Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – Turma 02A
Prof. Atílio Gomes

Terceiro Trabalho

A solução do problema descrito neste documento deve ser entregue até a meia-noite do dia **25/11/2019** via SIPPA.

Leia atentamente as instruções abaixo.

Instruções:

- Este trabalho **DEVE** ser feito em **DUPLA** e implementado usando a linguagem de programação C++
- O seu trabalho deve ser compactado (**.gz, .tar, .zip, .rar**) e enviado para o SIPPA na atividade correspondente ao **Trabalho 5** da disciplina.
- 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**.
- Este trabalho 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

Neste trabalho, deve-se implementar os seguintes algoritmos de ordenação: **BubbleSort, InsertionSort, SelectionSort, MergeSort, HeapSort e QuickSort**.

Deve-se ter:

- Uma versão iterativa e uma versão recursiva para cada um desses seis algoritmos usando **vetor**;
- Uma versão (iterativa ou recursiva) para cada um desses seis algoritmos usando **lista duplamente encadeada**.

O código do seu trabalho deve estar organizado em arquivos separados. Uma possibilidade é proposta abaixo:

- `ordenacaoVetor.h`: contém os protótipos das funções de ordenação usando vetor.
- `ordenacaoLista.h`: contém os protótipos das funções de ordenação usando lista.
- `ordenacaoVetor.cpp`: contém as implementações das funções de ordenação usando vetor.
- `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/listas aleatoriamente gerados com tamanho $N = 1000$, 5000, 10000, 50000, 100000, 500000 e 1000000, no mínimo. Para cada valor de N, realize experimentos com 5 **sementes** diferentes. Para a comparação dos algoritmos de ordenação, avalie:

(a) os **valores médios do tempo de execução**[1];

(b) o **número de comparações de chaves**;

(c) e o **número de cópias de valores**.

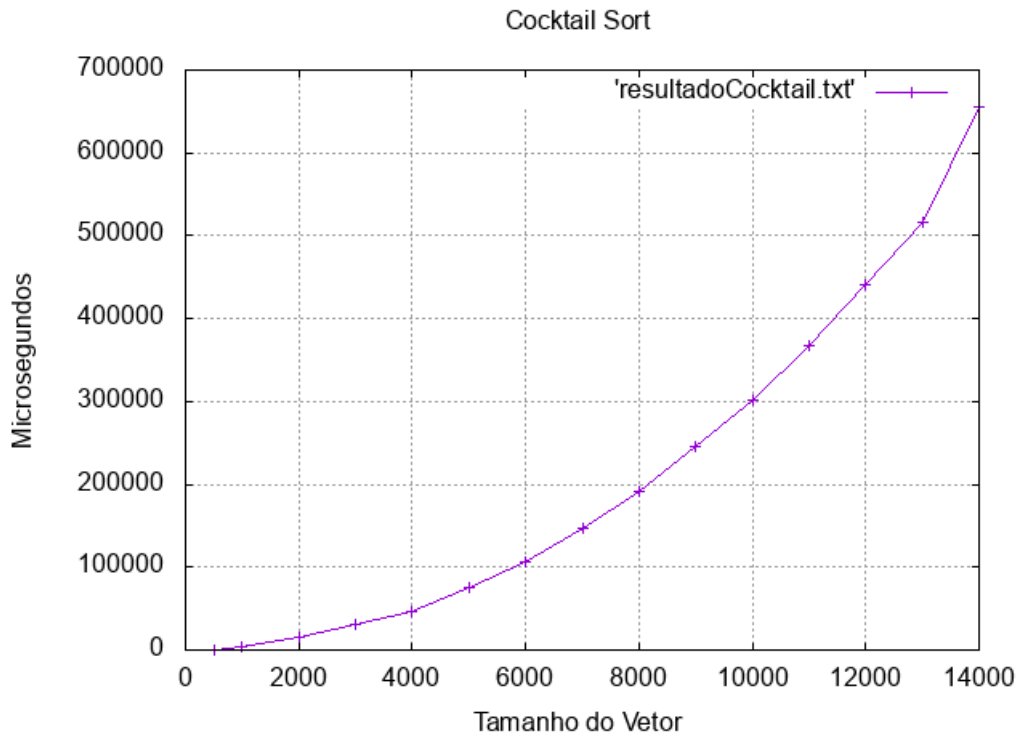
No relatório, você deve apresentar uma pequena comparação entre os algoritmos/implementações. Apresente gráficos e/ou tabelas com os resultados obtidos. Discuta os resultados e conclusões obtidas, considerando as diferentes métricas. Qual algoritmo tem melhor desempenho. Por quê?

[1]Existem diversas formas de medir o tempo de execução de uma função em C++. Pesquise, por exemplo, a biblioteca `std::chrono`. Em C++, também é possível medir o tempo de execução de uma função no estilo do C, usando a função `clock()` da biblioteca `ctime`.

Observação: Juntamente com esta descrição do trabalho, foi disponibilizado no SIPPA um pequeno exemplo[2], com a implementação do algoritmo BubbleSort e de um outro algoritmo que ordena vetores de inteiros chamado `CocktailSort`. No programa-exemplo, 15 vetores de inteiros de tamanhos variados e gerados aleatoriamente são ordenados usando os algoritmos BubbleSort e CocktailSort. No programa fornecido são calculados apenas os valores médios dos tempos de execução. As demais métricas não foram programadas.

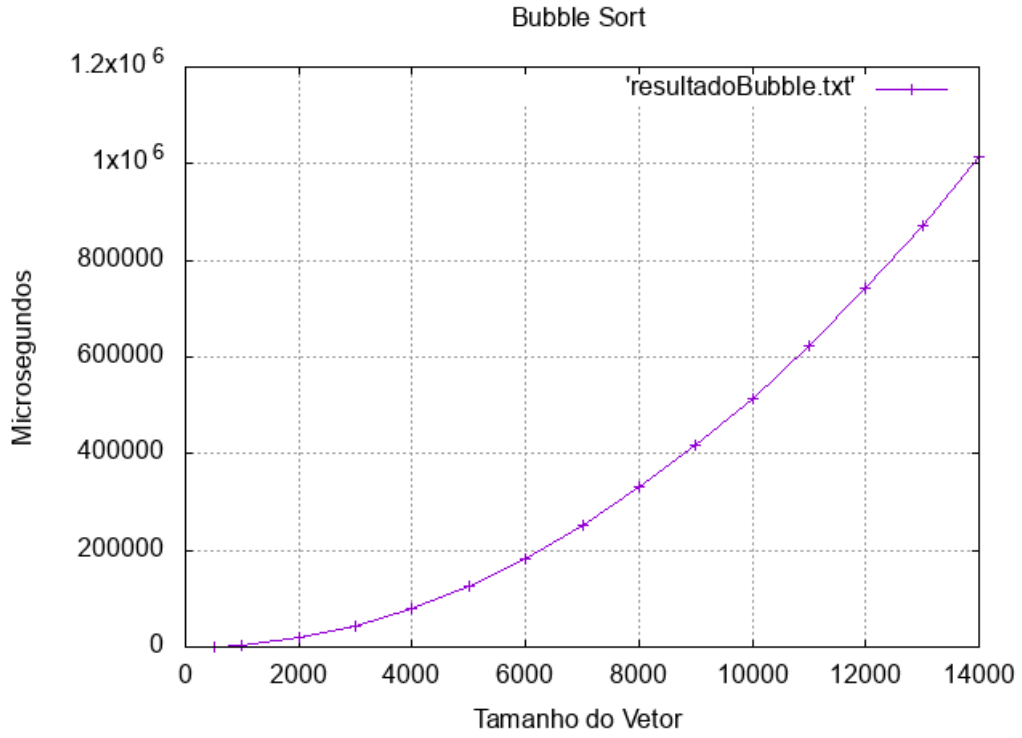
Para cada execução do CocktailSort e do BubbleSort, é calculada a média do seu tempo de execução em microssegundos e esses dados são gravados em arquivos chamados `resultadoCocktail.txt` e `resultadoBubble.txt` (que encontram-se na pasta `resultados`). Cada um desses arquivos é composto de duas colunas: 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.

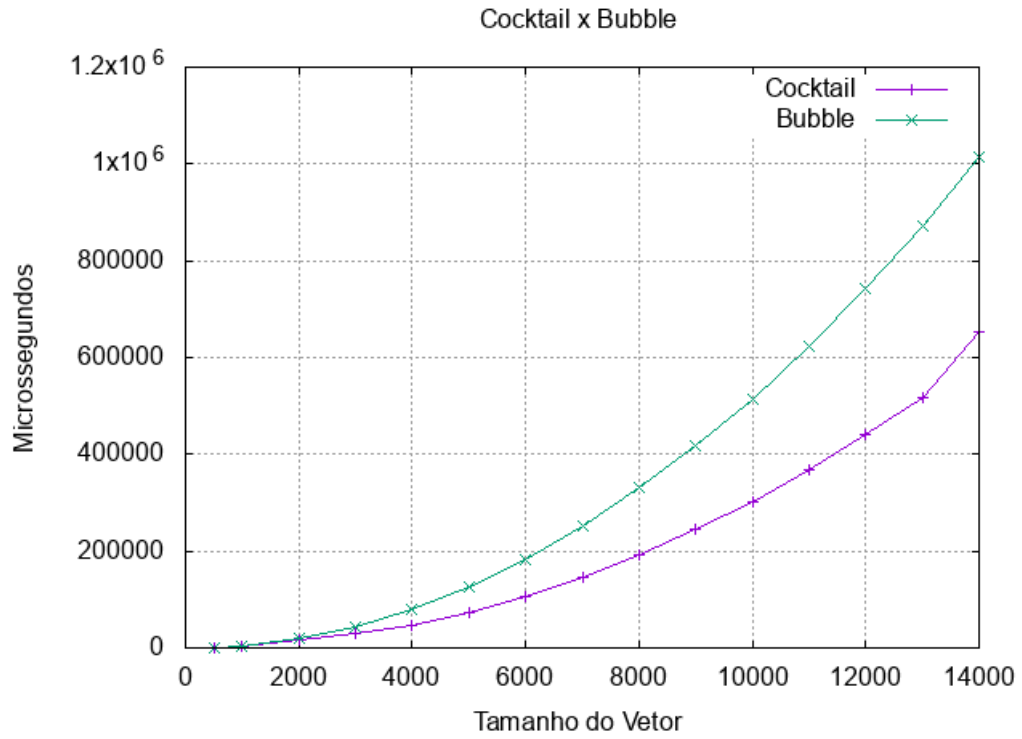
Por exemplo, com o arquivo `resultadoCocktail.txt` em mãos, é possível usar uma ferramenta de geração de gráficos para plotar(desenhar) um gráfico que mostre a relação entre o tamanho do vetor gerado e o tempo em microssegundos que levou para o CocktailSort ordenar o vetor. Para quem usa GNU/Linux, existe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



[2]O programa-exemplo é mais para ilustração. Não se prenda a ele para fazer o seu trabalho. Até porquê está tudo em um arquivo só. Não é desse jeito que o seu trabalho deve ser organizado. Ao final, você terá muitas funções e deve pensar como vai gerenciar todas elas e como vai organizar os dados e resultados.

[3]Os arquivos usados para gerar os gráficos no gnuplot estão todos na pasta `resultados`.





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:
  - Uma descrição de cada um dos seis algoritmos de ordenação;

  - Gráficos ou tabelas mostrando comparações entre os algoritmos para distintos tamanhos de entrada;
  - Comparação entre os algoritmos iterativos e recursivos;
  - Comparação entre as implementações que usam vetor e que usam listas;
  - Uma seção descrevendo como o trabalho foi dividido entre as duplas;
  - Uma seção descrevendo otimizações que foram feitas no código;
  - Uma seção de dificuldades encontradas.

- O trabalho deverá ser feito em dupla (**as notas serão individuais**);
- O trabalho deverá ser entregue até o dia **25 de Novembro**;
- O número máximo de páginas do relatório são 8 páginas.
- A apresentação do trabalho será feita em horário definido pelo professor.