Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – Turma 05A
Prof. Atílio Gomes

Projeto 02

A solução do problema descrito neste documento deve ser entregue até as 23h59 do dia **07/04/2021** via Moodle.

Leia atentamente as instruções abaixo.

Instruções:

- Este trabalho **DEVE** ser feito em **DUPLA** ou **INDIVIDUALMENTE** e implementado usando a linguagem de programação C++
- O seu trabalho deve ser compactado (**.gz, .tar, .zip, .rar**) e enviado pelo Moodle.
- Identifique o seu código-fonte colocando os **nomes** e **matrículas** dos integrantes da equipe como comentário no início do código.
- Indente corretamente o seu código para facilitar o entendimento.
- O código-fonte deve estar devidamente **organizado** e **documentado**.
- Esta avaliação vale de 0 a 10 pontos.
- **Observação:** Se você alocar memória dinamicamente, lembre-se de desalocar os endereços de memória alocados quando os mesmos não forem mais ser usados.
- **Observação: Qualquer indício de plágio resultará em nota ZERO para todos os envolvidos.**

DICA: COMECE O TRABALHO O QUANTO ANTES.

1 Problema: Comparando empiricamente o tempo de execução de algoritmos de ordenação

Neste trabalho, deve-se implementar os seguintes algoritmos de ordenação:

- InsertionSort, SelectionSort, MergeSort e QuickSort

Você deve programar duas versões de cada algoritmo acima:

- Uma versão para cada um desses quatro algoritmos usando **vetor**.
- Uma versão para cada um desses quatro algoritmos usando **lista duplamente encadeada**.

Além disso, pesquise um algoritmo que não foi dado em sala e implemente esse algoritmo também (para este algoritmo, tanto faz usar vetor ou lista duplamente encadeada. Fica a seu critério.)

Obs.: Ou seja, devem ser implementados e executados o total de 9 algoritmos.

Você e seu par devem inicialmente pensar em como vão dividir o trabalho entre a dupla, para que uma pessoa não fique com os algoritmos mais simples e outra fique sobrecarregada com algoritmos mais complexos.

Como será preciso dividir os algoritmos entre a dupla, é prudente pensar em como vocês vão dividir os arquivos de implementação do trabalho. Uma possibilidade é proposta abaixo, mas você não precisa se prender a ela, pode organizar os arquivos como for melhor e mais eficiente para você:

- `ordenacaoVetor.h`: contém os protótipos das funções de ordenação usando vetor.
- `ordenacaoVetor.cpp`: contém as implementações das funções de ordenação usando vetor.
- `ordenacaoLista.h`: contém os protótipos das funções de ordenação usando lista duplamente encadeada.
- `ordenacaoLista.cpp`: contém as implementações das funções de ordenação usando lista duplamente encadeada.
- `main.cpp`: onde todas a funções devem ser testadas.

2 Testes

Você deve comparar diferentes estratégias de ordenação para ordenar um conjunto de N inteiros positivos, **aleatoriamente gerados**. Realize experimentos considerando vetores aleatoriamente gerados com tamanho $N = 500$, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000 e 20000, no mínimo. Para cada valor de N, realize experimentos com 5 **sementes** diferentes. Para a comparação dos algoritmos de ordenação, avalie:

- os **valores médios do tempo de execução**[1].

No relatório, você deve apresentar uma pequena comparação entre os algoritmos/implementações. Apresente gráficos e/ou tabelas com os resultados obtidos. Discuta os resultados e conclusões obtidas. No seu experimento, qual algoritmo teve melhor desempenho? Você pode dizer por quê?

Observação: Juntamente com esta descrição do trabalho, foi disponibilizado no Moodle um pequeno exemplo[2], com a implementação usual do algoritmo BubbleSort e de um outro algoritmo que ordena vetores de inteiros chamado `CocktailSort`. No programa-exemplo, 21 vetores de inteiros de tamanhos variados e gerados aleatoriamente são ordenados usando os algoritmos BubbleSort e CocktailSort. No programa fornecido são calculados os valores médios dos tempos de execução. Para cada valor de N estipulado acima, são gerados 5 vetores aleatórios de tamanho N e a média do tempo das cinco execuções do algoritmo sobre esses cinco vetores de tamanho N é armazenada em um arquivo.

Para cada execução do CocktailSort e do BubbleSort, é calculada a média do seu tempo de execução em microssegundos e esses dados são gravados em arquivos chamados `resultadoCocktail.txt` e `resultadoBubble.txt` (que encontram-se na pasta `resultados`). Cada um desses arquivos é composto de duas colunas: a primeira indica o tamanho do vetor e a segunda indica o tempo médio em microssegundos que o respectivo algoritmo levou para ordenar o respectivo vetor.

2.1 Gerando os gráficos

Uma vez gerado o arquivo `resultadoCocktail.txt`, por exemplo, você pode usar este arquivo para gerar o gráfico usando o gerador de gráficos (plotador) que você quiser. Se você nunca fez isso na vida e não conhece nenhum gerador de gráficos, há alguns escritos em Python e que são razoavelmente fáceis de usar. Procure por **Matplotlib** na internet.

[1] Existem diversas formas de medir o tempo de execução de uma função em C++. Pesquise, por exemplo, a biblioteca `std::chrono`.

[2] O programa-exemplo está todo em um arquivo único. Você pode partir dele para fazer o seu trabalho. Como o programa-exemplo está todo em um arquivo só, não é desse jeito que o seu trabalho deve ser organizado. Ao final, você terá muitas funções (9 algoritmos) e deve pensar como vai gerenciar todas elas e como vai organizar os dados e resultados.

Pois bem, com o arquivo `resultadoCocktail.txt` em mãos, é possível usar um plotador para plotar(desenhar) um gráfico que mostre a relação entre o tamanho do vetor gerado e o tempo em microssegundos que levou para o CocktailSort ordenar o vetor. Para quem usa GNU/Linux, existe uma ferramenta de linha de comando chamada `gnuplot`, que eu usei para gerar os seguintes gráficos[3] das Figuras 1 e 2.

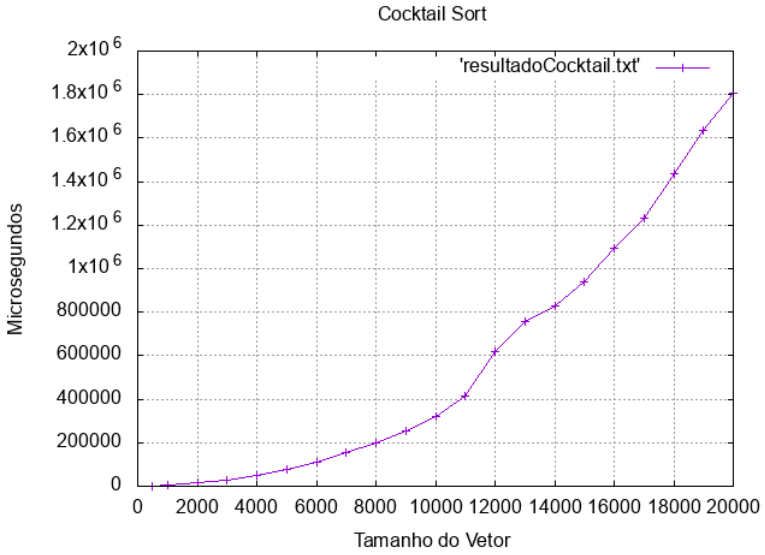

Figura 1: CocktailSort

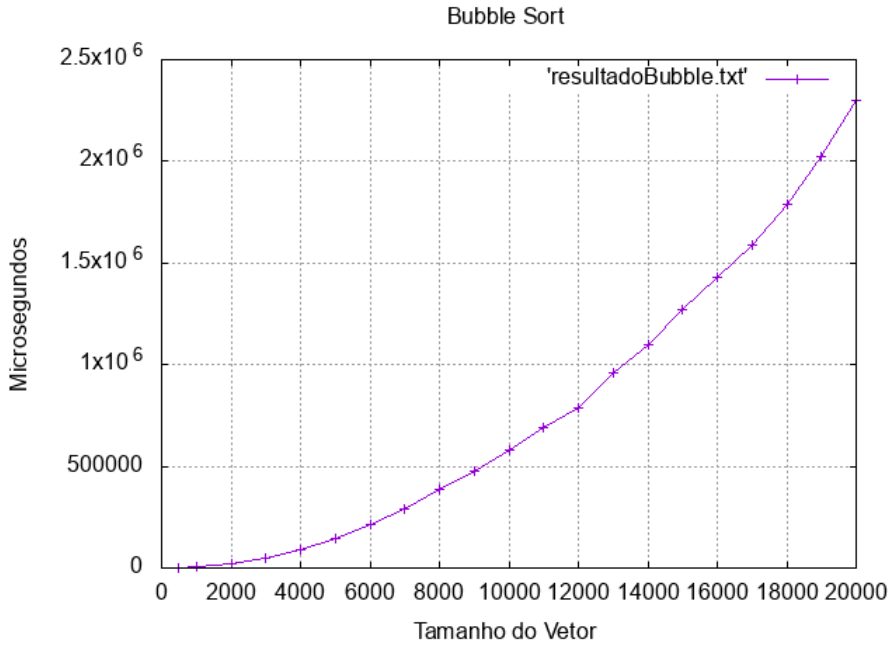

Figura 2: BubbleSort

[3]Os arquivos usados para gerar os gráficos no gnuplot estão todos na pasta `resultados`. Os arquivos que o programa gnuplot lê são os arquivos com extensão .p que estão na pasta, ou seja, os arquivos graphBubble.p, graphCocktail.p e graphBubbleCocktail.p

No gráfico da Figura 3, os tempos de execução do BubbleSorte e do CockTail Sort são plotados no mesmo gráfico a fim de compararmos visualmente as suas performances.

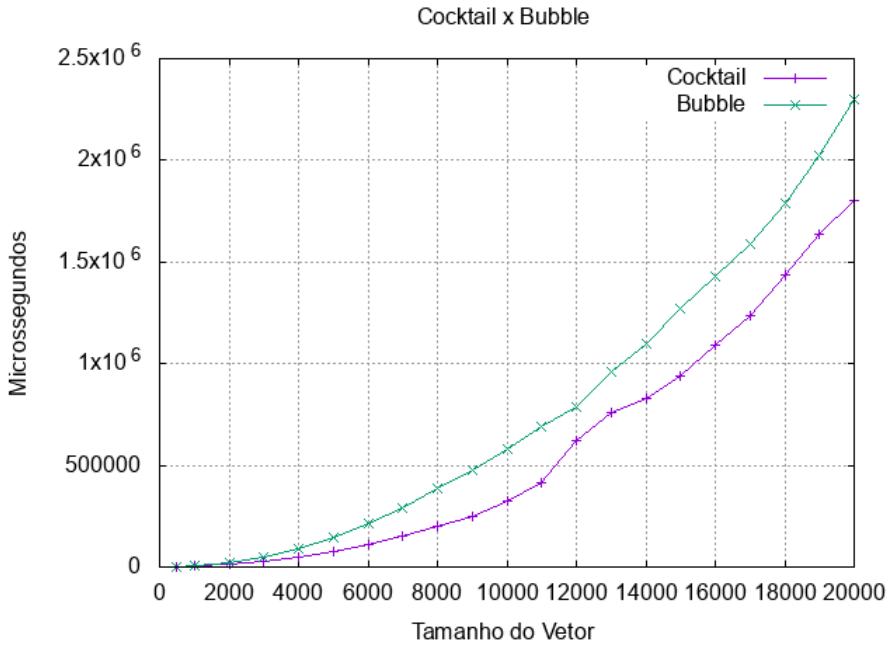

Figura 3: BubbleSort × CockTailSort. O CockTailSort acaba sendo mais rápido que o BubbleSort.

Para quem quiser aprender a usar o **gnuplot** para plotar os gráficos a partir de um arquivo texto, esses links podem ajudar:

- `http://www.matsuura.com.br/2016/08/como-instalar-o-gnuplot-504.html`
- `https://howtoinstall.co/pt/gnuplot`
- `http://www.gnuplotting.org/plotting-data`
- `https://www.asc.ohio-state.edu/physics/ntg/780/handouts/gnuplot_quadeq_example.pdf`

3 Informações adicionais

- Deverá ser submetido, juntamente com o código, um **relatório técnico** explicando tudo o que foi feito no trabalho. Dentre outras coisas, o relatório deve ter:

  (1) Os nove algoritmos de ordenação que foram programados;

  (2) Gráficos mostrando os tempos de execução dos algoritmos para distintos tamanhos de entrada; e sua interpretação dos dados que são mostrados;

  (3) Uma explicação do algoritmo que você pesquisou e implementou.

  (4) Comparação entre os algoritmos usando vetor;

  (5) Comparação entre os algoritmos usando listas;

  (6) Uma seção descrevendo como o trabalho foi dividido entre as duplas;

  (7) Uma seção de dificuldades encontradas.

  (8) Uma seção de bibliografia contendo as referências utilizadas. **Se você consultar algum site da internet ou livro, coloque esta fonte de pesquisa no seu trabalho. Não omita.**

- O relatório deve ser entregue em formato PDF.

- O trabalho deverá ser entregue até o dia **07 de abril de 2021, 23h59**.