Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados
Prof. Atílio G. Luiz

Projeto Final

A solução do problema descrito neste documento deve ser entregue até as 23h59 do dia **07/07/2023** via Moodle. **A equipe deve apresentar os resultados obtidos para o professor e a turma entre os dias 10 e 11 de julho de 2023.**

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 em listas duplamente encadeadas

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

- BubbleSort, InsertionSort, SelectionSort, MergeSort e QuickSort

Você deve programar em C++:

(1) Uma versão para cada um dos cinco algoritmos acima usando **lista duplamente encadeada de inteiros**. A lista duplamente encadeada deve ser, obrigatoriamente, programada como uma classe, usando programação orientada a objetos, como fizemos em sala e nos exercícios.

- **Preste bastante atenção:** Como nós implementamos nossas listas encadeadas usando programação orientada a objetos, não temos acesso à estrutura interna da lista. Porém, esses algoritmos ficarão muito mais rápidos se eles forem implementados tendo acesso à estrutura interna da lista encadeada. Assim, uma **dica** para realizar esse trabalho de modo eficiente e decente é implementar os cinco algoritmos acima como funções-membros da classe que implementa a lista duplamente encadeada. (Você pode usar a lista duplamente encadeada iniciada em sala ou pode implementar a sua do zero.) Deste modo, quando você tivesse um objeto lista duplamente encadeada chamado `myList` e invocasse `myList.bubblesort()`, a sua lista `myList` seria ordenada pelo bubblesort que você programou para ela.

Além disso, você deve:

(2) Pesquisar um outro algoritmo de ordenação de seu interesse, diferente dos já apresentados, e implementá-lo usando lista duplamente encadeada.

Obs.: Ou seja, devem ser implementados e executados o total de **6 algoritmos**.

Você e sua dupla 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. O formato da divisão irá impactar na nota da dupla.

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:

- `List.h`: contém a declaração da classe que implementa a lista duplamente encadeada.
- `List.cpp`: contém a implementação da lista duplamente encadeada e das suas funções-membro.
- `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 listas de inteiros aleatoriamente geradas com tamanho N = 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000, 20000, 21000, 22000, 23000, 24000, 25000, 26000, 27000, 28000, 29000, 30000, 31000, 32000, 33000, 34000, 35000, 36000, 37000, 38000, 39000, 40000, 41000, 42000, 43000, 44000, 45000, 46000, 47000, 48000, 49000, 50000, 51000, 52000, 53000, 54000, 55000, 56000, 57000, 58000, 59000, 60000, 61000, 62000, 63000, 64000, 65000, 66000, 67000, 68000, 69000, 70000, 71000, 72000, 73000, 74000, 75000, 76000, 77000, 78000, 79000, 80000, 81000, 82000, 83000, 84000, 85000, 86000, 87000, 88000, 89000, 90000, 91000, 92000, 93000, 94000, 95000, 96000, 97000, 98000 e 99000. 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 com os resultados obtidos. Discuta os resultados e conclusões obtidas. No seu experimento, qual algoritmo teve melhor desempenho? Você sabe dizer por quê? Você sabe interpretar os resultados obtidos?

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, 495 vetores de inteiros gerados aleatoriamente são ordenados usando os algoritmos BubbleSort e CocktailSort (são 495 porque são gerados 5 vetores de tamanho N para cada um dos 99 possíveis valores de N listados acima; logo, $495 = 99 \cdot 5$). 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

[1] Existem diversas formas de medir o tempo de execução de uma função em C++. Pesquise, por exemplo, a biblioteca `<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 muitos algoritmos e deve pensar como vai gerenciar todos eles e como vai organizar os dados e resultados.

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.

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.

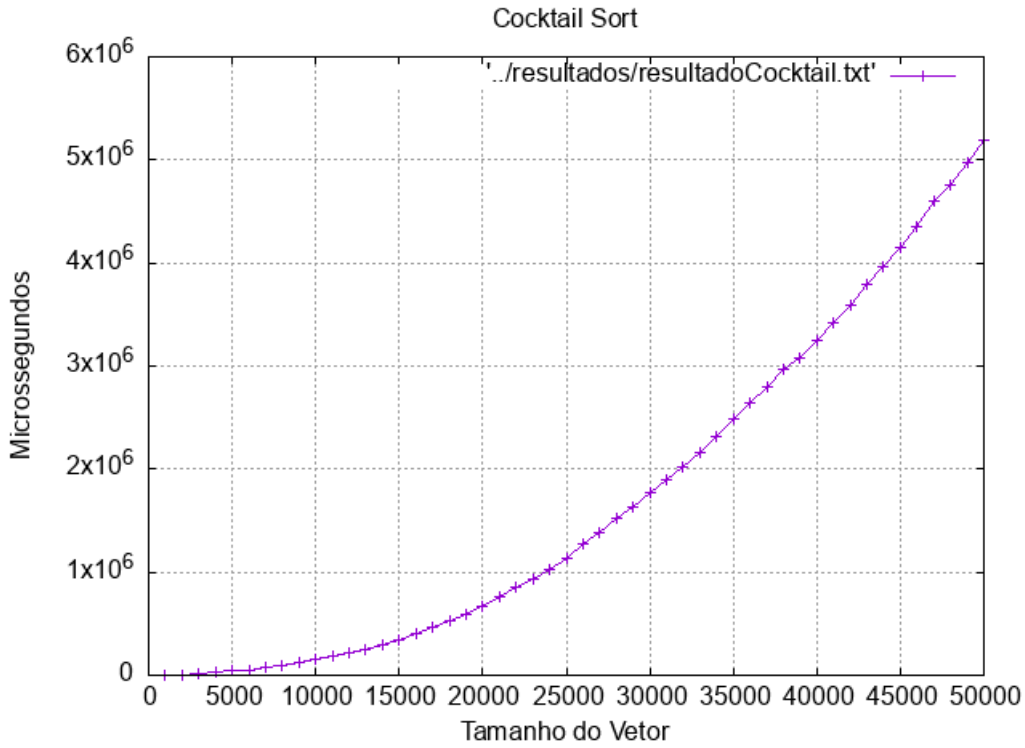

Figura 1: CocktailSort

[3]Os arquivos usados para gerar os gráficos no gnuplot estão na pasta `graficos`. Os arquivos que o programa gnuplot lê são os arquivos com extensão .p que estão na pasta `graficos`, ou seja, os arquivos graphBubble.p, graphCocktail.p e graphBubbleCocktail.p; esses arquivos leem os dados que estão nos arquivos da pasta `resultados` e então geram os gráficos.

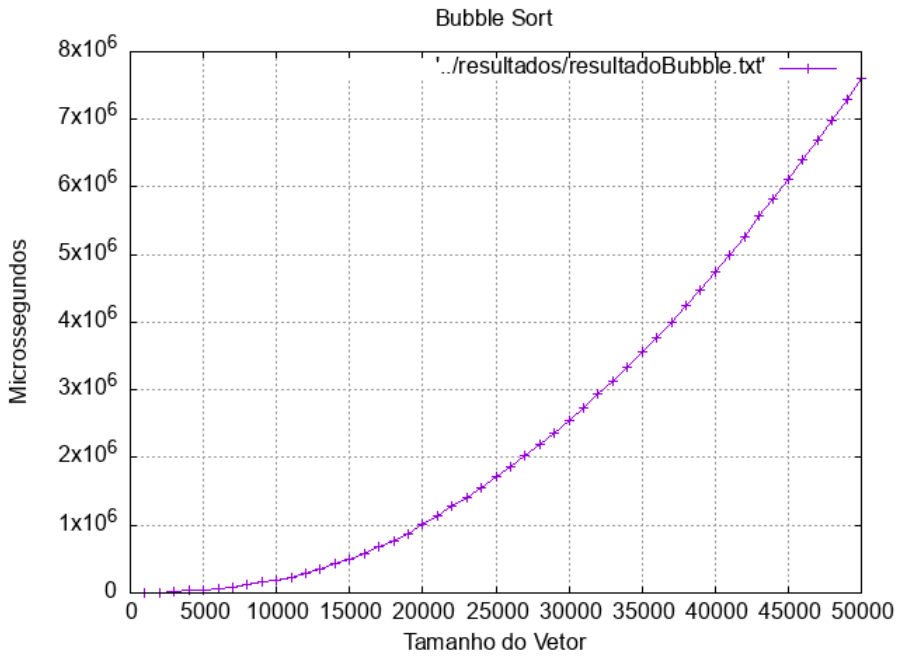

Figura 2: BubbleSort

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.

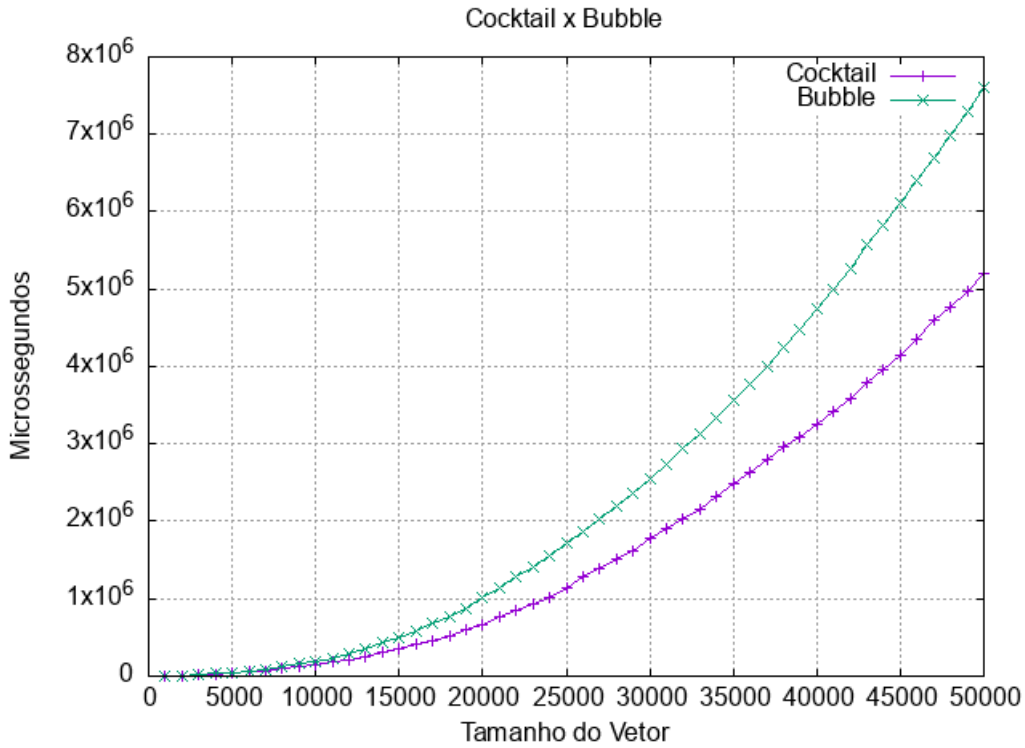

Figura 3: BubbleSort × CockTailSort. O CockTailSort acaba sendo mais rápido que o BubbleSort.

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://youtu.be/0ENnVZsMjL8
- http://ftp.demec.ufpr.br/disciplinas/TM226/Luciano/Cap%C3%ADtulo%2011v5.pdf
- https://www.dicas-l.com.br/arquivo/usando_gnuplot_para_gerar_bons_graficos.php
- 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** contendo:

  (1) Uma descrição breve das principais características de cada um dos 6 algoritmos de ordenação que foram programados, como por exemplo:

  (i) qual a complexidade de pior caso?

  (ii) são estáveis?

  (iii) são algoritmos `in loco`?

  (iv) são recursivos ou iterativos?

  (v) Qual a ideia principal do algoritmo? Como ele faz para ordenar o vetor?

  (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 gráfica entre os tempos de execução dos 6 algoritmos. Se não couber os 6 numa mesma figura, você pode agrupá-los em duplas de figuras. **Atenção:** cada gráfico deve vir acompanhado de um texto seu interpretando os resultados e discursando os motivos de tal gráfico dar tal resultado.

  (5) Uma seção descrevendo como o trabalho foi dividido entre a dupla;

  (6) Uma seção de dificuldades encontradas.

  (7) Uma seção de bibliografia contendo as referências utilizadas. **Se você consultar algum site da internet, vídeo ou livro, coloque esta fonte de pesquisa no seu trabalho. Não há porque omitir as consultas.**

- O relatório deve ser entregue em formato PDF.

- A apresentação do trabalho será feita em horário definido pelo professor.