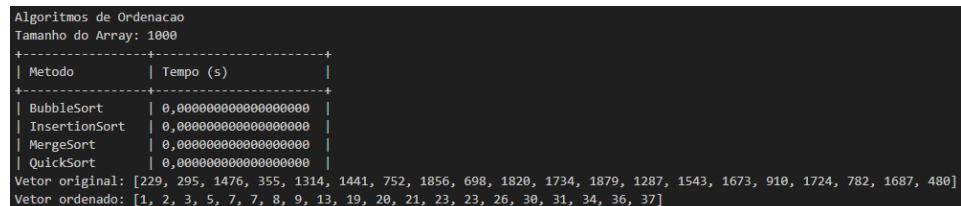
Repositório: <https://github.com/AndersonCSM/algorithm-and-data-structure/tree/main/laboratory/lista-5>

Resposta:

1. Implemente os algoritmos de ordenação vistos na disciplina (bubble, insertion, merge, quick) e compare seus tempos de execução nos seguintes cenários:

a) Array com 1000 elementos

Para o cenário de um vetor com 1000 elementos inteiros aleatórios, todos os algoritmos de ordenação foram capazes de ordená-lo dentro de um tempo satisfatório, em que a diferença de tempo de execução, não sendo a implementação capaz de diferenciar, visto na Figura 1.



```
Algoritmos de Ordenacao
Tamanho do Array: 1000
+-----+-----+
| Metodo      | Tempo (s) |
+-----+-----+
| BubbleSort  | 0,0000000000000000 |
| InsertionSort | 0,0000000000000000 |
| MergeSort   | 0,0000000000000000 |
| QuickSort   | 0,0000000000000000 |
+-----+-----+
Vetor original: [229, 295, 1476, 355, 1314, 1441, 752, 1856, 698, 1820, 1734, 1879, 1287, 1543, 1673, 910, 1724, 782, 1687, 480]
Vetor ordenado: [1, 2, 3, 5, 7, 7, 8, 9, 13, 19, 20, 21, 23, 23, 26, 30, 31, 34, 36, 37]
```

Fig. 1. Resultado dos testes para array de 1000 elementos e os primeiros elementos dos arrays. Autoria própria.

b) Array com 100000 elementos

Para o cenário de um vetor de 100000 elementos inteiros aleatórios, todos os algoritmos de ordenação foram capazes de ordenar o vetor dentro de um tempo satisfatório, contudo, mostrando diferenças no tempo de execução, no teste pontual executado, o algoritmo mais rápido foi o Mergesort, seguindo os demais na ordem de mais rápidos para os mais lentos listado abaixo e seus resultados sendo ilustrados na Figura 2.

1. Mergesort

2. Quicksort
3. Insertionsort
4. Bubblesort

```
Tamanho do Array: 100000
```

Metodo	Tempo (s)
BubbleSort	22,23699999999998000
InsertionSort	7,29699999999999700
MergeSort	0,00000000000000000
QuickSort	0,01600000000000000

Vetor original: [27630, 6054, 14073, 22271, 3228, 15272, 19471, 23206, 26153, 3
Vetor ordenado: [0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 6]

Fig. 2. Resultado dos testes para array de 100000 elementos e os primeiros elementos dos arrays. Autoria própria.

c) Array com 1000000 elementos

Nesse cenário, os algoritmos BubbleSort e InsertionSort não foram capazes de ordenar o vetor com 1000000 elementos inteiros aleatórios, levando mais de 30 minutos de espera e sem apresentar expectativa de término. Enquanto o quickSort conseguiu ordenar o vetor com o tempo de 0,084 segundos, como apresentado na Figura 3.

```
Tamanho do Array: 1000000
```

Metodo	Tempo (s)
quicksort	0,084000000000000005

Fig. 3. Resultado do quickSort para o cenário de um milhão de elementos. Autoria própria.

Outro comportamento inesperado ocorreu para o MergeSort, em que foi notado o estouro de memória devido a necessidade de alocar memória em arrays temporários para sua execução, o erro está ilustrado na Figura 4.

```
Tamanho do Array: 1000000
Makefile:29: recipe for target 'run' failed
mingw32-make: *** [run] Error -1073741571
```

Fig. 4. Erro de acesso de memória para o MergeSort. Autoria própria.

d) Array de Caracteres

Todos os algoritmos de ordenação foram capazes de ordenar um vetor de caracteres contendo 255 caracteres aleatórios dentro de um tempo satisfatório, em que a diferença de tempo de execução é mínima no qual o algoritmo do trabalho não é capaz de diferenciar, apresentando o resultado na Figura 5.

```

+-----+
Cenário: Array de 255 caracteres
+-----+
| Metodo      | Tempo (s)      |
+-----+
| BubbleSort  | 0,0000000000000000 |
| InsertionSort | 0,0000000000000000 |
| MergeSort   | 0,0000000000000000 |
| QuickSort   | 0,0000000000000000 |
Vetor char original: [, , \, !, \, , , +, U, o, K, /, c, i, ), J, Z, @, 8, w, ), d]
Vetor char ordenado: [ , , , !, !, ", ", ", #, #, #, #, $, $, $, %, %, %, &, &]
+-----+

```

Fig. 5. Resultado do teste para ordenação de vetor de caracteres. Autoria própria.

2. Implemente os dois algoritmos de busca vistos em sala (Sequencial e Binária) e em seguida compare seus tempos de busca nos seguintes cenários:
 - a) Array com 1000 elementos
 - b) Array com 100000
 - c) Array com 1000000
 - d) Buscando uma palavra em um texto Solução:

Nos cenários apresentados, as máquinas modernas conseguem executar a busca com mínimo esforço computacional, mesmo em cenários com alta quantidade de elementos como no item C, os resultados do teste são apresentados na Figura 6.

```

idry / lista > data > = quick
Algoritmos de Busca
Cenário: Array de 1000 inteiros
+-----+
| Metodo      | Tempo (s)      |
+-----+
| linear       | 0,0000000000000000 |
| binaria      | 0,0000000000000000 |
+-----+
Cenário: Array de 100000 inteiros
+-----+
| Metodo      | Tempo (s)      |
+-----+
| linear       | 0,0000000000000000 |
| binaria      | 0,0000000000000000 |
+-----+
Cenário: Array de 1000000 inteiros
+-----+
| Metodo      | Tempo (s)      |
+-----+
| linear       | 0,0000000000000000 |
| binaria      | 0,0000000000000000 |
+-----+
Cenário: Buscando a palavra 'voo' em um texto com 154 palavras
+-----+
| Metodo      | Tempo (s)      |
+-----+
| linear       | 0,0000000000000000 |
| binaria      | 0,0000000000000000 |
+-----+

```

Fig. 6. Resultado do teste para buscas em vetores de tamanhos diferentes sem repetições de teste. Autoria própria.

Para que seja possível diferenciar o tempo de execução de cada algoritmo de busca, os testes de cada cenário serão repetidos 50.000 vezes, considerando o tempo final de execução como o tempo para realizar 50.000 vezes a busca por um mesmo elemento.

O resultado para todos os cenários de teste é idêntico, em que o algoritmo de busca binária se sobressai ao algoritmo de busca linear. As análises dos resultados ressaltam que o tempo de busca do algoritmo linear é proporcional a quantidade de elementos do vetor a ser buscado, enquanto o algoritmo de busca binário se mantém com tempo mais próximos mesmo em vetores de tamanhos distintos, conforme Figura 7.

Algoritmos de Busca	
Cenário: Array de 1000 inteiros	
Metodo	Tempo (s)
linear	0,049000000000000002
binaria	0,001000000000000000
Cenário: Array de 100000 inteiros	
Metodo	Tempo (s)
linear	0,480999999999999980
binaria	0,000000000000000000
Cenário: Array de 1000000 inteiros	
Metodo	Tempo (s)
linear	0,687999999999999940
binaria	0,002000000000000000
Cenário: Buscando a palavra 'dia' em um texto com 154 palavras	
Metodo	Tempo (s)
linear	0,003000000000000000
binaria	0,001000000000000000

Fig. 7. Resultado do teste para buscas em vetores de tamanhos diferentes com repetições de teste. Autoria própria.