UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
Rua Dom Manoel de Medeiros, s/n – Dois Irmãos 52171-900 Recife-PE
Fone: 81 3302 1000 www.dc.ufrpe.br

DISCIPLINA: Algoritmos e Estruturas de Dados	**CÓDIGO**: 06214
DEPARTAMENTO: Computação	**ÁREA**: Informática
CURSO: Bacharelado em Ciência da Computação / Licenciatura em Computação	
PROFESSOR RESPONSÁVEL: Luciano Demétrio Santos Pacífico	
DATA MÁXIMA DE ENTREGA: **26-10-2025**	

Regras da Lista de Exercícios 02

1. **Não é permitido o uso de Estruturas de Dados prontas de Linguagens de Programação**. O aluno deve implementar suas próprias Estruturas de Dados. Na Linguagem de Programação **C**, deve-se usar **structs**. Nas demais Linguagens de Programação permitidas (vide **Regras da Disciplina**), deve-se usar **classes**.

2. **Não é permitido o uso de Algoritmos e comandos otimizados prontos de Linguagens de Programação (funções, métodos, laços de repetição otimizados, etc.)**. Todos os algoritmos solicitados devem ser implementados pelos alunos como **procedimentos** (funções, métodos, etc.).

3. As questões que solicitam **escrita de código** devem ser resolvidas **apenas através dos recursos oferecidos pela pseudolingaugem definida para a disciplina, e dos recursos equivalentes em Linguagens de Programação reais**, sendo eles: variáveis, constantes e tipos primitivos, expressões, estruturas condicionais, estruturas de repetição, sub-rotinas, estruturas de dados homogêneas (**Arrays**) e estruturas de dados heterogêneas (registros – classes e sctructs).

4. Para esta Lista de Exercícios, os **Arrays** devem ser implementados através de **Alocação Estática**. Não será permitido o uso de **Estruturas de Dados dinâmicas** implementadas em Linguagens de Programação, como os **Vectors** e **Lists** da Linguagem de Programação **Java**, por exemplo. Deve-se usar **arrays**.

5. Como a Linguagem de Programação **Python** não suporta **arrays estáticos**, o aluno que optar por usar esta Linguagem deverá usar **Lists**, **única e exclusivamente para simular o comportamento de arrays estáticos**, de acordo com os seguintes critérios:

    a. A estrutura deve ser **inicializada** através da operação **<nome_da_lista_estatica> = [None] * M**, onde **<nome_da_lista_estatica>** refere-se ao **nome atribuído pelo aluno à variável que representará a Lista Estática**, e **M** refere-se à **capacidade máxima da estrutura estática**;

    b. O limite máximo de M objetos deve ser controlado **através de código**;

    c. A Estrutura deve ser manipulada como se fosse um **array estático**, com os **procedimentos escritos pelo aluno**, **não sendo permitido** o uso de métodos, funções ou otimizações oferecidos pela Linguagem Python.

6. **Regra de Ouro**: **Todos os alunos envolvidos em cópias terão suas notas ANULADAS nas referidas questões.**

7. Apenas o código “.c”, “.cpp”, “.java”, “.py”, etc. deve ser enviado ao professor para cada questão. Deve-se enviar **um único arquivo resposta por questão**, que conterá todas as classes/estruturas e procedimentos necessários para a solução da questão. Todos os arquivos devem ser enviados **em uma única pasta, “zipados”**.

8. O arquivo de resposta com o código para cada questão deve ser nomeado na forma “L#Q%.c”, “L#Q%.java”, etc., onde “#” refere-se ao número da lista e “%” refere-se ao número da questão (ex.: L1Q2.c, para o arquivo de resposta da segunda questão da primeira lista, usando a Linguagem C).

9. A resposta desta Lista de Exercício deve ser submetida **unicamente através da tarefa criada no Google Classroom para este propósito**.

Lista de Exercícios 02 – Listas Lineares e Ordenação

1. Escreva os algoritmos, usando **uma Linguagem de Programação**, de **Visualização** (**Impressão**), **Busca**, **Inserção** e **Remoção** em uma **Lista Linear Sequencial Ordenada em Ordem Crescente** com capacidade máxima **inicial** para **M** elementos, de acordo com o especificado: (3.0 pontos)

    a. Defina as Estruturas de Dados básicas necessárias.
    b. O tamanho **M** inicial deve ser definido pelo usuário no início do uso da aplicação (parâmetro).
    c. A lista deverá estar inicialmente vazia.
    d. Na operação de impressão, posições não ocupadas da estrutura devem ser impressas como **NIL**.
    e. A operação de **inserção** deve ser implementada de modo que, **sempre** que uma tentativa de inserção for realizada quando a lista **já contiver M elementos**, um novo array estático deve ser criado, com **capacidade máxima 2*M**, e todos os elementos contidos no array original **devem ser copiados** para o novo array. Após a cópia, o array original deve ser **substituído** pelo novo, e o tamanho da lista nova deve ser **atualizado** e **impresso para o usuário**.
    f. Deve haver uma forma de **interação do usuário** com o programa, permitindo que esse usuário execute as operações de visualização, busca, inserção e remoção, **de acordo com a sua necessidade**.
    g. Os procedimentos devem imprimir **mensagens de erro**, caso o usuário tente realizar **operações inválidas** (com remoções de dados inexistentes na lista).

2. Implemente, em uma **Linguagem de Programação**, o algoritmo **Mergesort**, que recebe **um vetor de dados de tamanho M**, **completamente preenchido pelo usuário**, como entrada. Implemente as **Estruturas de Dados básicas** necessárias. Imprima, em linha própria, o **vetor de dados original** (passado como entrada ao algoritmo). A cada chamada ao procedimento **`mergesort`** (desde sua chamada inicial) deve-se imprimir a mensagem "**mergesort <p> <r>: empilhado!**", em linha própria, onde **<p>** refere-se ao valor do parâmetro de limite mais à esquerda (limite inferior) e **<r>** refere-se ao valor do parâmetro de limite mais à direita (limite superior) da chamada ao mergesort. Ao final de cada chamada ao procedimento **`mergesort`** deve-se imprimir a mensagem "**mergesort <p> <r>: dsempilhado!**", em linha própria. A cada chamada ao procedimento **`merge`** deve-se imprimir, em linha própria, a mensagem "**merge <p> <q> <r>**", onde **<q>** refere-se ao valor do parâmetro de "metade" do subvetor atual. Ao final de **cada execução do procedimento `merge`**, o vetor de dados modificado deve ser **impresso** em uma linha própria. Ao fim da execução do algoritmo mergesort, deve-se imprimir o número total de chamadas ao procedimento mergesort, contando com a primeira chamada. (2.0 pontos)

3. Implemente, em uma **Linguagem de Programação**, o algoritmo **Quicksort**, de acordo com a variação apresentada em sala de aula (**pivô como elemento mais à direita**), que recebe **um vetor de dados de tamanho M**, **completamente preenchido pelo usuário**, como entrada. Implemente as **Estruturas de Dados básicas** necessárias. Imprima o **vetor de dados original** (passado como entrada ao algoritmo), em uma linha própria. A cada chamada ao procedimento **`quicksort`** (desde sua chamada inicial) deve-se imprimir a mensagem "**quicksort <p> <r>: empilhado!**", em linha própria, onde **<p>** refere-se ao valor do parâmetro de limite mais à esquerda (limite inferior) e **<r>** refere-se ao valor do parâmetro de limite mais à direita (limite superior) da chamada ao Quicksort. Ao final de cada chamada ao procedimento **`quicksort`** deve-se imprimir a

mensagem "**quicksort <p> <r>: dsempilhado!**" em linha própria. A **cada chamada do procedimento `particionar`**, deve-se imprimir a mensagem "**particionar <p> <r>**", em linha própria. Ao final de **cada execução do procedimento `particionar`**, o vetor de dados modificado deve ser **impresso** em uma linha própria. Ao final da execução do **`quicksort,`** imprima o **número total de vezes** que o procedimento **`trocar`** é utilizado durante a execução das chamadas ao procedimento **`particionar`**. (2.5 pontos)

4. Implemente, em uma **Linguagem de Programação**, o algoritmo **Heapsort**, que recebe **um vetor de dados de tamanho M**, **completamente preenchido pelo usuário**, como entrada. Implemente as **Estruturas de Dados básicas** necessárias. Imprima o **vetor de dados original** (passado como entrada ao algoritmo) em uma linha própria. Imprima o **vetor de dados gerado após** a execução do procedimento **`construirMaxHeap`** em linha uma própria. A **cada chamada ao procedimento `maxHeapfy`**, deve-se imprimir "**maxheapfy <i>**", em linha própria, onde **<i>** refere-se ao índice do objeto no vetor de dados a ser verificado. Ao final de **cada execução do procedimento `maxHeapfy`**, o vetor de dados modificado deve ser **impresso** em linha própria. Imprima, em linha própria, o **número total de vezes** que o procedimento **`trocar`** é utilizado durante a execução das chamadas ao **`maxHeapfy`** ao longo da execução do **Heapsort**. Imprima, em linha própria, o **número total de vezes** que o procedimento **`maxHeapfy`** é utilizado durante a execução do **Heapsort**. (2.5 pontos).