# Construção e Análise de Algoritmo

Instituto Federal de Educação, Ciência e Tecnologia do Ceará
(IFCE) - Campus Tianguá
Bacharelado em Ciência da Computação

Dhiego Cavalcanti da Silveira
João Pedro dos Santos Jácome da Silva
Luís Guilherme Ferreira da Costa
Maria Eduarda Araujo Sales
Victor Oliveira do Nascimento

**Professor Me. Adonias Caetano**

Tianguá (CE), 4 de agosto de 2025

# Sumário

# Questões Gerais

**A) Implemente em Python todos os dez algoritmos de ordenação ensinados em sala de aula, realizando experimentos que avaliem o tempo de execução para ordenar de acordo com as seguintes regras:**

I. Serão nove vetores com os seguintes tamanhos para cada um: 1000, 3000, 6000, 9000, 12000, 15000, 18000, 21000, 24000.

II. Os valores armazenados nos nove vetores serão números inteiros gerados aleatoriamente.

III. Usar a biblioteca matplotlib.pyplot

IV. Plotar um gráfico geral, isto é, todas curvas de tempo de execução dos métodos de ordenação juntos, comparando o desempenho de execução dos algoritmos de acordo com o tamanho do vetor.

V. Apresente gráficos específicos que considere conveniente para facilitar a análise comparativa dos métodos, tais como gráficos por métodos com desempenho de execução semelhantes ou com base nos tamanhos.

# Bubble Sort

```python
def bubble_sort(lista):
    n = len(lista)
    for i in range(n-1):
        for j in range(n-1-i):
            if lista[j] > lista[j+1]:
                lista[j], lista[j+1] = lista[j+1], lista[j]
    return lista
```

# Insertion Sort

```python
def insertionSort(lista):
    tamanho = len(lista)

    for i in range(1, tamanho):
        elemento = lista[i]
        j = i - 1
        while j >= 0 and elemento < lista[j]:
            lista[j+1] = lista[j]
            j = j - 1
        lista[j+1] = elemento

    return lista
```

# Selection Sort

```
1  def selectionSort(lista):
2
3      tamanho = len(lista)
4
5      for i in range(tamanho):
6          j = i + 1
7          minimo = i
8          while j < tamanho:
9              if lista[minimo] > lista[j]:
10                 minimo = j
11             j = j + 1
12
13     if lista[i] != lista[minimo]:
14         lista[i], lista[minimo] = lista[minimo], lista[i]
15
16     return lista
```

# Merge Sort

```python
def mergeSort(lista):

    n = len(lista)
    if n <= 1:
        return lista

    metade = n//2
    listaEsquerda = mergeSort(lista[:metade])
    listaDireita = mergeSort(lista[metade:])
    return merge(listaEsquerda, listaDireita)
```

# Merge Sort

```python
def merge(esquerda, direita):
    i = j = 0
    resultado = []

    while i < len(esquerda) and j < len(direita):
        if esquerda[i] < direita[j]:
            resultado.append(esquerda[i])
            i = i + 1
        else:
            resultado.append(direita[j])
            j = j + 1

    resultado.extend(esquerda[i:])
    resultado.extend(direita[j:])

    return resultado
```

# Quick Sort

```python
def quickSort(lista):
    quickSortOrdenar(lista, 0, len(lista) - 1)
    return lista

def quickSortOrdenar(lista, inicio, fim):

    if inicio >= fim:
        return

    pivot = particao(lista, inicio, fim)
    quickSortOrdenar(lista, inicio, pivot - 1)
    quickSortOrdenar(lista, pivot + 1, fim)
```

# Quick Sort

```python
def particao(lista, inicio, fim):

    pivot = lista[inicio]
    i = inicio + 1

    for j in range(inicio + 1, fim + 1):
        if lista[j] < pivot:
            lista[i], lista[j] = lista[j], lista[i]
            i = i + 1

    lista[inicio], lista[i - 1] = lista[i - 1], lista[inicio]
    return i - 1
```

# Counting Sort

```python
def countingSort(lista):

    minimo = min(lista)
    maximo = max(lista)
    tamanho = maximo - minimo + 1
    arrayCopia = [0 for i in range(len(lista))]
    arrayContagem = [0 for i in range(tamanho)]

    for i in range(0, len(lista)):
        arrayContagem[lista[i] - minimo] += 1

    for i in range(1, len(arrayContagem)):
        arrayContagem[i] += arrayContagem[i - 1]

    for i in range(len(lista) - 1, -1, -1):
        valor = lista[i]
        arrayContagem[valor - minimo] -= 1
        posicao = arrayContagem[valor - minimo]
        arrayCopia[posicao] = valor

    return arrayCopia
```

# Radix Sort

```
1  def radixSort(lista):
2
3      negativos = [-x for x in lista if x < 0]
4      nao_negativos = [x for x in lista if x >= 0]
5      negativos_ordenados = []
6      if negativos:
7          minimo = max(negativos)
8          exp = 1
9          while minimo // exp > 0:
10             negativos = contagem(negativos, exp)
11             exp *= 10
12         negativos_ordenados = [-x for x in reversed(
       negativos)]
13     if nao_negativos:
14         maximo = max(nao_negativos)
15         exp = 1
16         while maximo // exp > 0:
17             nao_negativos = contagem(nao_negativos, exp)
18             exp *= 10
19
20     return negativos_ordenados + nao_negativos
```

# Radix Sort

```python
def contagem(lista, exp):

    arrayContagem = [0 for i in range(10)]
    resultado = [0 for i in range(len(lista))]

    for i in range(0, len(lista)):
        digito = (lista[i] // exp) % 10
        arrayContagem[digito] += 1

    for i in range(1, 10):
        arrayContagem[i] += arrayContagem[i - 1]

    for i in range(len(lista) - 1, -1, -1):
        digito = (lista[i] // exp) % 10
        arrayContagem[digito] -= 1
        resultado[arrayContagem[digito]] = lista[i]

    return resultado
```

# Bucket Sort

```python
def bucketSort(lista):

    maximo = max(lista)
    minimo = min(lista)
    numBaldes = int(math.sqrt(len(lista))) or 1
    intervalo = (maximo - minimo + 1) / numBaldes
    baldes = [[] for _ in range(numBaldes)]

    for valor in lista:
        index = int((valor - minimo) / intervalo)
        if index == numBaldes:
            index -= 1
        baldes[index].append(valor)

    resultado = []
    for balde in baldes:
        balde = insertionSort(balde)
        resultado.extend(balde)

    return resultado
```

# Shell Sort

```python
def shellSort(lista):

    n = len(lista)
    h = len(lista) // 2

    while h > 0:
        for i in range(h, n):
            valor = lista[i]
            j = i
            while j >= h and valor < lista[j - h]:
                lista[j] = lista[j - h]
                j = j - h
            lista[j] = valor
        h = int(h / 2)

  return lista
```

# Heap Sort

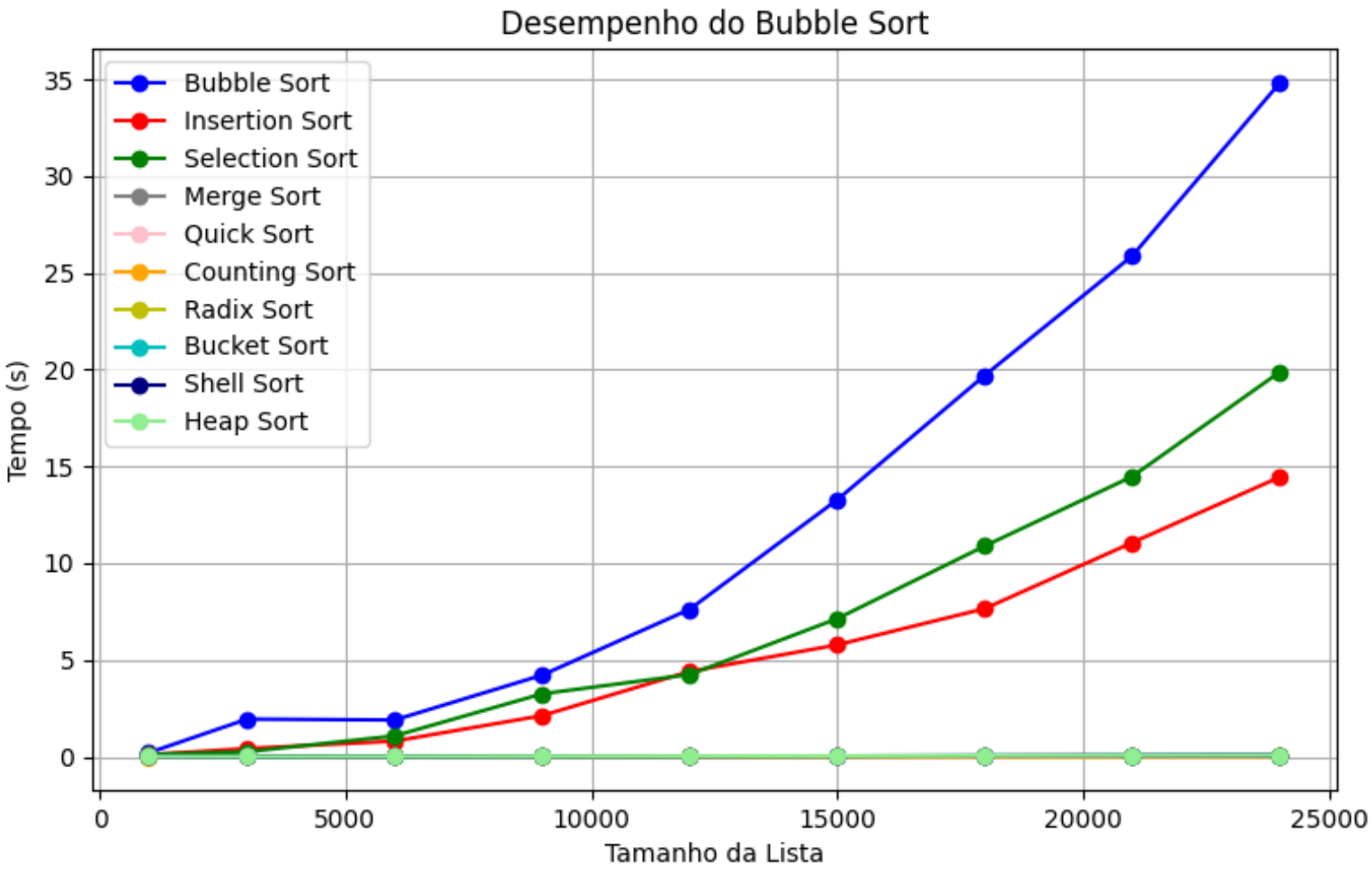
```python
def heapify(lista, n, i):

    maior = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and lista[left] > lista[maior]:
        maior = left

    if right < n and lista[right] > lista[maior]:
        maior = right

    if maior != i:
        lista[maior], lista[i] = lista[i], lista[maior]
        heapify(lista, n, maior)
```

# Heap Sort

```python
def heapSort(lista):

    n = len(lista)

    for i in range(n//2 - 1, -1, -1):
        heapify(lista, n, i)

    for i in range(n - 1, 0, -1):
        lista[0], lista[i] = lista[i], lista[0]
        heapify(lista, i, 0)

    return lista
```

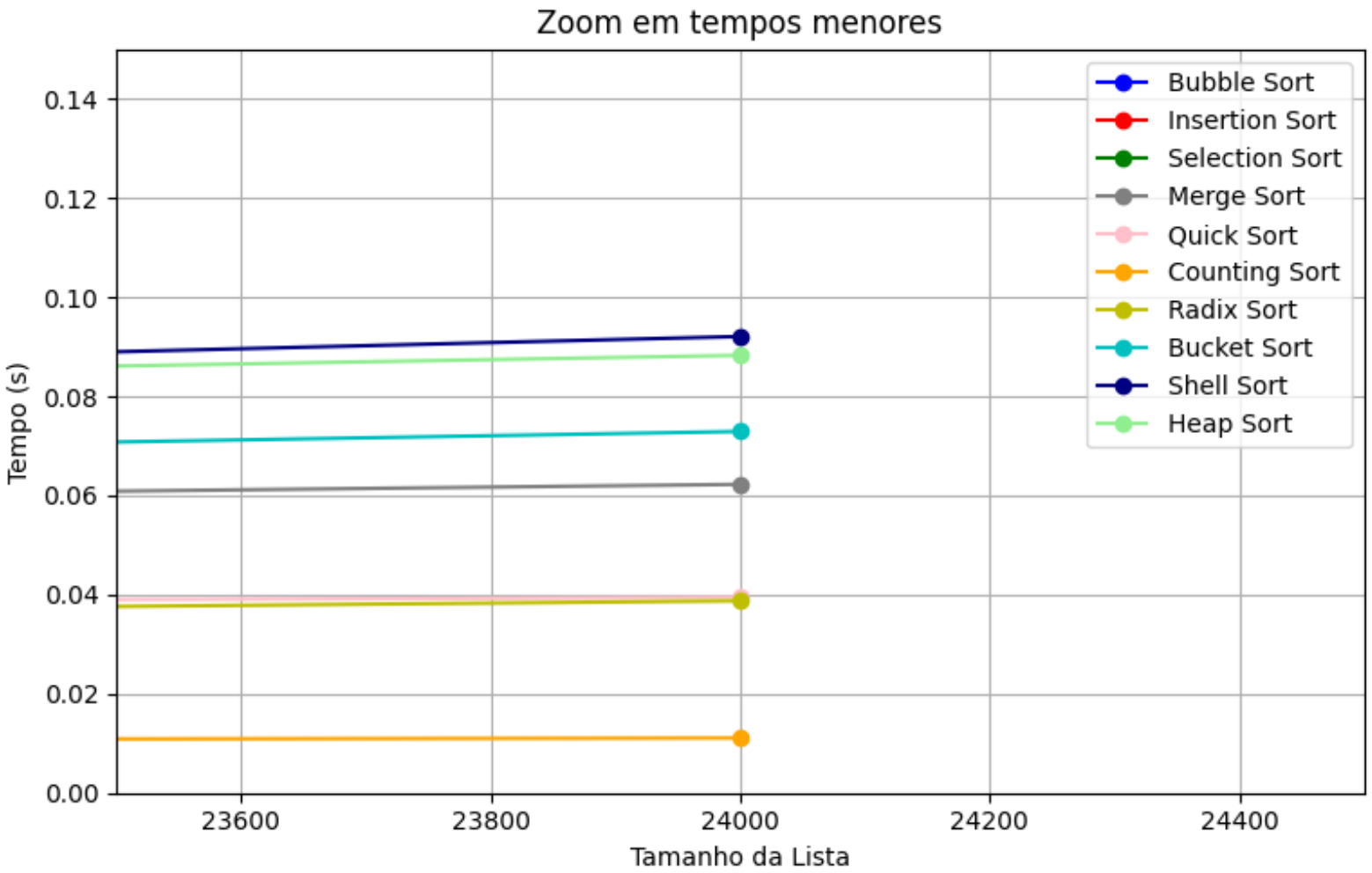
# Comparação

## Gráfico geral



# Comparação

## Gráfico com Zoom

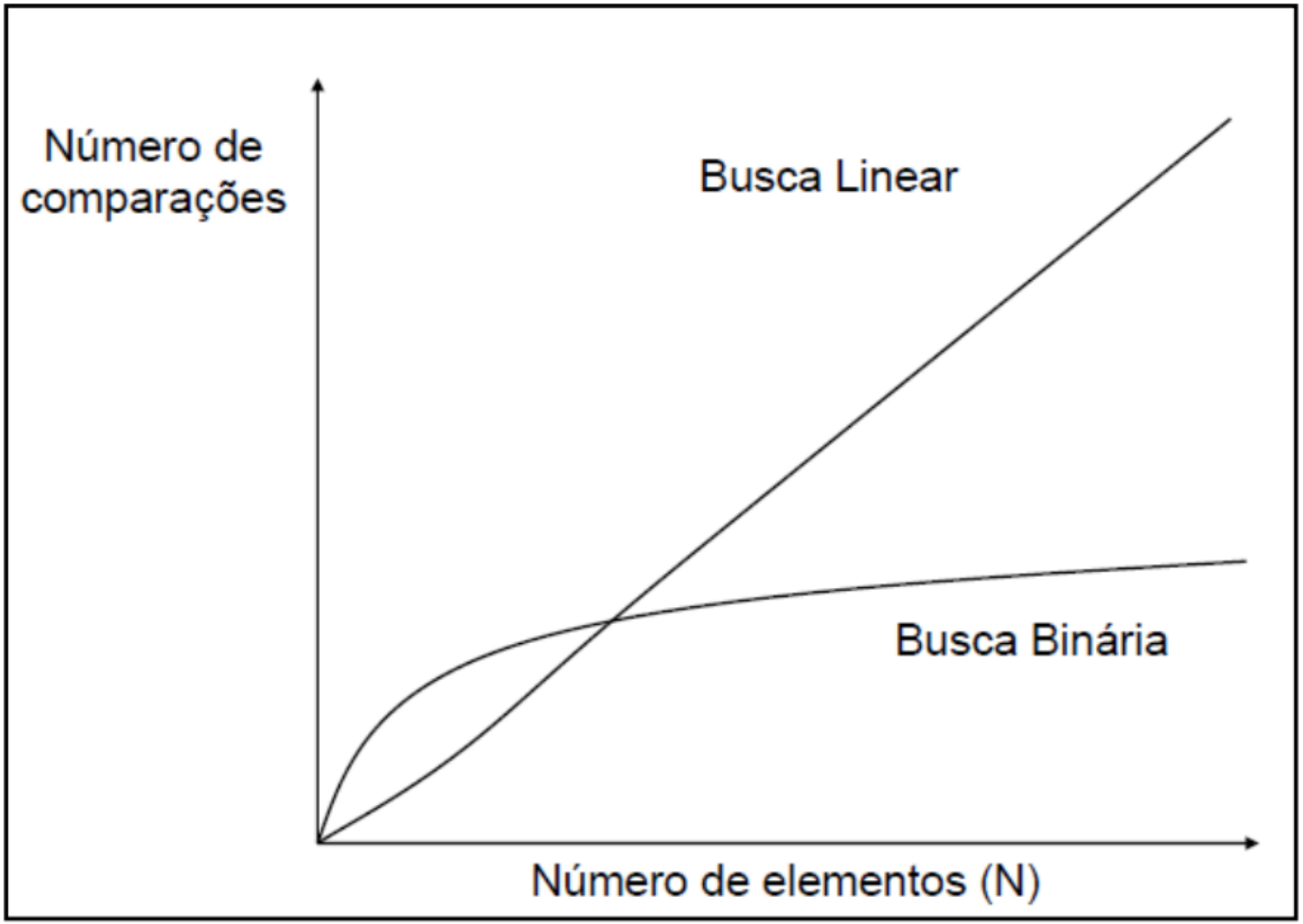


# Questões Gerais

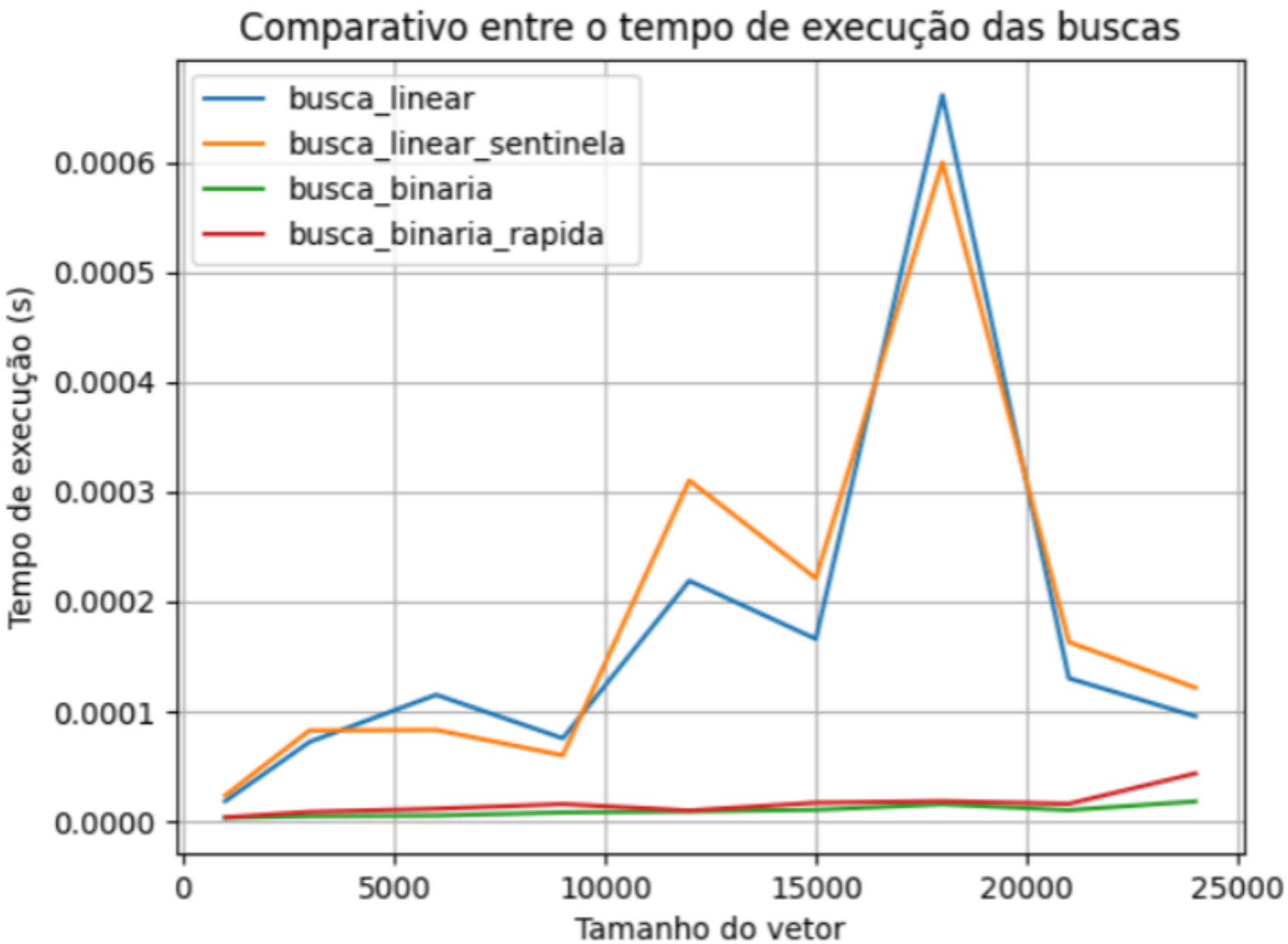
B) Implemente em Python a busca linear convencional e com sentinela, busca binária convencional e rápida realizando experimentos que avaliem o tempo de execução para encontrar uma chave de acordo com as seguintes regras:

I. Serão nove vetores com os seguintes tamanhos para cada um: 1000, 3000, 6000, 9000, 12000, 15000, 18000, 21000, 24000;
II. Os valores armazenados nos nove vetores serão números inteiros gerados aleatoriamente;
III. Aplique um método de ordenação linear nas buscas do tipo binária;
IV. O elemento chave a ser buscado pode ser informado pelo usuário ou gerado aleatoriamente;
V. Usar a biblioteca matplotlib.pyplot;
VI. Conforme as Figuras 3 e 4, plote um gráfico comparando o tempo de execução dos algoritmos de acordo com o tamanho do vetor;
VII. Faça comentários objetivos e sucintos sobre os gráficos.

## Figura 3 - Exemplo do gráfico ilustrativo



# Figura 4 - Exemplo do gráfico de comparação dos métodos de busca



# Bibliotecas

```
1 import random
2 import time
3 import math
4 import matplotlib.pyplot as plt
```

# Insertion Sort  Algoritmo de Ordenação

```python
def insertion_sort(vetor):
    for i in range(1, len(vetor)):
        chave = vetor[i]
        j = i - 1
        while j >= 0 and vetor[j] > chave:
            vetor[j + 1] = vetor[j]
            j -= 1
        vetor[j + 1] = chave
```

# Busca Linear

```python
def busca_linear(vetor, chave):
    for i in range(len(vetor)):
        if vetor[i] == chave:
            return i
    return -1
```

# Busca Linear Sentinela

```python
def busca_linear_sentinela(vetor, chave):
    vetor.append(chave)
    i = 0
    while vetor[i] != chave:
        i += 1
    vetor.pop()
    return i if i < len(vetor) else -1
```

# Busca Binária

```python
def busca_binaria(vetor, chave):
    esquerda, direita = 0, len(vetor) - 1
    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        if vetor[meio] == chave:
            return meio
        elif vetor[meio] < chave:
            esquerda = meio + 1
        else:
            direita = meio - 1
    return -1
```

# Busca Binária Rápida

```python
def busca_binaria_rapida(vetor, chave):
    esquerda, direita = 0, len(vetor) - 1
    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        if vetor[meio] < chave:
            esquerda = meio + 1
        else:
            direita = meio - 1
    return esquerda if esquerda < len(vetor) and vetor[
    esquerda] == chave else -1
```

# Parâmetros

```
tamanhos = [1000, 3000, 6000, 9000, 12000, 15000, 18000,
    21000, 24000]
tempos = {
    "busca_linear": [],
    "busca_linear_sentinela": [],
    "busca_binaria": [],
    "busca_binaria_rapida": []
}
```

# Entrada da chave

```
entrada = input("Digite o valor da chave a ser buscada (ou
    pressione Enter para gerar aleatoriamente): ")
chave_usuario = int(entrada) if entrada.strip().isdigit()
    else None
}
```

# Execução dos testes

```
1 for tamanho in tamanhos:
2     print(f"Executando testes para vetor de tamanho {tamanho
      }...")
3     vetor = random.sample(range(0, tamanho * 2), tamanho)
4     chave = chave_usuario if chave_usuario is not None else
      random.choice(vetor)
```

# Linear

```
1     inicio = time.perf_counter()
2     busca_linear(vetor[:], chave)
3     tempos["busca_linear"].append(time.perf_counter() -
      inicio)
```

# Linear com sentinela

```
    inicio = time.perf_counter()
    busca_linear_sentinela(vetor[:], chave)
    tempos["busca_linear_sentinela"].append(time.
    perf_counter() - inicio)
```

# Ordenar para busca binária

```
1     vetor_ordenado = vetor[:]
2     insertion_sort(vetor_ordenado)
```

# Binária

```
1     inicio = time.perf_counter()
2     busca_binaria(vetor_ordenado[:], chave)
3     tempos["busca_binaria"].append(time.perf_counter() -
      inicio)
```

# Binária Rápida

```
1     inicio = time.perf_counter()
2     busca_binaria_rapida(vetor_ordenado[:], chave)
3     tempos["busca_binaria_rapida"].append(time.perf_counter
      () - inicio)
```

# Gráfico de tempo de execução

```
    plt.figure(figsize=(10, 6))
    for metodo, valores in tempos.items():
        plt.plot(tamanhos, valores, marker='o', label=metodo
    .replace("_", " ").capitalize())

    plt.xlabel("Tamanho do vetor")
    plt.ylabel("Tempo de execução (s)")
    plt.title("Comparativo de Tempo de Execução das Buscas")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

# Gráfico do número de comparações

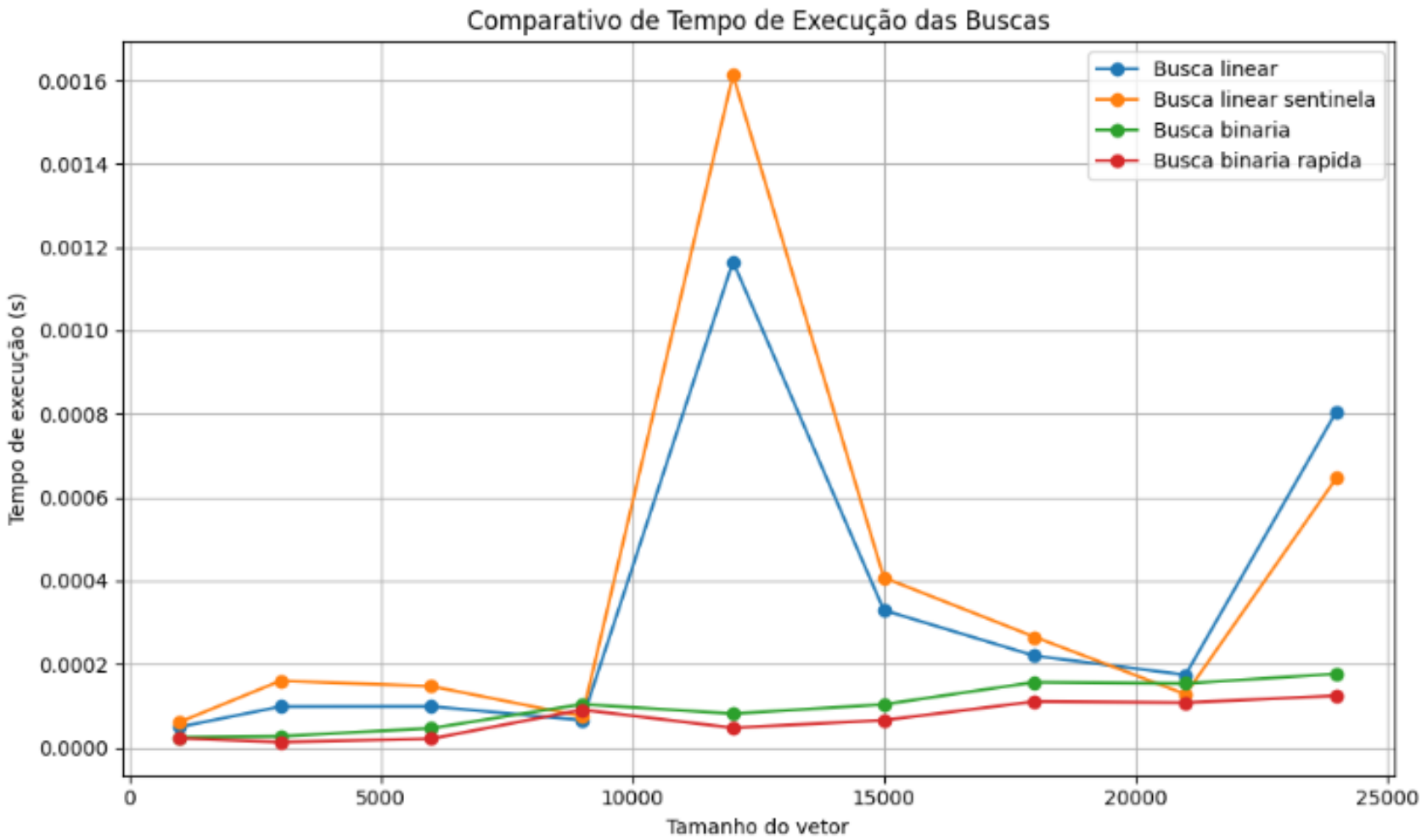
```
    comparacoes = {
        "Busca Linear": [n / 2 for n in tamanhos],
        "Busca Linear Sentinela": [n / 2 for n in tamanhos],
        "Busca Binária": [math.log2(n) for n in tamanhos],
        "Busca Binária Rápida": [math.log2(n) for n in
    tamanhos]
    }

    plt.figure(figsize=(10, 6))
    for metodo, valores in comparacoes.items():
        plt.plot(tamanhos, valores, label=metodo)

    plt.xlabel("Número de elementos (N)")
    plt.ylabel("Número de comparações")
    plt.title("Comparativo de Número de Comparações nas
    Buscas")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

# Comentários

```
1    print("1. A busca linear e a com sentinela têm tempo
     proporcional a N.")
2    print("2. As buscas binárias são muito mais rápidas com
     crescimento logarítmico.")
3    print("3. Para vetores grandes, a ordenação para a busca
      binária compensa.")
4    print("4. A busca com sentinela é levemente mais rápida
     que a linear padrão.")
```

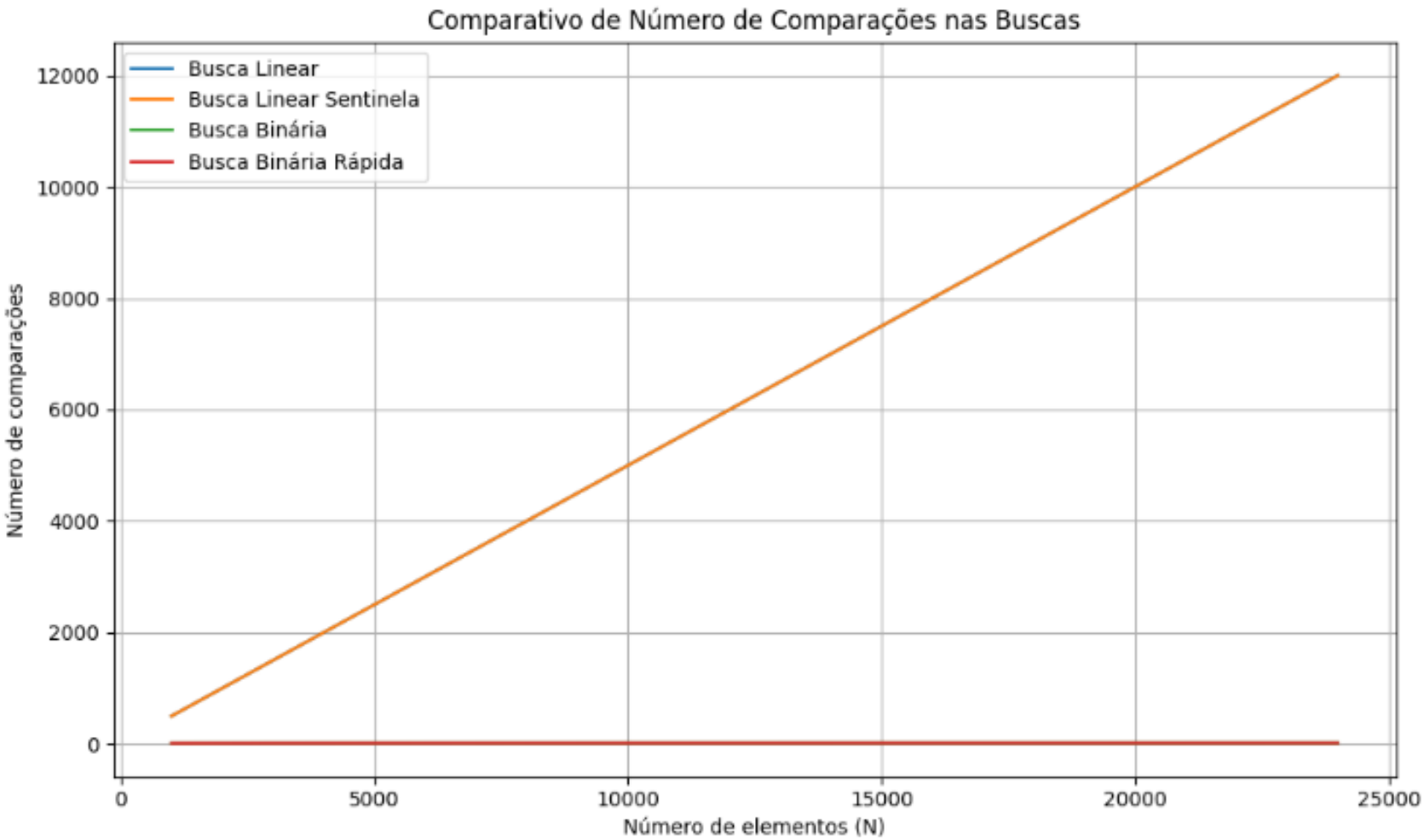
# Comparação

## Gráfico Comparativo de Tempo de Execução das Buscas



# Comparação

## Gráfico Comparativo de Número de Comparações nas Buscas



# Questão 09

**Faça um programa que cadastre 10 números, ordene-os e em seguida encontre e mostre (Linguagem: Use C ou Python ou Ruby):**

- O menor número e quantas vezes ele aparece no vetor;
- O maior número e quantas vezes ele aparece no vetor.

Obs.: Priorize métodos de ordenação com melhores desempenho.

# InsertionSort e Contar Ocorrências

```python
def insertionSort(vetor):
    for i in range(1, len(vetor)):
        chave = vetor[i]
        j = i - 1

        while j >= 0 and vetor[j] > chave:
            vetor[j + 1] = vetor[j]
            j -= 1

        vetor[j + 1] = chave

    return vetor

def contarOcorrencias(vetor, valor):
    contador = 0

    for item in vetor:
        if item == valor:
            contador += 1

    return contador
```

# Main

```python
def main():
    print("Digite 10 numeros inteiros:\n")
    numeros = []

    for i in range(10):
        num = int(input(f"{i + 1}ř Numero: "))
        numeros.append(num)

    numeros_ordenados = insertionSort(numeros.copy())

    menor = numeros_ordenados[0]
    maior = numeros_ordenados[-1]
    freq_menor = contarOcorrencias(numeros_ordenados, menor)
    freq_maior = contarOcorrencias(numeros_ordenados, maior)

    print("\nNumeros desordendos:", numeros)
    print("Numeros ordenados:", numeros_ordenados)
    print(f"\nMenor numero: {menor} (aparece {freq_menor}
    vezes)")
    print(f"Maior numero: {maior} (aparece {freq_maior}
    vezes)\n")
```

# Questão 10

Implementar o algoritmo Selection Sort para ordenar uma lista encadeada de números. Linguagens permitidas: C, C++, Java, Python e Ruby

# Selection Sort

```
class ListaEncadeada:
    def selectionSort(self):
        atual = self.cabeca

        while atual:
            menor = atual
            proximo = atual.proximo

            while proximo:
                if proximo.valor < menor.valor:
                    menor = proximo

                proximo = proximo.proximo

            atual.valor, menor.valor = menor.valor, atual.valor
            atual = atual.proximo
```

# Main

```
1  def main():
2      lista = ListaEncadeada()
3      numeros = [29, 10, 14, 37, 13]
4
5      for num in numeros:
6          lista.inserir(num)
7
8      print("Lista antes da ordenacao:")
9      lista.imprimir()
10
11     lista.selectionSort()
12
13     print("\nLista apos ordenacao com Selection Sort:")
14     lista.imprimir()
```

# Questão 11

Implemente em C/C++ os algoritmos de busca sequencial e binária para listas simplesmente encadeadas.

# Lista sequencial e binária

```c
#include <stdio.h>
#include <stdlib.h>

// Definição do nó da lista
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Função para criar um novo nó
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Erro ao alocar memória.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

# Função para inserir no final da lista

```
1  // Função para inserir no final da lista
2  void insertEnd(Node** head, int data) {
3      Node* newNode = createNode(data);
4      if (*head == NULL) {
5          *head = newNode;
6          return;
7      }
8      Node* temp = *head;
9      while (temp->next != NULL)
10         temp = temp->next;
11     temp->next = newNode;
12 }
```

# Busca Sequencial

```
1 Node* sequentialSearch(Node* head, int target) {
2     while (head != NULL) {
3         if (head->data == target)
4             return head;
5         head = head->next;
6     }
7     return NULL;
8 }
```

# Função auxiliar para retornar o ponteiro do meio entre dois nós

```
1  Node* getMiddle(Node* start, Node* end) {
2      if (start == NULL)
3          return NULL;
4      Node* slow = start;
5      Node* fast = start->next;
6
7      while (fast != end) {
8          fast = fast->next;
9          if (fast != end) {
10             slow = slow->next;
11             fast = fast->next;
12         }
13     }
14     return slow;
15 }
```

# Busca binária

```
1  Node* binarySearch(Node* head, int target) {
2      Node* start = head;
3      Node* end = NULL;
4
5      do {
6          Node* mid = getMiddle(start, end);
7          if (mid == NULL)
8              return NULL;
9
10         if (mid->data == target)
11             return mid;
12         else if (mid->data < target)
13             start = mid->next;
14         else
15             end = mid;
16     } while (end == NULL || end != start);
17
18     return NULL;
19 }
```

# Função para imprimir lista

```c
void printList(Node* head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
```

# Exemplo de uso

```
1
2  int main() {
3      Node* list = NULL;
4      // Lista ordenada
5      insertEnd(&list, 1);
6      insertEnd(&list, 3);
7      insertEnd(&list, 5);
8      insertEnd(&list, 7);
9      insertEnd(&list, 9);
10     insertEnd(&list, 11);
11
12     printf("Lista:\n");
13     printList(list);
14
15     int value = 7;
16     Node* resultSeq = sequentialSearch(list, value);
17     if (resultSeq)
18         printf("Busca Sequencial: Encontrado %d.\n",
       resultSeq->data);
19     else
20         printf("Busca Sequencial: Não encontrado.\n");
```

# Exemplo de uso

```
    Node* resultBin = binarySearch(list, value);
    if (resultBin)
        printf("Busca Binária: Encontrado %d.\n", resultBin
    ->data);
    else
        printf("Busca Binária: Não encontrado.\n");

    return 0;
}

\begin{frame}{Questões Gerais}
```

# Questão 12

Implemente a estrutura de dados Árvore Binária em Python ou Java ou C++ usando conceitos de orientação a objetos. Para esta árvore, implemente os percursos pré-ordem, em ordem e pós-ordem. Com base na árvore binária da Figura 8, imprima os percursos de cada método.

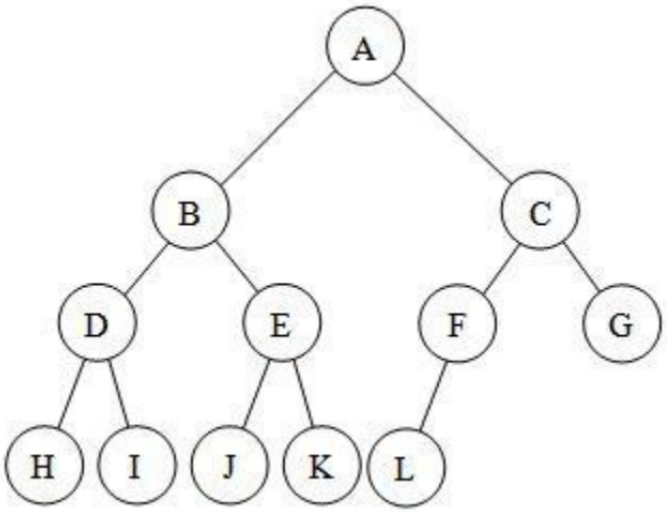



Figura: Figura 8 Árvore binária para teste

# Classe Nó

```
class No {
private:
    char valor;
    No *esquerda;
    No *direita;
public:
    No(char valor)
    {
        this->valor = valor;
        this->esquerda = nullptr;
        this->direita = nullptr;
    }

    char getValor() { return this->valor; }
    No *getEsquerda() { return this->esquerda; }
    No *getDireita() { return this->direita; }

    void setValor(char valor) { this->valor = valor; }
    void setEsquerda(No *node) { this->esquerda = node; }
    void setDireita(No *node) { this->direita = node; }
};
```

# Classe Árvore

```
class Arvore
{
private:
    No *raiz;

    No *construirArvoreArray(const vector<char> &arr, int i)
    {
        if (i >= arr.size() || arr[i] == ' ')
        {
            return nullptr;
        }

        No *no = new No(arr[i]);
        no->setEsquerda(construirArvoreArray(arr, 2 * i + 1)
    );
        no->setDireita(construirArvoreArray(arr, 2 * i + 2))
    ;
        return no;
    }
```

# Percurso Pré-Ordem

```
public:
    Arvore() : raiz(nullptr) {}

    void construirArvore(const vector<char> &arr)
    {
        raiz = construirArvoreArray(arr, 0);
    }

    No *getRaiz() { return raiz; }

    void preOrdem(No *noAtual)
    {
        if (noAtual)
        {
            cout << noAtual->getValor() << " ";
            preOrdem(noAtual->getEsquerda());
            preOrdem(noAtual->getDireita());
        }
    }
```

# Percurso Em Ordem

```
void emOrder(No *noAtual)
    {
        if (noAtual)
        {
            emOrder(noAtual->getEsquerda());
            cout << noAtual->getValor() << " ";
            emOrder(noAtual->getDireita());
        }
    }
```

# Percurso Pós-Ordem

```
1  void posOrdem(No *noAtual)
2      {
3          if (noAtual)
4          {
5              posOrdem(noAtual->getEsquerda());
6              posOrdem(noAtual->getDireita());
7              cout << noAtual->getValor() << " ";
8          }
9      }
10 };
```

# Main

```
1  int main()
2  {
3      vector<char> arr = {'A', 'B', 'C', 'D', 'E', 'F', 'G', '
       H', 'I', 'J', 'K', 'L', };
4
5      Arvore arvore;
6      arvore.construirArvore(arr);
7
8      cout << "Pre-ordem: ";
9      arvore.preOrdem(arvore.getRaiz());
10     cout << "\nEm ordem: ";
11     arvore.emOrder(arvore.getRaiz());
12     cout << "\nPos-ordem: ";
13     arvore.posOrdem(arvore.getRaiz());
14     cout << endl;
15
16     return 0;
17 }
```