SSC0503 - Introdução à Ciência de Computação II
1ª Trabalho
Professor: Claudio Fabiano Motta Toledo (`claudio@icmc.usp.br`)
Estagiário PAE: Jesimar da Silva Arantes (`jesimar.arantes@usp.br`)

---

1. Informações Gerais

- Equipe de duas a três pessoas;
- Entrega dia 23/09/2016 até as 23h59min via STOA;
- Em caso de dúvida do enunciado, procurar o professor ou o estagiário PAE.

2. Objetivo

Este trabalho tem por objetivo permitir ao aluno simular e testar a execução dos algoritmos de BubbleSort, InsertionSort e MergeSort com vetores de tamanho variado e verificar qual dos três algoritmos possui a maior complexidade.

3. Principais Tarefas

- Fazer simulações com o algoritmo de BubbleSort;
- Fazer simulações com o algoritmo de InsertionSort;
- Fazer simulações com o algoritmo de MergeSort;
- Fazer simulações com vetores de diferentes tamanhos de entrada;
- Fazer a análise da complexidade durante a execução;
- Fazer um relatório explicando a ideia geral do algoritmo, as simulações efetuadas, apresentando os gráficos de comparação do desempenho dos três algoritmos e se eles se classificam como logarítmicos, polinomiais ou exponenciais.
- O relatório deve conter entre duas a três páginas.

4. Enunciado

Desenvolver um programa contendo os algoritmos de ordenação BubbleSort, InsertionSort e MergeSort atendendo aos requisitos listados abaixo.

Os seguintes Requisitos Funcionais devem ser satisfeitos:

- Programe os algoritmos de ordenação BubbleSort, InsertionSort e MergeSort.
- O Vetor a ser ordenado deve possuir 100.000 posições.
- O programa terá as seguintes opções no menu: "1-Testar", "2-Simular" e "3-Sair".
- Quando o usuário escolher a opção "1-Testar", o programa deverá exibir um sub-menu com as opções "1-BubbleSort", "2-InsertionSort", "3-MergeSort", "4-Retornar".

- Se o usuário escolher a opção retornar, o programa deverá retornar ao menu principal.
- Quando o usuário escolher um dos algoritmos dentro da opção testar, o programa deverá carregar um arquivo, cujo nome é fornecido via teclado. O arquivo poderá conter qualquer tamanho até 100.000 posições. Na sequência, o programa deverá executar o algoritmo escolhido.
- Para os algoritmos executados na opção testar, a função de ordenação deverá retornar o vetor ordenado, que será argumento para outra função que testará se o vetor realmente está ordenado. Após executar a função testar o programa deverá escrever, num arquivo de saída, o vetor ordenado no caso da ordenação ter obtido sucesso, ou escrever "INSUCESSO" dentro do arquivo no caso do teste verificar que existem elementos fora de lugar. A primeira linha do arquivo deverá conter a sentença "Essa ordenação necessitou de X comparações." seguida da quantidade de interações necessárias para a ordenação. Vide detalhamento para o arquivo SaidaXXX.txt mais adiante.
- A saída da opção testar, descrita acima, deverá ser salvo em um arquivo de saída chamado "SaidaXXXX.txt", onde XXXX refere-se ao nome do método de ordenação. Portanto, XXXX será substituído por BubbleSort, InsertionSort, MergeSort. Este arquivo de saída é gerado apenas na opção "1 - Testar".
- A opção "Simular" carregará um vetor a partir da leitura de um arquivo que estará no mesmo diretório do programa executável. Este arquivo terá 100.000 valores. Nenhuma operação deverá ser executada em cima do arquivo, ele apenas deve ser utilizado para carregar o vetor. Para maiores detalhes do formato do arquivo, veja exemplo do arquivo "Entrada.txt" fornecido mais adiante neste documento.
- O nome do arquivo de entrada deverá ser lido via teclado e ele estará no mesmo diretório do programa executável.
- Após carregar o vetor a partir do arquivo, na sequência, e automaticamente, o programa deverá executar todos os algoritmos de ordenação, registrando informações sobre seu desempenho e o número de instruções executadas.
- Utilize um contador para contar o número de vezes que as instruções são executadas nos principais laços de repetição dos algoritmos.
- Executar cada algoritmo considerando a variação do tamanho do vetor de 10.000 posições. Isto é, deverá ser feita a simulação de cada um dos algoritmos para os vetores de tamanho 0, 10.000, 20.000 até 100.000 (de 10.000 em 10.000). Para facilitar o trabalho, é permitido carregar o vetor de 100.000 de posições e utilizar uma variável para limitar virtualmente o tamanho do vetor e então simular a existência de vários vetores de tamanho 0, 10.000, 20.000 até 100.000 (de 10.000 em 10.000), a partir de um único vetor.
- A opção Simular deverá executar todos os algoritmos de ordenação no mesmo vetor carregado. Para isto, serão necessários 2 vetores, um contendo o vetor original e o qual não será alterado. O segundo vetor conterá uma cópia do vetor original e será

utilizado para ser ordenado, a fim de se obter o desempenho do número de comparações necessárias para sua ordenação. A cada execução de um algoritmo, o vetor original deverá ser carregado para sua cópia.

  - Utilize um contador para contar o número de comparações necessárias para fazer a ordenação, utilizando os principais laços de repetição para isto;
  - Executar cada algoritmo 1 vez para cada tamanho de vetor. Isto é, vetor de tamanho zero será ordenado 1 vez para cada um dos algoritmos de ordenação. O vetor de tamanho 10.000 será ordenado 1 vez para cada um dos algoritmos de ordenação e assim sucessivamente, com o tamanho variando de 10.000 em 10.000 até o tamanho máximo do vetor de 100.000.
  - A quantidade de comparações executadas por cada algoritmo em cada tamanho solicitado deverá será gravado num arquivo de SAIDA detalhado mais abaixo.
  - O arquivo utilizado para carregar o vetor para executar a opção Simular conterá exatamente 100.000 de posições.

- Nenhuma operação deverá ser executada em cima do arquivo, ele apenas deve ser utilizado para carregar o vetor. O algoritmo de ordenação será executado sobre o vetor.
- As funções/procedimentos de ordenação devem retornar o vetor devidamente ordenado.
- Como saída o programa deverá informar quantas comparações foram necessárias para ordenar o vetor, considerando-se cada um dos tamanhos do vetor e cada um dos algoritmos executados. Vide arquivo de saída abaixo para maiores detalhes.
- Gerar um relatório contendo as seguintes informações.
  - Apresente uma tabela com o nome de todos os algoritmos que você programou e sua complexidade oficial, por exemplo, n, $log\ n$, n^2, n^n, k^n, n^k, etc.
  - Um gráfico para cada um dos algoritmos simulados contendo duas curvas cada gráfico. Uma curva refere-se ao desempenho obtido na simulação e gravado no arquivo "SAIDA.TXT". A outra curva refere-se à complexidade oficial do algoritmo. Comente se sua curva é similar a complexidade oficial do algoritmo. Caso haja discrepâncias entre as curvas, justifique a razão da diferença entre as curvas da simulação e a oficial.
  - Informe se o algoritmo apresenta comportamento Exponencial, Polinomial, Logarítmico de acordo com o formato das curvas extraídas do experimento realizado.

Os seguintes Requisitos Não Funcionais devem ser satisfeitos:

- Programar os algoritmos em C/C++.
- Entregar o código fonte do programa.
- Entregar o relatório no formato PDF, observação **não** serão aceitos relatórios no formato DOC, DOCX, ODT.

- Identifique os membros da equipe com nome completo no cabeçalho do código fonte do programa para evitar enganos.
- Os arquivos criados deverão ser compactados e entregues via STOA.
- O nome do arquivo compactado deverá ser "Trabalho1_NomeDoAluno1_E_NomedoAluno2.zip", onde NomeDoAluno1 e NomeDoAluno2 são os nomes e pelo menos um sobrenome, dos alunos que participam do grupo.
- Caso haja necessidade de explicar alguma coisa do programa como dicas de compilação, dentre outros, crie um arquivo "Leia-me.txt" e entregue junto com o programa.

5. Descrição do Arquivo de Entrada

Exemplo do arquivo "Entrada.txt"

```
23
51004
23098
365874
398452
10000
567
6865
12760
9234
98076
2587
```

- O arquivo pode estar vazio ou conter qualquer quantidade de números até o máximo de 100.000 valores para a opção "Testar";
- O arquivo conterá exatamente 100.000 de valores para a opção "Simular";
- Cada valor estará em uma linha;
- Cada valor está no intervalo 0 e 1.000.000 inclusive.

6. Descrição do Arquivo de "SaidaXXXX.txt" para a opção "Testar".

- Os valores gravados no arquivo referem-se à saída (resposta) do programa de ordenação, isto é, eles devem estar ordenados dentro do arquivo;
- A primeira linha deverá conter: "Essa ordenação necessitou de X comparações.", onde X é o número de comparações contadas durante a execução;
- Cada valor deve estar em uma linha;
- XXXX refere-se ao nome do algoritmo como explicado nos requisitos.

Exemplo do arquivo "SaidaBubbleSort.txt"

```
Essa ordenação necessitou de 76 comparações.
23
54
234
567
1276
9234
98076
256987
365874
398452
689365
```

7. Descrição do Arquivo "SAIDA.TXT" para a opção "Simular"

- Cada simulação de um algoritmo com um determinado tamanho de vetor estará em uma linha do arquivo;
- Cada linha possuirá 4 valores, separados por vírgula. O primeiro representa o tamanho do vetor e do segundo até o quarto representam a média do desempenho dos algoritmos de ordenação na seguinte ordem: BubbleSort, InsertionSort e MergeSort.

Exemplo do arquivo "SAIDA.TXT"

```
0, 1, 1, 1
10000, 28000, 27000, 25
20000, 67000, 65000, 298
...
90000, 420000, 410000, 765
100000, 535000, 520000,876
```

8. Dicas de Passos para o Desenvolvimento

- Primeiramente desenvolva os algoritmos de ordenação para compreender a lógica do que você vai resolver, e assim evitar passar tempo em demasia tentando compreender onde errou;
- Programe o algoritmo em C/C++ pensando em um vetor de 20 posições para verificar se o algoritmo funciona;
- Após estar funcionando com um vetor de 20 posições programe a leitura de arquivo para carregar o vetor a partir de um arquivo.

9. Dicas para antes da entrega

- Antes de entregar, certifique-se de que o arquivo compactado contém os arquivos do programa e o relatório;
- Passe por cada requisito funcional e não funcional, certificando-se de que eles foram cumpridos;