CEFET-MG Campus Leopoldina
Engenharia de Computação
Classificação e Pesquisa de Dados

ATIVIDADE 3

Semestre: 2023.1 — Entrega: 14/04/2023
Prof. Anderson Grandi Pires — Local: SIGAA

OBSERVAÇÕES

LEIA ATENTAMENTE AS INFORMAÇÕES ABAIXO E SIGA-AS RIGOROSAMENTE

- A atividade é **individual**. Valor da atividade: **6 pontos**.
- A linguagem C++ deverá ser utilizada para implementar a solução para os exercícios práticos. Os exercícios poderão ser desenvolvidos em qualquer ambiente de desenvolvimento integrado (IDE) ou utilizando o prompt de comandos. A sugestão é utilizar o VSCode.
- Você deverá criar uma pasta com o seu nome (Exemplo: anderson), onde serão armazenados os códigos-fonte de todas as questões. Não somente seu primeiro nome sem caracteres especiais para o nome da pasta.
- Cada exercício deverá ser nomeado com o seguinte padrão: exN.c, onde N é o número do exercício (Exemplo: ex1.c, ex2.c, etc.)
- Somente os códigos-fonte e os arquivos cabeçalhos deverão ser entregues.
- Para cada questão, você deverá implementar uma função principal (main) para testar as funcionalidades.
- Os exercícios teóricos deverão ser resolvidos manualmente ou usando softwares de processamento de texto. Tire fotos da resolução ou gere um PDF para entregar e coloque-os dentro da pasta criada.
- Para envio, a pasta deverá ser compactada no formato ZIP e enviada no link da tarefa no SIGAA.
- Utilize boas práticas de programação, tais como indentação e comentários relevantes.
- Inclua um comentário geral nos arquivos que implementam a função `main`, informando o autor e o procedimento para compilação e execução do programa.

ALGUMAS PENALIDADES NA CORREÇÃO

- A identificação de cópia resultará em penalidade de 6 pts para todos os envolvidos.
- Programa possui erros de compilação: penalidade de 6 pts.
- Penalidades por entrega com atraso: 1 dia de atraso (3 pts); 2 dias de atraso (4,5 pts); 3 dias ou mais de atraso (6 pts).
- Programa desorganizado e com problemas na indentação (1 pt).
- Ausência de comentário geral: penalidade de 30% do valor da tarefa relativa ao código sem comentário.

EXERCÍCIOS DE REVISÃO - TEORIA

RESPONDA AS PERGUNTAS ABAIXO PARA AVALIAR A SUA COMPREENSÃO

[0,33 pt] O que significa dizer que $g(n)=O(f(n))?$ Como mostrar que $g(n)=O(f(n))?$ Apresente detalhes para justificar sua resposta.

[0,33 pt] O que significa dizer que $g(n)=\Omega(f(n))?$ Como mostrar que $g(n)=\Omega(f(n))?$ Apresente detalhes para justificar sua resposta.

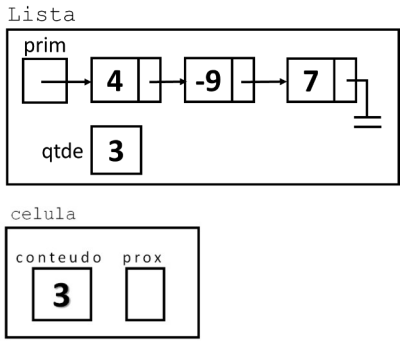
[0,33 pt] O que significa dizer que $g(n)=\Theta(f(n))?$ Como mostrar que $g(n)=\Theta(f(n))?$ Apresente detalhes para justificar sua resposta.

EXERCÍCIO DE APROFUNDAMENTO - TEORIA

DESENVOLVA O EXERCÍCIO ABAIXO PARA APROFUNDAR SEU CONHECIMENTO

[1 pt] Considere o pseudocódigo abaixo, relativo às sub-rotinas de inserção, remoção e busca em **listas lineares sequencias**. Apresente uma análise de complexidade para cada uma dessas sub-rotinas. Justifique sua análise usando a notação assintótica. Sempre que possível, utilize a notação mais restrita (Θ).

Representação



Algumas regras

- Os itens presentes nas células da lista são acessados por meio do campo (membro) `prim`, sendo esse uma referência para o primeiro item da lista. Caso a referência seja nula, a lista estará vazia. O campo (membro) `qtde` indica o número de itens presentes na lista.
- Os itens na lista não estão indexados, porém mantém uma ordem: 1ª item da lista, 2ª item da lista, etc. O acesso à primeira célula da lista é feito por meio do campo `prim`, enquanto que o acesso aos demais itens é feito utilizando as referências armazenadas no campo `prox` de cada célula.
- A representação ao lado compreende somente a parte de dados do TAD Lista Linear Encadeada, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Sub-rotinas

```
lógico invalido(p)
Entrada: lista L
Saída: verdadeiro se parâmetro inválido; falso caso contrário
    retorna p < 0 ou p ≥ L.qtde
```

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens válidos presentes na lista
    retorna L.qtde
```

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver cheia; falso caso contrário
    retorna L.qtde = 0
```

```
lógico buscar(L, ctd)
Entrada: lista L, conteúdo ctd a ser buscado
Saída: sucesso ou falha na operação
    temp ← L.prim
    enquanto temp ≠ NULO faça
        se temp.conteudo = ctd então
          retorna verdadeiro
        temp ← temp.prox
    retorna falso
```

```
lógico inserir_inicio(L, e)
Entrada: lista L, elemento e a ser inserido
Saída: sucesso ou falha na operação
cel: célula a ser inserida
    se invalido(ctd) então
          retorna falso
    cel.conteudo ← ctd
    cel.prox ← L.prim
    L.prim ← cel
    L.qtde ← L.qtde + 1
    retorna verdadeiro
```

```
lógico inserir_meio(L, ctd, k)
Entrada: lista L, conteúdo ctd, k-ésima posição
Saída: sucesso ou falha na operação
cel: célula a ser inserida
    se invalido(ctd) ou invalido(k) então
          retorna falso
    cel.conteudo ← ctd
    se k = 1 então
          cel.prox ← L.prim
          L.prim ← cel
    senão
          temp ← L.prim
          para i ← 1 até k-2 passo 1 faça
                temp ← temp.prox
          cel.prox ← temp.prox
          temp.prox ← cel
    L.qtde ← L.qtde + 1
    retorna verdadeiro
```

```
lógico inserir_fim(L, ctd)
Entrada: lista L, conteúdo ctd a inserir
Saída: sucesso ou falha na operação
cel: célula a ser inserida
    se invalido(ctd) então
          retorna falso
    cel.conteudo ← ctd
    cel.prox ← NULO
    se esta_vazia(L) então
          L.prim ← cel
    senão
          temp ← L.prim
          enquanto temp.prox ≠ NULO faça
                temp ← temp.prox
          temp.prox ← cel
    L.qtde ← L.qtde + 1
    retorna verdadeiro
```

```
lógico remover_inicio(L)
Entrada: lista L
Saída: sucesso ou falha na operação
    se vazia(L) então
          retorna falso
    L.prim ← L.prim.prox
    L.qtde ← L.qtde - 1
retorna verdadeiro
```

```
lógico remover_meio(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação
    se vazia(L) ou invalido(k) então
          retorna falso
    temp ← L.prim
    se L.qtde = 1 então
          L.prim ← NULO
    senão
          para i ← 1 até k-2 passo 1 faça
                temp ← temp.prox
          temp.prox ← temp.prox.prox
    L.qtde ← L.qtde - 1
    retorna verdadeiro
```

```
lógico remover_fim(L)
Entrada: lista L
Saída: sucesso ou falha na operação
    se vazia(L) então
        retorna falso
    se L.qtde = 1 então
        L.prim ← NULO
    senão
        temp ← L.prim
        enquanto temp.prox.prox ≠ NULO faça
            temp ← temp.prox
        temp.prox ← NULO
    L.qtde ← L.qtde – 1
    retorna verdadeiro
```

EXERCÍCIO DE REVISÃO - PRÁTICA

DESENVOLVA O EXERCÍCIO ABAIXO PARA AVALIAR A SUA COMPREENSÃO

[1 pt] Preencha os espaços em branco no código-fonte abaixo para completá-lo com as instruções necessárias para estimar, empiricamente, o tempo de execução da função `ordenar`. Crie o arquivo cabeçalho (`funcoes.h`) e o arquivo com os códigos-fonte (`funcoes.cpp`) com a implementação das funções.

```
#include <bits/stdc++.h>
#include <chrono>
#include "funcoes.h"
using namespace std;
using namespace std::_________;

void exibir(_______ *v, int n);  // vetor de inteiros (inteiros de 2 bytes)
void preencher(_______ *v, int n, _______ min, _______ max);
void ordenar(_______ *v, int n);

const int N = 1000000;

int main() {
    srand(time(_______));
    _______ vetor[N];
    preencher(vetor, N, 10, 20);
    _______ inicio = _______::now();
    ordenar(vetor, N);          // implemente a ordenação por flutuação
    _______ fim = _______::now();
    auto intervalo = fim - inicio;
    cout << duration_cast<seconds>(intervalo).count() << "s" << endl;
    cout << duration_cast<___________>(intervalo).count() << "ms" << endl;
    cout << duration_cast<microseconds>(intervalo).count() << "us" << endl;
    cout << duration_cast<___________>(intervalo).count() << "ns" << endl;
    cout << duration<double>{intervalo}.count() << "s" << endl;

    return 0;
}
```

EXERCÍCIO DE APROFUNDAMENTO - PRÁTICA

DESENVOLVA O EXERCÍCIO ABAIXO PARA APROFUNDAR SEU CONHECIMENTO

1) [1,5 pt] Implemente uma função com o algoritmo de ordenação abaixo.

```
para i ← 1 até tam-1 faça
    chave ← v[i]
    j ← i - 1
    enquanto j ≥ 0 e v[j] > chave faça
        v[j + 1] ← v[j]
        j ← j - 1
    v[j + 1] ← chave
```

Por intermédio de análise empírica, faça um experimento que possibilite estimar a complexidade de tempo do algoritmo. Esta tarefa pode ser feita com os seguintes passos:

(a) Crie um vetor com N grande (`N = 1000000`, por exemplo).

(b) Utilize a função para ordenar partes do vetor de tamanho crescente. Por exemplo, ordene os 25000 primeiros elementos, depois os 50000 primeiros e, assim, sucessivamente até alcançar o valor de N. O conteúdo da parte já ordenada do vetor deve ser restaurada para os valores iniciais, de modo a tentar criar condições semelhantes para a exploração do algoritmo. Para cada ordenação efetuada, grave em arquivo o tamanho da parte ordenada (N = 25000, por exemplo) e a estimativa de tempo gerada pela função apropriada em C++.

(c) Utilize as instruções em python abaixo para ler o arquivo e gerar um gráfico, de forma a possibilitar uma visualização do comportamento do algoritmo. Apresente o gráfico e discuta os resultados obtidos.

```
#%%
#pode executar aqui => https://www.codabrainy.com/en/python-compiler/

import matplotlib.pyplot as plt
import numpy as np

X, Y = np.loadtxt('dados.txt', delimiter=' ', unpack=True)
plt.scatter(X, Y)
plt.title('Gráfico da análise experimental')
plt.xlabel('n')
plt.ylabel('time (s)')
plt.show()
#%%
```

(d) Efetue uma análise assintótica do algoritmo e compare-a com o comportamento obtido com a análise empírica. Justifique sua análise usando a notação assintótica e, sempre que possível, utilize a notação mais restrita (Θ).

2) [1,5 pt] Os arquivos `intro08.cpp` e `heapsort.cpp` na pasta `aula05_20230405` possuem códigos que podem auxiliar a resolver este exercício.

(a) Implemente a função ordem com o propósito de informar se um vetor está ou não ordenado, retornando `true` ou `false`, respectivamente.

(b) Por intermédio de análise empírica, faça um experimento que possibilite estimar a complexidade de tempo das funções: `maxheap`, `heapify` e `heapsort`. Esta tarefa pode ser feita seguindo os passos indicados no exercício anterior. Apresente um gráfico e discuta os resultados obtidos.

(c) Implemente a função `exibir` de modo que ela possa ser utilizada para colorir parte de um vetor passado por parâmetro. Desse modo, mostre um passo-a-passo da função de ordenação do vetor usando cores para facilitar a visualização: os números na cor encontram-se na parte ordenada do vetor. Segue um exemplo abaixo.

```
60 50 21 32 77 55 35 99 44 81
77 60 55 50 44 21 35 32 81 99
60 50 55 32 44 21 35 77 81 99
55 50 35 32 44 21 60 77 81 99
50 44 35 32 21 55 60 77 81 99
44 32 35 21 50 55 60 77 81 99
35 32 21 44 50 55 60 77 81 99
32 21 35 44 50 55 60 77 81 99
21 32 35 44 50 55 60 77 81 99
21 32 35 44 50 55 60 77 81 99
```