



NOME:

DATA: / /

TRABALHO 02 – Ordenação e Busca

INTRODUÇÃO

Ordenação é o ato de se colocar os elementos de uma sequência de informações, ou dados, em uma relação de ordem predefinida. O termo técnico em inglês para ordenação é *sorting*, cuja tradução literal é "classificação".

Dado uma sequência de n dados:

$\langle a_1, a_2, \dots, a_n \rangle$

O problema de ordenação é uma permutação dessa sequência:

$\langle a'_1, a'_2, \dots, a'_n \rangle$

tal que:

$a'_1 \leq a'_2 \leq \dots \leq a'_n$

para alguma relação de ordem.

Algumas ordens são facilmente definidas. Por exemplo, a ordem numérica, ou a ordem alfabética—crescentes ou decrescentes. Contudo, existem ordens, especialmente de dados compostos, que podem ser não triviais de se estabelecer.

Um algoritmo que ordena um conjunto, geralmente representada num vetor, é chamado de algoritmo de ordenação. **Algoritmo de ordenação** em ciência da computação é um algoritmo que coloca os elementos de uma dada sequência em uma certa ordem—em outras palavras, efetua sua ordenação completa ou parcial. As ordens mais usadas são a numérica e a lexicográfica. Existem várias razões para se ordenar uma sequência. Uma delas é a possibilidade de se acessar seus dados de modo mais eficiente.

Entre os mais importantes, podemos citar *bubble sort* (ou ordenação por flutuação), *insertion sort* (ou ordenação por inserção), *merge sort* (ou ordenação por mistura) e o *quicksort*. Existem diversos outros, que o aluno pode com dedicação pesquisar por si. Para estudo no entanto nos concentraremos nos principais.

Além disso, a ordenação é importante para a busca, por exemplo, no caso da busca binária, onde a lista deve estar ordenada para que a busca seja realizada com sucesso.

OBJETIVO

- 1) **(2 pontos)** Fazer uma pesquisa aprofundada com. Definição, algoritmo, vantagens e desvantagens, referências bibliográficas dos seguintes algoritmos de ordenação:
 - Selection Sort;
 - Insertion Sort;
 - Bubble Sort;
 - Radix Sort;
 - Quick Sort;
 - Merge Sort;
 - Heap Sort
 - Busca sequencial
 - Busca Binária
- 2) **(3 pontos)** Cada dupla irá fazer uma apresentação de um dos algoritmos supracitados. O algoritmo apresentado para cada dupla será sorteado no momento das apresentações. Destaca-se que a nota é individual, neste quesito;

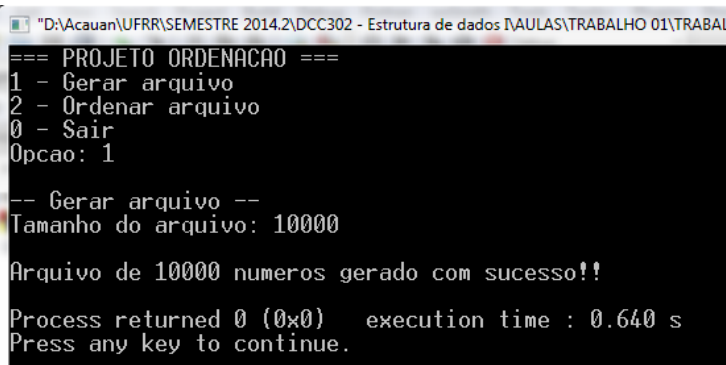
- a. ('Busca Sequencial', 'Insertion Sort') -> Tiago, Barbara e Diego Sousa **(04/11/2019)**
 - b. ('Selection', 'Bubble') → Ramses e Matheus **(04/11/2019)**
 - c. Busca Binaria → Joao Paulo e Karen **(06/11/2019)**
 - d. HeapSort → Fabio e George **(06/11/2019)**
 - e. MergeSort → Luiz e Jeovane **(06/11/2019)**
 - f. QuickSort → Willian e Jorge **(08/11/2019)** → **aula de reposição**
 - g. Radix Sort → Cristiano e Joao Roberto **(08/11/2019)** → **aula de reposição**
- 3) **(5 pontos)** Implemente um sistema em C que gere 4 sequencias numéricas aleatórias (não ordenadas), uma de 100 números, outra com 1000, outra de 10.000 números e outra de 100.000 números. O programa deve gerar esses números e armazena-los em um arquivo .txt (cada sequencia em um arquivo diferente). Este arquivo gerado deve ser lido pelo próprio programa e ordenado pelos algoritmos antes citados.
- Implemente os algoritmos de forma a receber este arquivo contendo os números, ordenando-os e salvando-os novamente em 4 arquivos diferentes (ordenados) .txt. Implemente uma linha em cada que mostre a velocidade de processamento de cada algoritmo ao final da ordenação. A velocidade na verdade será mensurada de três formas (*observar as tabelas no final deste arquivo*):
1. Contagem do número de trocas na ordenação;\
 2. Contagem do número de comparações na ordenação;
 3. Contagem em milissegundos e em segundos do tempo de execução de cada algoritmo.
- O objetivo é comparar o desempenho dos algoritmos de ordenação. Por fim, crie um módulo no qual será possível **buscar** por um valor na lista ordenada, neste caso usando **busca binária** e na lista aleatória usando **busca sequencial**.
- Obs.: não será necessário a contagem do número de comparações/trocas do Radix Sort.**

METODOLOGIA

- O trabalho deve ser feito em dupla;
- Procure primeiro entender o problema, tente quebrá-lo em partes menores;
- Faça testes da geração dos números aleatórios. São 3 arquivos que deverão ser lidos pelo programa um com cem números, outro com 10mil números e outro com 100mil números..
- Depois de conseguir gerar os arquivos, implemente os algoritmos de ordenação e faça com que os mesmos recebam o conteúdo dos arquivos, ordene-os e guarde-os novamente em arquivos ordenados.
- Quando necessário utilize as TADs de estrutura de dados para armazenar os números no processo de ordenação, ou leitura pelo algoritmo.

FUNCIONAMENTO

Inicialmente deverá ser elaborado um MENU, semelhante ao da figura (você pode implementar sua forma) para gerar e ler os arquivos. **Crie uma função de busca depois que o arquivo estiver ordenado.**



```
"D:\Acauan\UFRR\SEMESTRE 2014.2\DCC302 - Estrutura de dados I\AULAS\TRABALHO 01\TRABALHO 01"
=== PROJETO ORDENACAO ===
1 - Gerar arquivo
2 - Ordenar arquivo
0 - Sair
Opcao: 1

-- Gerar arquivo --
Tamanho do arquivo: 10000

Arquivo de 10000 numeros gerado com sucesso!!

Process returned 0 (0x0)   execution time : 0.640 s
Press any key to continue.
```

A cada arquivo lido mostrar as descrições do arquivo como Tamanho do Arquivo, tempo de execução do algoritmo e se o arquivo foi gerado com sucesso.

```
"D:\Acauan\UFRR\SEMESTRE 2014.2\DCC302 - Estrutura de dados \AULAS\TRABALHO

=== PROJETO ORDENACAO ===
1 - Gerar arquivo
2 - Ordenar arquivo
0 - Sair
Opcao: 2

-- Ordenando arquivo --
Ler arquivo:

ALGORITMO: Selection Sort
Tamanho: 10000
Tempo: x.xxx s
Arquivo Ordenado: ok

ALGORITMO: Bubble Sort
Tamanho: 10000
Tempo: x.xxx s
Arquivo Ordenado: ok

ALGORITMO: Quick Sort
Tamanho: 10000
Tempo: x.xxx s
Arquivo Ordenado: ok
```

Estas imagens são meramente para referencia, vocês podem implementar do seu jeito, tanto que faça o solicitado.

ENTREGA

- Deverão ser entregues:
 - Pesquisa sobre os algoritmos de ordenação
 - Programa que gera arquivos e ordena utilizando algoritmos conhecidos e faz comparação de tempo entre os algoritmos.
- Valorizam-se trabalhos que implementarem comandos e/ou recursos adicionais;
- **Data de Apresentações: 20/11/2019**
- **Data de Entrega do programa e da parte escrita: 20/11/2019**
- **Data de Defesa do trabalho: 22/11/2019 (aula reposição) e 25/11/2019.** Obs.: as defesas acontecerão durante o horário da aula, entretanto se não der tempo de todos apresentarem de 08h às 10h, então o restante apresentará no bloquinho em horário a combinar. Na defesa o professor irá perguntar sobre o funcionamento dos algoritmos de ordenação e busca para a dupla. Observe que você irá defender o seu trabalho escrito que é sobre todos os algoritmos de ordenação e busca mencionados no item 1) da seção objetivo.

Quantidade de Comparações				
	100	1.000	10.000	100.000
BubbleSort	4.950	499.500	49.995.000	4.999.950.000
QuickSort	997	12.852	181.203	2.114.943
MergeSort	558	8.744	123.685	1.566.749

Quantidade de Movimentos				
	100	1.000	10.000	100.000
BubbleSort	2.628	242.827	25.160.491	2.499.136.980
QuickSort	570	8.136	103.575	1.310.586
MergeSort	1376	19968	272640	3385984

Tempo de execução (s)				
	100	1.000	10.000	100.000
BubbleSort	0,00007	0,0081	0,8587	114,8400
QuickSort	0,00003	0,0004	0,0049	0,0844
MergeSort	0,00015	0,0016	0,0194	0,2316

Imagem de Exemplo mostrando as informações que devem ser obtidas ao rodar os algoritmos de ordenação

Créditos: adaptação do trabalho do professor Acauan Ribeiro na disciplina de Estrutura de Dados I.