

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

- Questão Geral - A
- Questão 09
- Questão 11

- Questão Geral - B
- Questão 10
- Questão 12

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

```
1 def bubble_sort(lista):
2     n = len(lista)
3     for i in range(n-1):
4         for j in range(n-1-i):
5             if lista[j] > lista[j+1]:
6                 lista[j], lista[j+1] = lista[j+1], lista[j]
7     return lista
```

Insertion Sort

```
1 def insertionSort(lista):
2     tamanho = len(lista)
3
4     for i in range(1, tamanho):
5         elemento = lista[i]
6         j = i - 1
7         while j >= 0 and elemento < lista[j]:
8             lista[j+1] = lista[j]
9             j = j - 1
10        lista[j+1] = elemento
11
12    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

```
1 def mergeSort(lista):
2
3     n = len(lista)
4     if n <= 1:
5         return lista
6
7     metade = n//2
8     listaEsquerda = mergeSort(lista[:metade])
9     listaDireita = mergeSort(lista[metade:])
10    return merge(listaEsquerda, listaDireita)
```

Merge Sort

```

1 def merge(esquerda, direita):
2     i = j = 0
3     resultado = []
4
5     while i < len(esquerda) and j < len(direita):
6         if esquerda[i] < direita[j]:
7             resultado.append(esquerda[i])
8             i = i + 1
9         else:
10            resultado.append(direita[j])
11            j = j + 1
12
13    resultado.extend(esquerda[i:])
14    resultado.extend(direita[j:])
15
16    return resultado
    
```


Quick Sort

```
1 def quickSort(lista):
2     quickSortOrdenar(lista, 0, len(lista) - 1)
3     return lista
4
5 def quickSortOrdenar(lista, inicio, fim):
6
7     if inicio >= fim:
8         return
9
10    pivot = particao(lista, inicio, fim)
11    quickSortOrdenar(lista, inicio, pivot - 1)
12    quickSortOrdenar(lista, pivot + 1, fim)
```

Quick Sort

```
1 def particao(lista, inicio, fim):
2
3     pivot = lista[inicio]
4     i = inicio + 1
5
6     for j in range(inicio + 1, fim + 1):
7         if lista[j] < pivot:
8             lista[i], lista[j] = lista[j], lista[i]
9             i = i + 1
10
11     lista[inicio], lista[i - 1] = lista[i - 1], lista[inicio]
12     return i - 1
```

Counting Sort

```
1 def countingSort(lista):
2
3     minimo = min(lista)
4     maximo = max(lista)
5     tamanho = maximo - minimo + 1
6     arrayCopia = [0 for i in range(len(lista))]
7     arrayContagem = [0 for i in range(tamanho)]
8
9     for i in range(0, len(lista)):
10         arrayContagem[lista[i] - minimo] += 1
11
12     for i in range(1, len(arrayContagem)):
13         arrayContagem[i] += arrayContagem[i - 1]
14
15     for i in range(len(lista) - 1, -1, -1):
16         valor = lista[i]
17         arrayContagem[valor - minimo] -= 1
18         posicao = arrayContagem[valor - minimo]
19         arrayCopia[posicao] = valor
20
21     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(
13                 negativos)]
14             if nao_negativos:
15                 maximo = max(nao_negativos)
16                 exp = 1
17                 while maximo // exp > 0:
18                     nao_negativos = contagem(nao_negativos, exp)
19                     exp *= 10
20
21     return negativos_ordenados + nao_negativos
  
```

Radix Sort

```
1 def contagem(lista, exp):
2
3     arrayContagem = [0 for i in range(10)]
4     resultado = [0 for i in range(len(lista))]
5
6     for i in range(0, len(lista)):
7         digito = (lista[i] // exp) % 10
8         arrayContagem[digito] += 1
9
10    for i in range(1, 10):
11        arrayContagem[i] += arrayContagem[i - 1]
12
13    for i in range(len(lista) - 1, -1, -1):
14        digito = (lista[i] // exp) % 10
15        arrayContagem[digito] -= 1
16        resultado[arrayContagem[digito]] = lista[i]
17
18    return resultado
```

Bucket Sort

```

1  def bucketSort(lista):
2
3      maximo = max(lista)
4      minimo = min(lista)
5      numBaldes = int(math.sqrt(len(lista))) or 1
6      intervalo = (maximo - minimo + 1) / numBaldes
7      baldes = [[] for _ in range(numBaldes)]
8
9      for valor in lista:
10         index = int((valor - minimo) / intervalo)
11         if index == numBaldes:
12             index -= 1
13         baldes[index].append(valor)
14
15     resultado = []
16     for balde in baldes:
17         balde = insertionSort(balde)
18         resultado.extend(balde)
19
20     return resultado
  
```

Shell Sort

```

1  def shellSort(lista):
2
3      n = len(lista)
4      h = len(lista) // 2
5
6      while h > 0:
7          for i in range(h, n):
8              valor = lista[i]
9              j = i
10             while j >= h and valor < lista[j - h]:
11                 lista[j] = lista[j - h]
12                 j = j - h
13             lista[j] = valor
14             h = int(h / 2)
15
16     return lista
  
```

Heap Sort

```

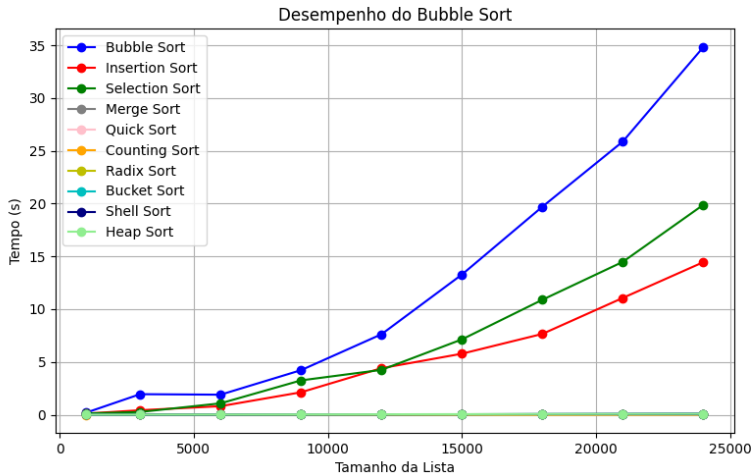
1 def heapify(lista, n, i):
2
3     maior = i
4     left = 2 * i + 1
5     right = 2 * i + 2
6
7     if left < n and lista[left] > lista[maior]:
8         maior = left
9
10    if right < n and lista[right] > lista[maior]:
11        maior = right
12
13    if maior != i:
14        lista[maior], lista[i] = lista[i], lista[maior]
15        heapify(lista, n, maior)
  
```


Heap Sort

```
1 def heapSort(lista):
2
3     n = len(lista)
4
5     for i in range(n//2 - 1, -1, -1):
6         heapify(lista, n, i)
7
8     for i in range(n - 1, 0, -1):
9         lista[0], lista[i] = lista[i], lista[0]
10        heapify(lista, i, 0)
11
12    return lista
```

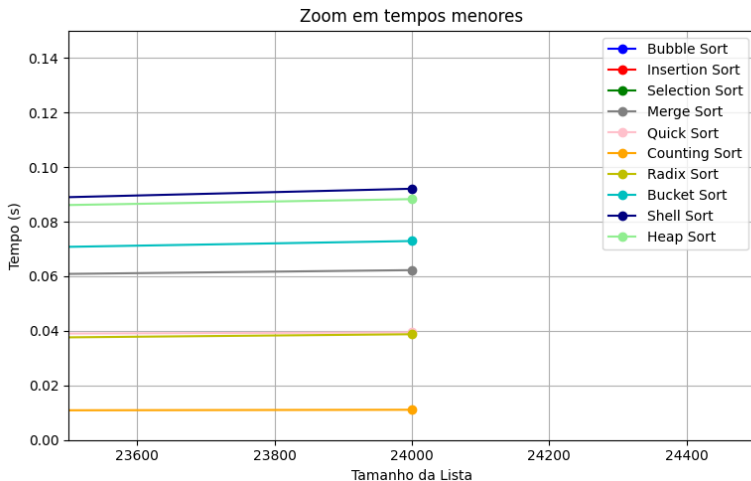
Comparação

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

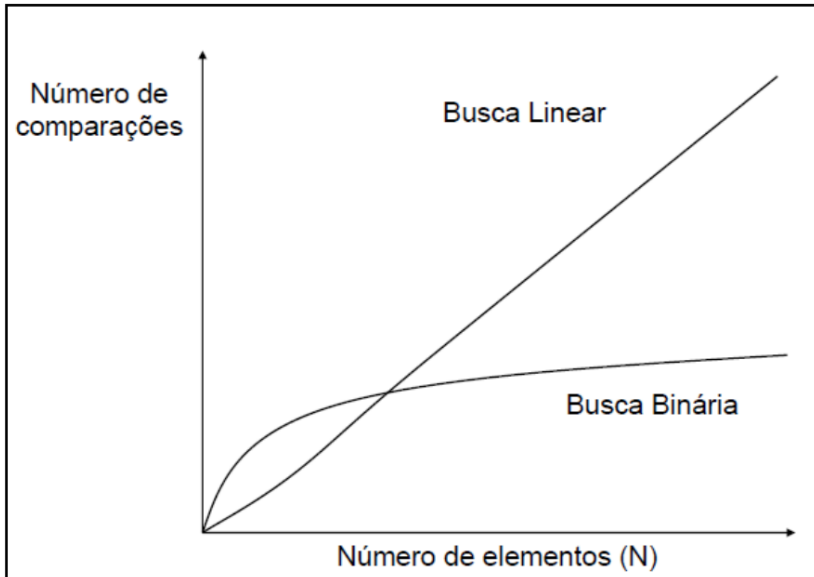
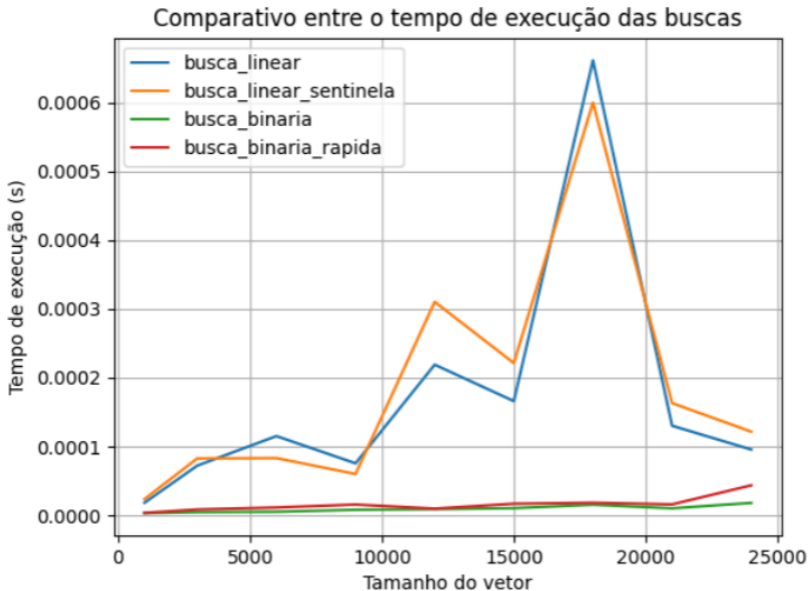


Figura 4 - Exemplo do gráfico de comparação dos métodos de busca



Bibliotecas

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

Insertion Sort Algoritmo de Ordenação

```
1 def insertion_sort(vetor):
2     for i in range(1, len(vetor)):
3         chave = vetor[i]
4         j = i - 1
5         while j >= 0 and vetor[j] > chave:
6             vetor[j + 1] = vetor[j]
7             j -= 1
8         vetor[j + 1] = chave
```


Busca Linear

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

Busca Linear Sentinela

```
1 def busca_linear_sentinela(vetor, chave):
2     vetor.append(chave)
3     i = 0
4     while vetor[i] != chave:
5         i += 1
6     vetor.pop()
7     return i if i < len(vetor) else -1
```

Busca Binária

```
1 def busca_binaria(vetor, chave):
2     esquerda, direita = 0, len(vetor) - 1
3     while esquerda <= direita:
4         meio = (esquerda + direita) // 2
5         if vetor[meio] == chave:
6             return meio
7         elif vetor[meio] < chave:
8             esquerda = meio + 1
9         else:
10            direita = meio - 1
11    return -1
```

Busca Binária Rápida

```
1 def busca_binaria_rapida(vetor, chave):
2     esquerda, direita = 0, len(vetor) - 1
3     while esquerda <= direita:
4         meio = (esquerda + direita) // 2
5         if vetor[meio] < chave:
6             esquerda = meio + 1
7         else:
8             direita = meio - 1
9     return esquerda if esquerda < len(vetor) and vetor[
        esquerda] == chave else -1
```

Parâmetros

```
1 tamanhos = [1000, 3000, 6000, 9000, 12000, 15000, 18000,
2           21000, 24000]
3 tempos = {
4     "busca_linear": [],
5     "busca_linear_sentinela": [],
6     "busca_binaria": [],
7     "busca_binaria_rapida": []
8 }
```

Entrada da chave

```
1 entrada = input("Digite o valor da chave a ser buscada (ou  
    pressione Enter para gerar aleatoriamente): ")  
2 chave_usuario = int(entrada) if entrada.strip().isdigit()  
    else None  
3 }
```

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

```
1      inicio = time.perf_counter()
2      busca_linear_sentinela(vetor[:], chave)
3      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

```

1 plt.figure(figsize=(10, 6))
2 for metodo, valores in tempos.items():
3     plt.plot(tamanhos, valores, marker='o', label=metodo
4             .replace("_", " ").capitalize())
5
6 plt.xlabel("Tamanho do vetor")
7 plt.ylabel("Tempo de execução (s)")
8 plt.title("Comparativo de Tempo de Execução das Buscas")
9 plt.legend()
10 plt.grid(True)
11 plt.tight_layout()
12 plt.show()

```

Gráfico do número de comparações

```

1  comparacoes = {
2      "Busca Linear": [n / 2 for n in tamanhos],
3      "Busca Linear Sentinela": [n / 2 for n in tamanhos],
4      "Busca Binária": [math.log2(n) for n in tamanhos],
5      "Busca Binária Rápida": [math.log2(n) for n in
6      tamanhos]
7  }
8
9  plt.figure(figsize=(10, 6))
10 for metodo, valores in comparacoes.items():
11     plt.plot(tamanhos, valores, label=metodo)
12
13 plt.xlabel("Número de elementos (N)")
14 plt.ylabel("Número de comparações")
15 plt.title("Comparativo de Número de Comparações nas
16 Buscas")
17 plt.legend()
18 plt.grid(True)
19 plt.tight_layout()
20 plt.show()

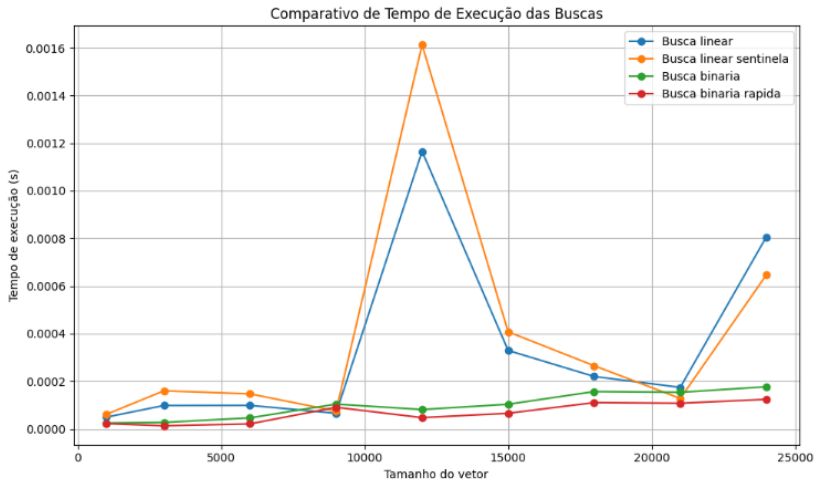
```

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.")
```

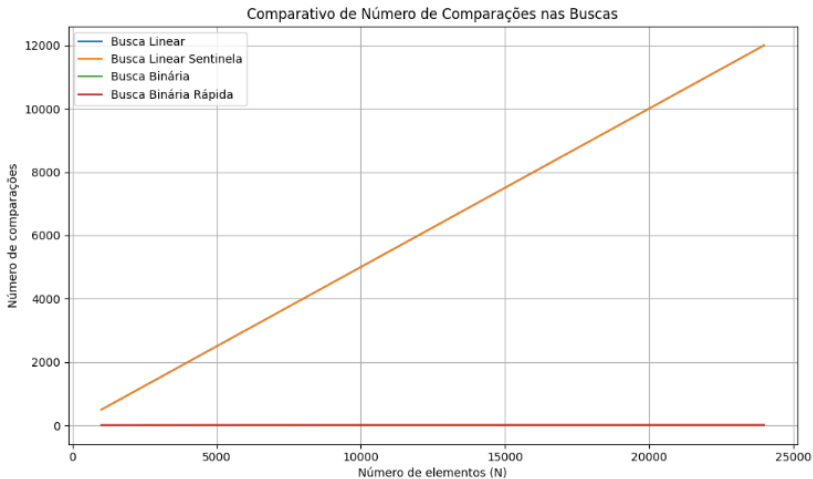
Comparação

Gráfico Comparativo de Tempo de Execução das Buscas



Comparação

Gráfico Comparativo de Número de Comparações nas Buscas



InsertionSort e Contar Ocorrências

```
1 def insertionSort(vetor):
2     for i in range(1, len(vetor)):
3         chave = vetor[i]
4         j = i - 1
5
6         while j >= 0 and vetor[j] > chave:
7             vetor[j + 1] = vetor[j]
8             j -= 1
9
10        vetor[j + 1] = chave
11
12    return vetor
13
14 def contarOcorrencias(vetor, valor):
15     contador = 0
16
17     for item in vetor:
18         if item == valor:
19             contador += 1
20
21    return contador
```

Main

```
1 def main():
2     print("Digite 10 numeros inteiros:\n")
3     numeros = []
4
5     for i in range(10):
6         num = int(input(f"{i + 1}ª Numero: "))
7         numeros.append(num)
8
9     numeros_ordenados = insertionSort(numeros.copy())
10
11     menor = numeros_ordenados[0]
12     maior = numeros_ordenados[-1]
13     freq_menor = contarOcorrencias(numeros_ordenados, menor)
14     freq_maior = contarOcorrencias(numeros_ordenados, maior)
15
16     print("\nNumeros desordendos:", numeros)
17     print("Numeros ordenados:", numeros_ordenados)
18     print(f"\nMenor numero: {menor} (aparece {freq_menor}
19     vezes)")
20     print(f"Maior numero: {maior} (aparece {freq_maior}
21     vezes)\n")
```

SelectionSort

```

1 class ListaEncadeada:
2     def selectionSort(self):
3         atual = self.cabeca
4
5         while atual:
6             menor = atual
7             proximo = atual.proximo
8
9             while proximo:
10                 if proximo.valor < menor.valor:
11                     menor = proximo
12
13                 proximo = proximo.proximo
14
15             atual.valor, menor.valor = menor.valor, atual.
16             valor
17             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()

```

Lista sequencial e binária

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Definição do nó da lista
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Função para criar um novo nó
11 Node* createNode(int data) {
12     Node* newNode = (Node*)malloc(sizeof(Node));
13     if (!newNode) {
14         printf("Erro ao alocar memória.\n");
15         exit(1);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }

```

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

```

1 void printList(Node* head) {
2     while (head != NULL) {
3         printf("%d -> ", head->data);
4         head = head->next;
5     }
6     printf("NULL\n");
7 }

```

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",
19              resultSeq->data);
20    else
21        printf("Busca Sequencial: Não encontrado.\n");
```

Exemplo de uso

```

1  Node* resultBin = binarySearch(list, value);
2  if (resultBin)
3      printf("Busca Binária: Encontrado %d.\n", resultBin
->data);
4  else
5      printf("Busca Binária: Não encontrado.\n");
6
7  return 0;
8  }
9
10 \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.

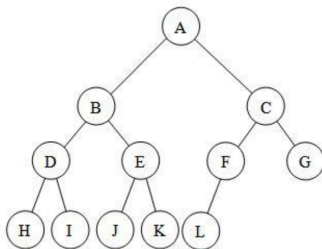


Figura: Figura 8 Árvore binária para teste

Classe Nó

```

1  class No {
2  private:
3      char valor;
4      No *esquerda;
5      No *direita;
6  public:
7      No(char valor)
8      {
9          this->valor = valor;
10         this->esquerda = nullptr;
11         this->direita = nullptr;
12     }
13
14     char getValor() { return this->valor; }
15     No *getEsquerda() { return this->esquerda; }
16     No *getDireita() { return this->direita; }
17
18     void setValor(char valor) { this->valor = valor; }
19     void setEsquerda(No *node) { this->esquerda = node; }
20     void setDireita(No *node) { this->direita = node; }
21 };
    
```

Classe Árvore

```

1  class Arvore
2  {
3  private:
4      No *raiz;
5
6      No *construirArvoreArray(const vector<char> &arr, int i)
7      {
8          if (i >= arr.size() || arr[i] == ' ')
9          {
10             return nullptr;
11         }
12
13         No *no = new No(arr[i]);
14         no->setEsquerda(construirArvoreArray(arr, 2 * i + 1)
15         );
16         no->setDireita(construirArvoreArray(arr, 2 * i + 2))
17         ;
18         return no;
19     }

```


Percurso Pré-Ordem

```

1 public:
2     Arvore() : raiz(nullptr) {}
3
4     void construirArvore(const vector<char> &arr)
5     {
6         raiz = construirArvoreArray(arr, 0);
7     }
8
9     No *getRaiz() { return raiz; }
10
11    void preOrdem(No *noAtual)
12    {
13        if (noAtual)
14        {
15            cout << noAtual->getValor() << " ";
16            preOrdem(noAtual->getEsquerda());
17            preOrdem(noAtual->getDireita());
18        }
19    }

```

Percurso Em Ordem

```

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

```

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 }
```