

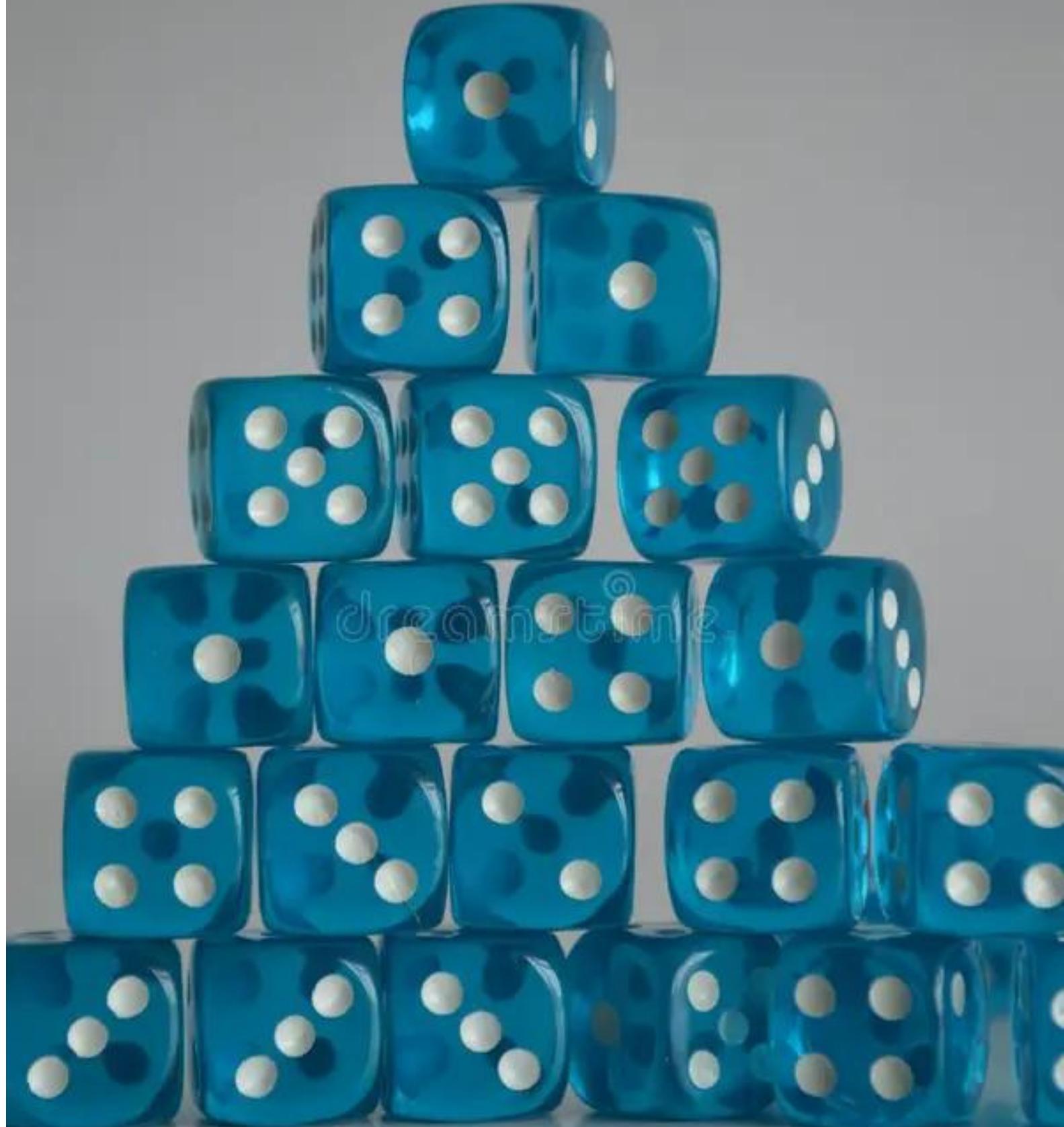
Estruturas de Dados

Explorando os Fundamentos

Por Rayane Paiva e Daniel Rodrigues

O que são dados ?

Dados são informações não tratadas. Dados surgem a partir de investigações ou pesquisas, porém seriam o que chamamos de conhecimento bruto, pois não foram tratados de forma que gerem alguma informação útil.



Tipo Abstrato de Dados (TADs)

TADs SE CONCENTRAM NA IDEIA DE ENCAPSULAR DADOS E OPERAÇÕES RELACIONADAS EM UMA ÚNICA UNIDADE

A ideia por trás do **TAD** é fornecer uma interface clara e bem definida para o uso de uma **estrutura de dados** ou objeto, ocultando os detalhes internos de como essa estrutura é representada e implementada. Isso ajuda a promover a modularidade e a abstração no design de programas, permitindo que os desenvolvedores se concentrem em como usar um tipo de dado em vez de se preocupar com sua implementação.

Biblioteca como um TAD:



Interface Abstrata

Ao entrar em uma biblioteca encontramos uma interface que inclui prateleiras organizadas, listas de categorias, catálogos de buscas e atendentes.

Abstração de Complexidade

A biblioteca cuida, internamente, do sistema de catalogação, registro de empréstimos, controle de datas, etc.

Operações Definidas

As operações disponíveis na interface permitem buscar livros, por título, autor ou categoria, fazer empréstimos e devolvê-los.

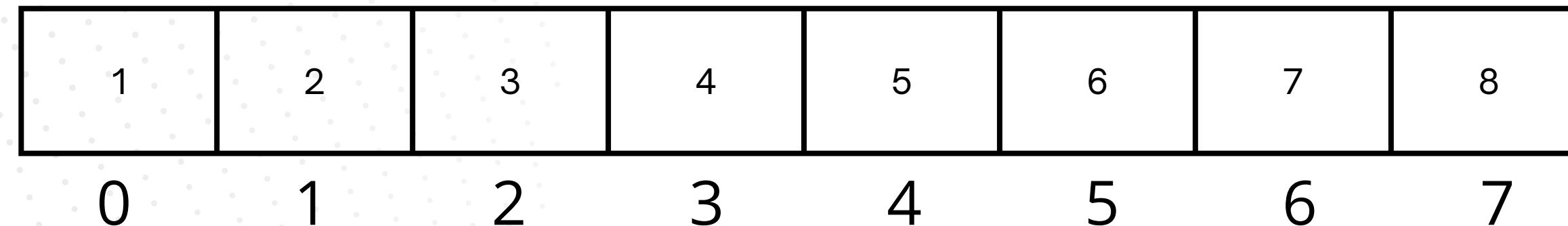
Reutilização Simples

Qualquer pessoa pode usar a biblioteca desde que conheça as operações definidas na interface.

TAD Estático

Um TAD estático tem o tamanho fixo, ou seja, é necessário definir o tamanho máximo dos dados que o TAD pode armazenar antes de usar a estrutura (arrays e matrizes em C/C++).

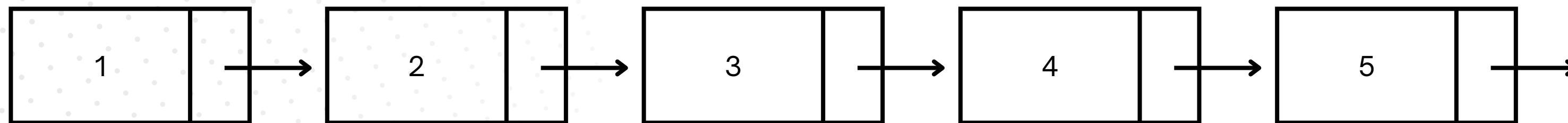
A principal desvantagem é a inflexibilidade no tamanho dos dados. Se for necessário mais espaço do que o tamanho especificado, um novo TAD terá que ser declarado com um tamanho maior.



TAD Dinâmico

Um TAD dinâmico tem tamanho variável, ou seja, a estrutura pode crescer ou diminuir conforme necessário (pilhas, filas e árvores binárias).

TADs dinâmicos são mais flexíveis e podem acomodar dados de tamanho variável, o que os torna adequados para uma ampla gama de problemas.



O que são estruturas de dados ?

“Uma estrutura de dados é um meio para organizar e armazenar dados de modo que possam ser acessados e modificados eficientemente. Mais precisamente, uma estrutura de dados é uma coleção de valores, as operações sobre esses valores e as regras que governam essas operações. Essas operações incluem a inserção, exclusão e consulta de elementos da estrutura de dados.” - T. H. Cormen



Por que são importantes ?



Eficiência Computacional

Escolher a estrutura de dados certa pode tornar um algoritmo mais eficiente

Organização de Dados

As estruturas de dados fornecem uma maneira organizada e lógica de armazenar dados

Solução de Problemas Complexos

Elas permitem que os programadores dividam problemas em partes menores e gerenciem os dados de maneira eficaz

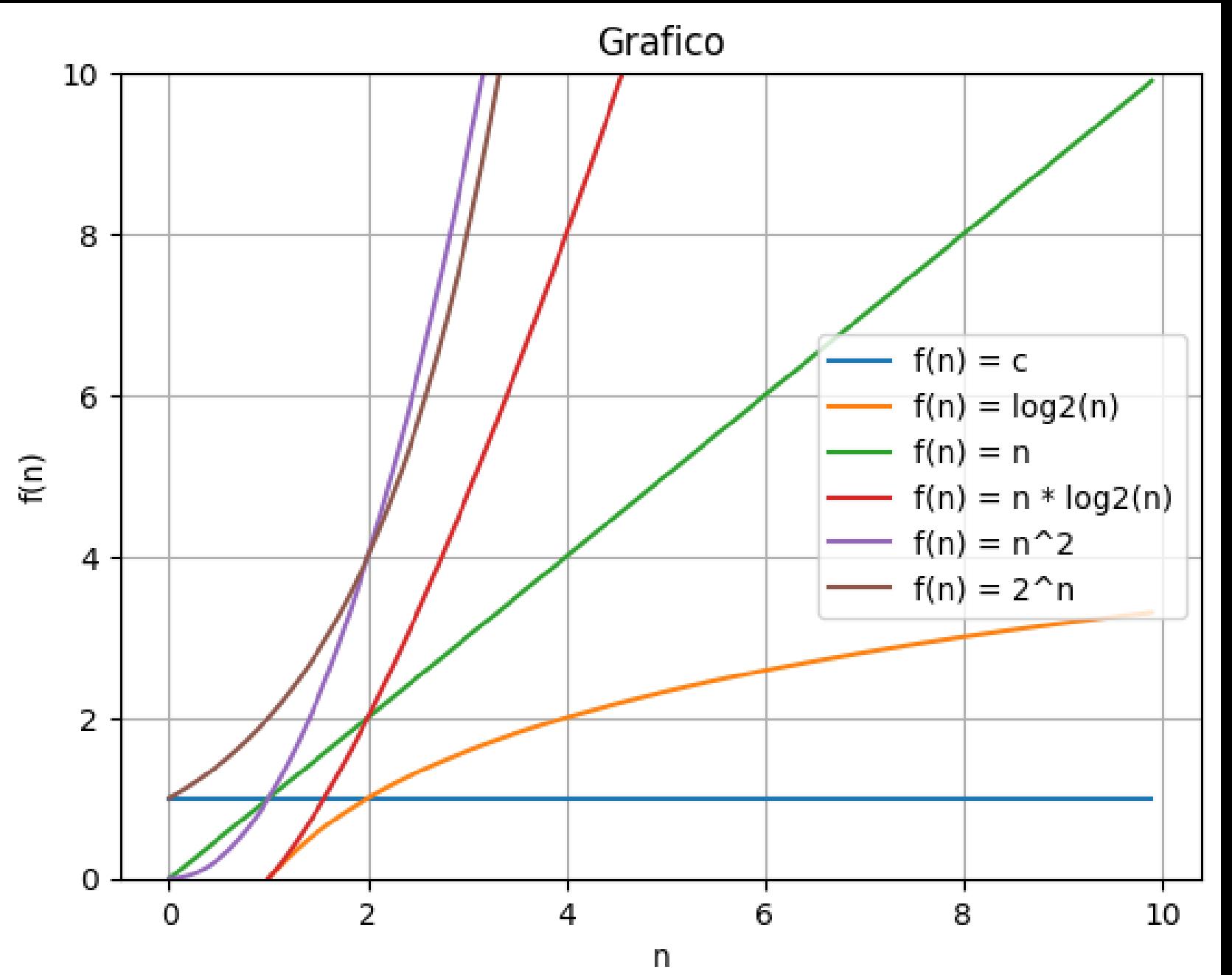
Aplicações em Diversas Áreas

São amplamente aplicáveis em muitas áreas da computação, como algoritmos, sistemas de banco de dados, design de compiladores, inteligência artificial, jogos, e muito mais.

Big O

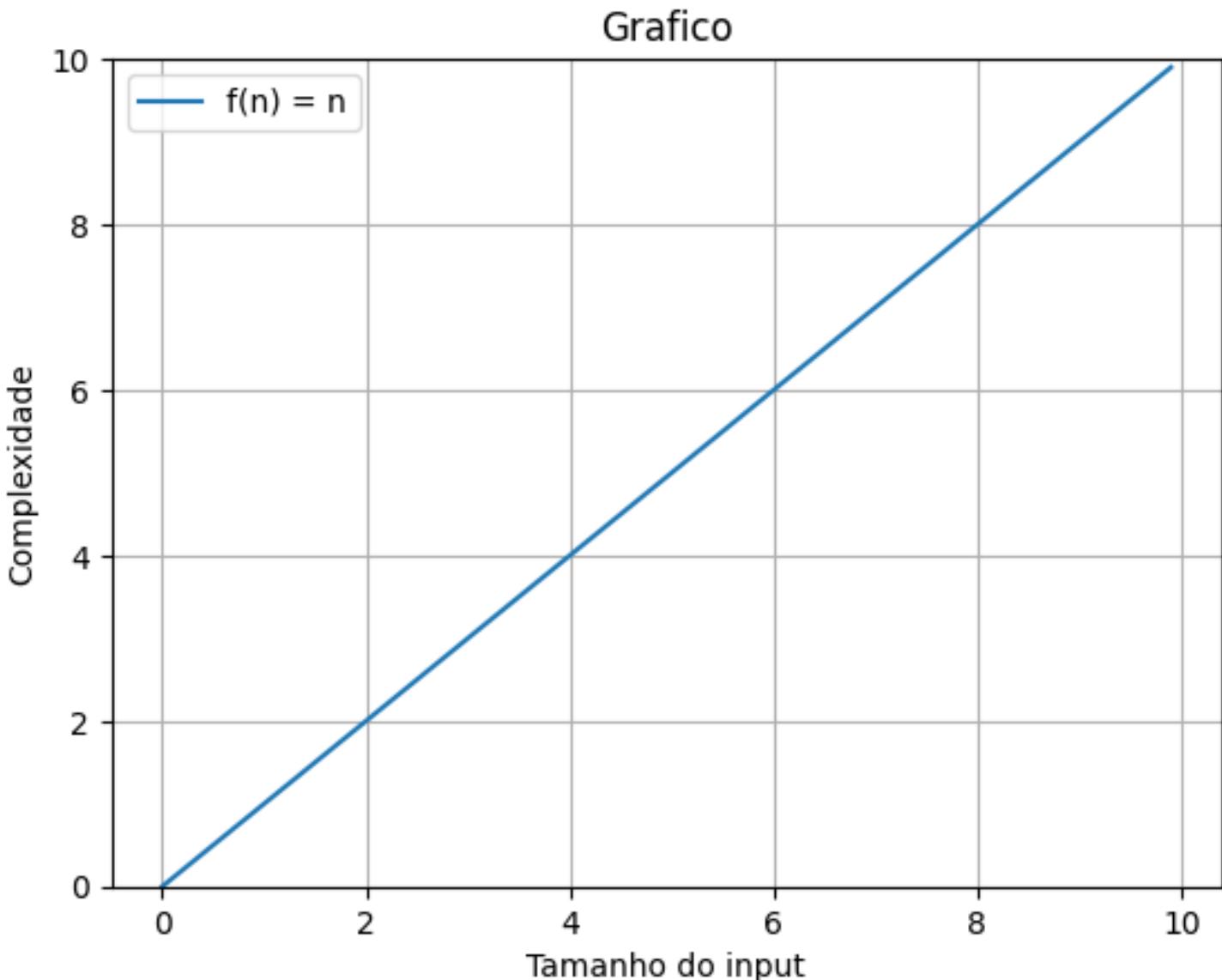
O PIOR CASO

A notação Big O é usada para descrever o limite superior do tempo de execução de um algoritmo em relação ao tamanho da entrada.



O pior caso na Busca Sequencial

```
● ● ●  
int busca(int vetor[], int n, int elemento) {  
    for (int i = 0; i < n; i++) {  
        if (vetor[i] == elemento) {  
            return i;  
        }  
    }  
    return -1;  
}
```

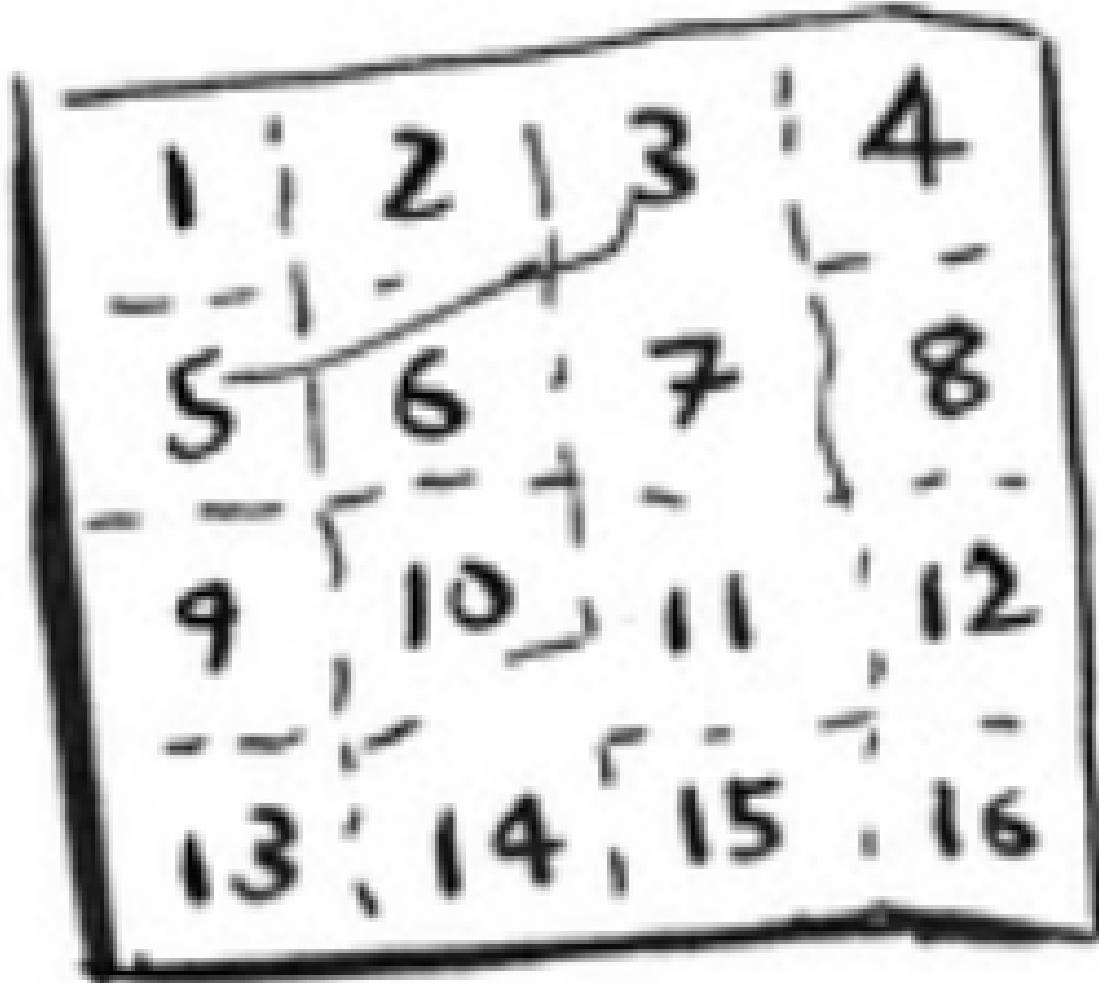


Melhor: $T(n) = 1$

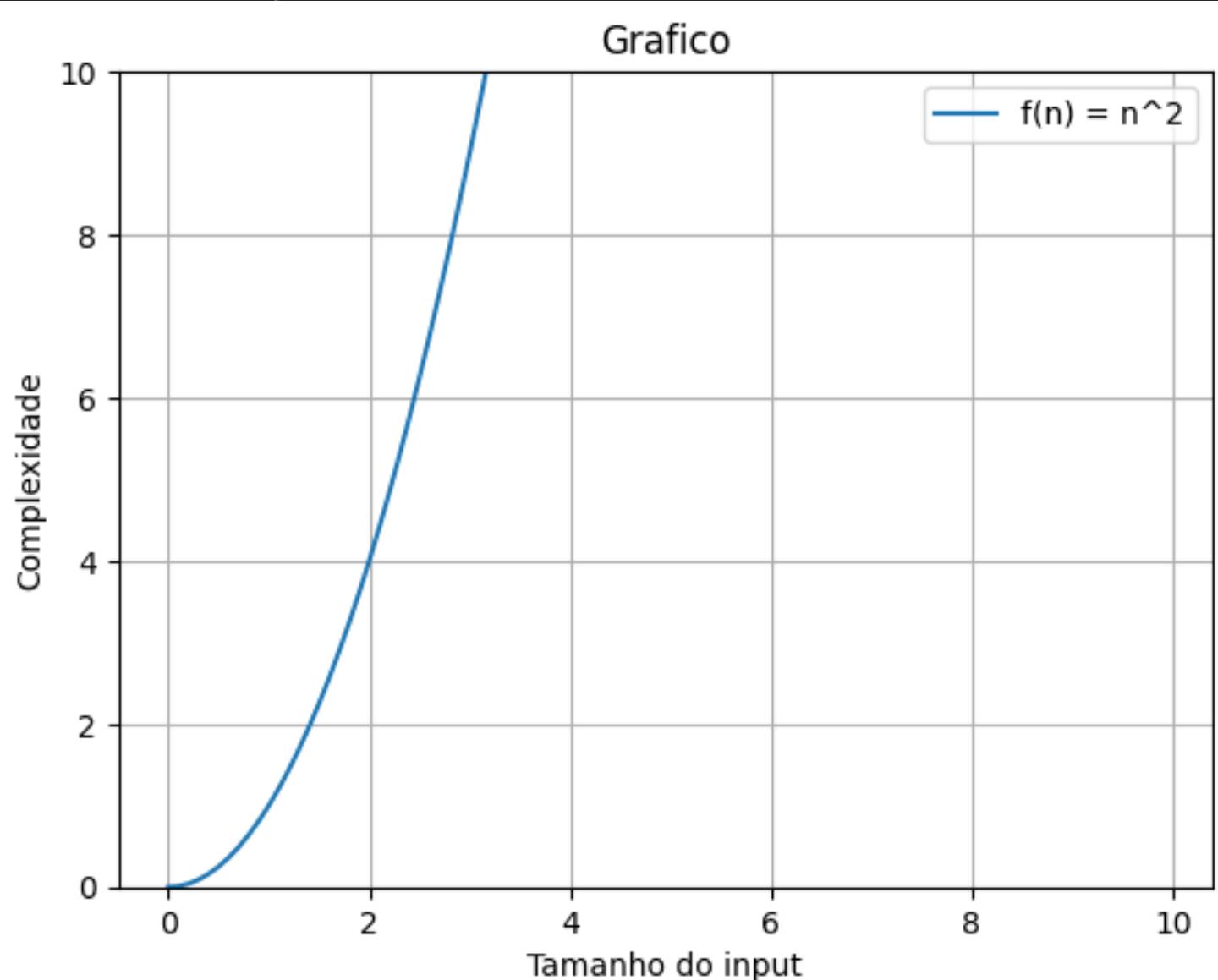
Medio: $T(n) = n/2$

Pior: $T(n) = n$

Busca Sequencial $O(n^2)$



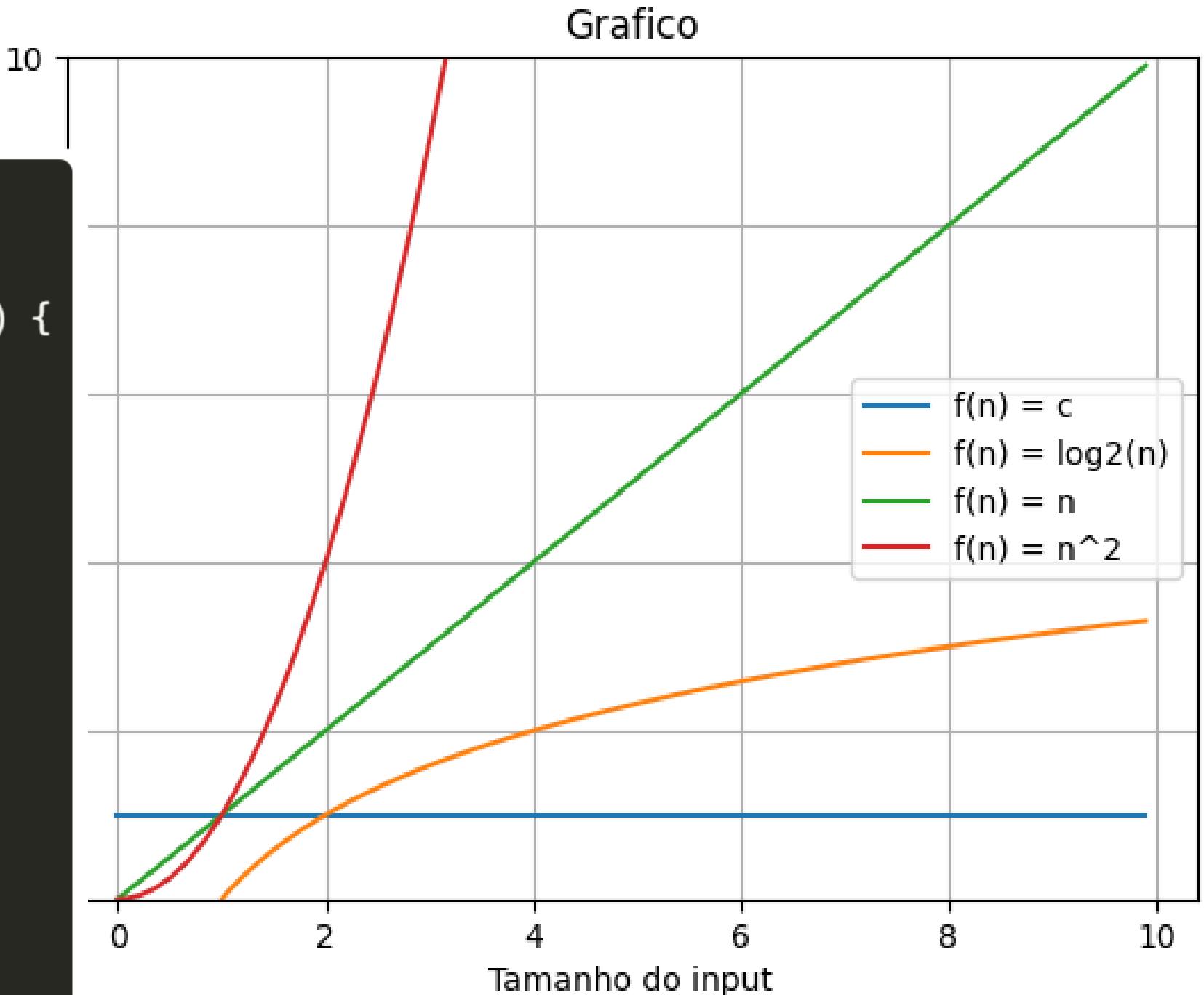
```
int busca(int matriz[][], int n, int elemento) {  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < n; j++) {  
            if (matriz[i][j] == elemento)  
                return matriz[i][j];  
        }  
    }  
    return -1;  
}
```



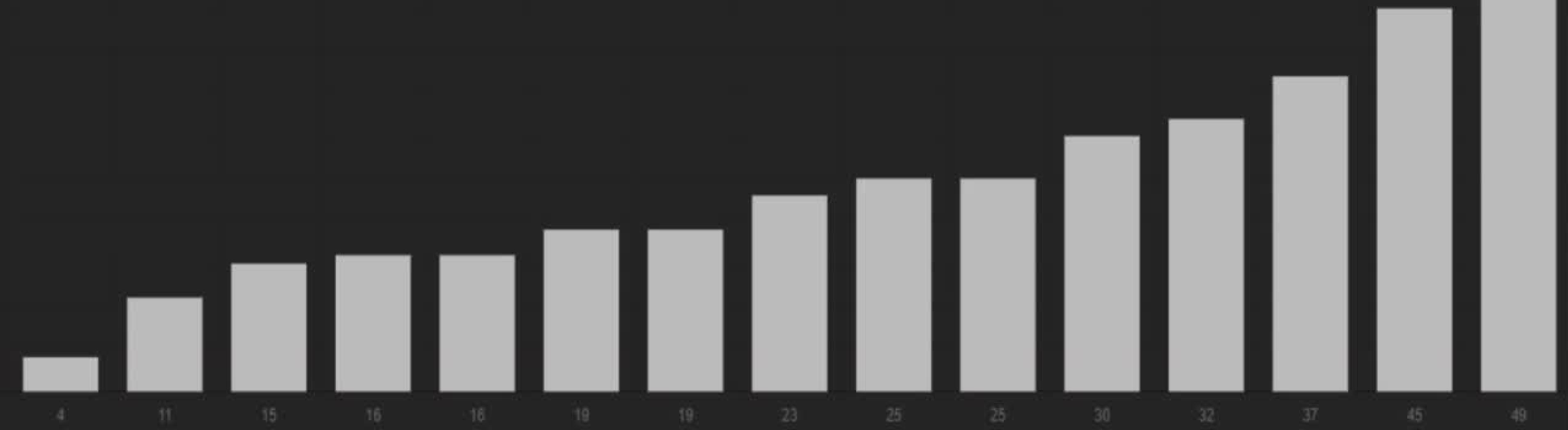
Busca O(log n)



```
int buscaBinaria(int array[], int inicio, int fim, int elemento) {  
    while (inicio <= fim) {  
        int meio = inicio + (fim - inicio) / 2;  
        if (array[meio] == elemento)  
            return meio;  
        else if (array[meio] > elemento)  
            fim = meio - 1;  
        else  
            inicio = meio + 1;  
    }  
    return -1;  
}
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	4	6	7	10	14	14	15	17	22	28	42	46	46	48



Pilha (Stack)

LAST IN, FIRST OUT - LIFO

É uma estrutura de dados linear onde o primeiro elemento a ser inserido (empilhar) é o ultimo a ser removido (desempilhar).



Características

As **operações fundamentais** em uma **pilha** são: **push** para adicionar elementos ao topo, **pop** para remover o elemento do topo, **peek** para consultar o elemento do topo e **isEmpty** para verificar se a pilha está vazia.

Estrutura simples

É simples e eficiente para manter controle de informações temporárias como rastreamento de histórico de ações em uma calculadora.

Limitação de acesso

Só é possível acessar o elemento mais recente (do topo) da pilha

Eficiencia

As operações básicas de uma pilha geralmente tem tempo de execução constante.



Casos de Uso

Histórico do Terminal

Conforme vamos executando scripts ou comandos no terminal, o mesmo vai empilhando para exibição os últimos comandos executados

Histórico de Navegação

Ao voltarmos para as telas anteriores, o navegador vai desempilhando os links e telas acessados anteriormente

Gerenciamento de chamadas de funções:

Cada vez que uma função é chamada, as informações relevantes são empilhadas, e quando a função retorna, essas informações são desempilhadas.

Fila (Queue)

FIRST IN, FIRST OUT - FIFO

Estrutura de dados linear onde o primeiro elemento a ser inserido (enfileirar) é o primeiro a ser removido (desenfileirar).



Características

As **operações fundamentais** em uma **fila** são: **enqueue** para adicionar elementos no final da fila, **dequeue** para remover o elemento inicio da fila e **isEmpty** para verificar se a fila está vazia.

Estrutura simples

É simples e eficiente para manter controle de informações temporárias como rastreamento de histórico de ações em uma calculadora.

Limitação de acesso

Só é possível acessar o elemento mais recente (do topo) da pilha

Eficiencia

As operações básicas de uma pilha geralmente tem tempo de execução constante.

>

<

<

Casos de Uso

Requisições em um servidor

Quando um servidor recebe mais requisições do que aguenta, ele enfileira elas e vai executando-as por ordem de chegada

Exibição de logs nos pods da aplicação

Conforme um sistema executa suas funções, ele vai exibindo alguns logs ordenados pelas tarefas que ele executa

Impressão de arquivos

Ao pedirmos para nossa impressora imprimir um livro para nós, ela vai imprimindo as folhas na ordem que foi post

Árvore (Tree)

ÁRVORE BINÁRIA DE BUSCA - BST

Estrutura de dados não linear onde cada elemento é armazenado em um nó, que armazena também os endereços para outros nós.



Características

As **operações fundamentais** em uma **BST** são: **insertion** para adicionar elementos na árvore, **deletion** para remover elementos, **search** para buscar um elemento e **traversals** para percorrer os nós da árvore.

Estrutura complexa

É simples de entender, mas complexa no que diz respeito a implementação eficiente das operações, especialmente quando se trata de manter a árvore

Balanceamento

Manter uma árvore binária de busca balanceada pode ser desafiador, especialmente quando as operações de inserção e exclusão são realizadas sem cuidado

Eficiencia

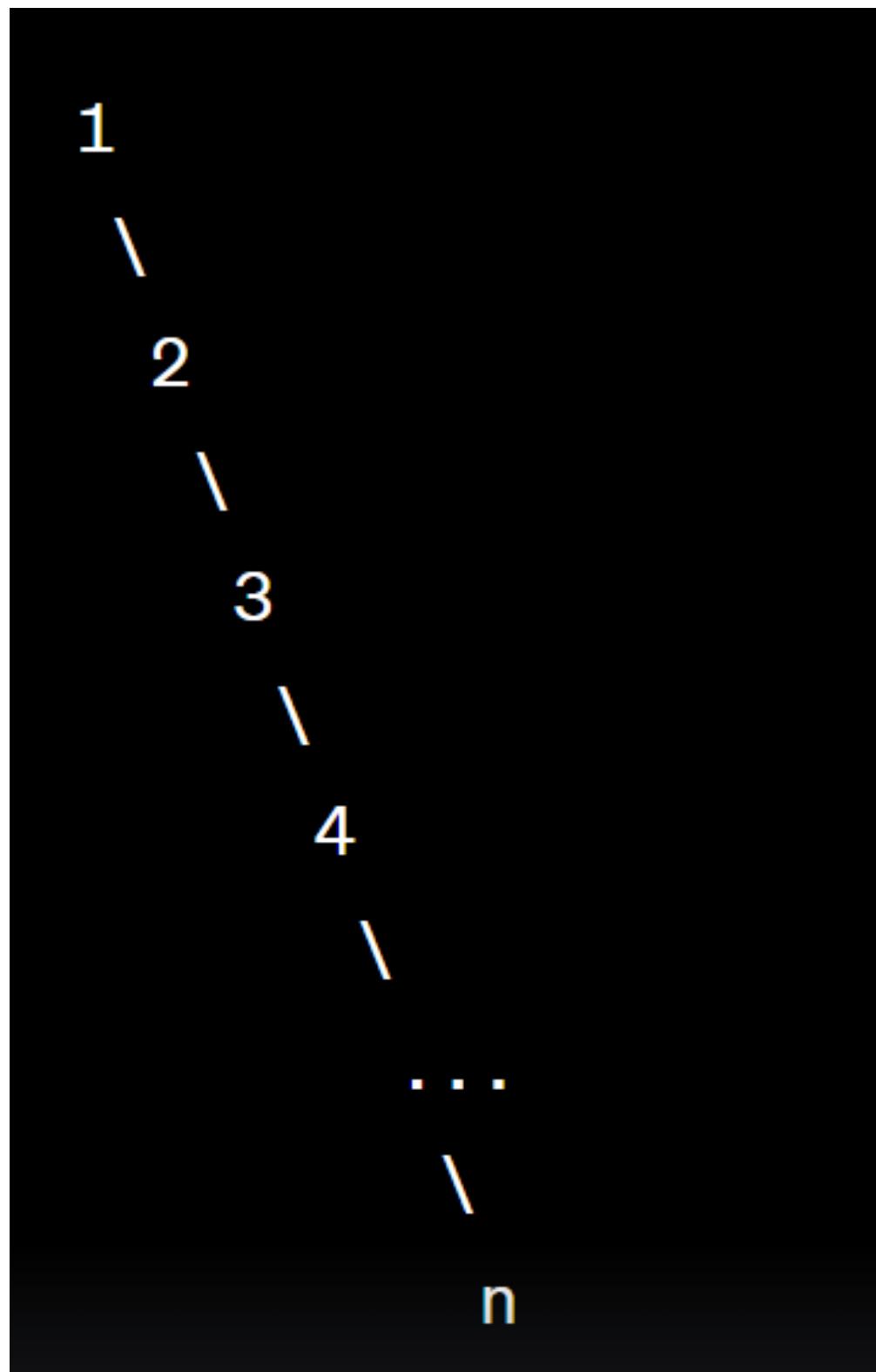
As operações básicas em uma árvore geralmente tem complexidade $O(\log n)$ no pior caso.



Ordem de Inserção

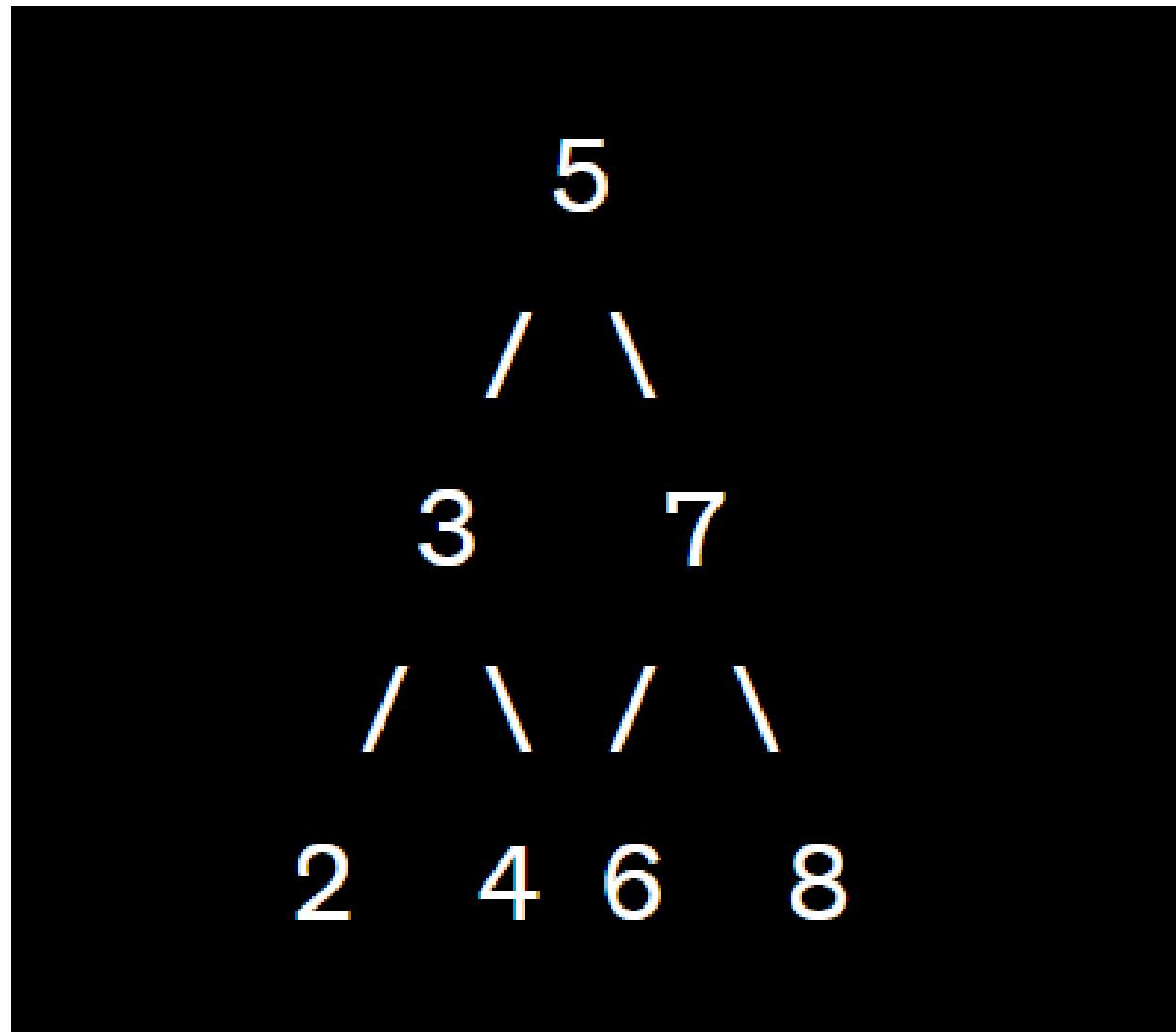
Se os elementos são inseridos em ordem ordenada, como crescente ou decrescente, a árvore resultante pode degenerar em uma lista linear.

E se a árvore é desbalanceada devido à ordem de inserção, as operações se tornam lineares, resultando em um desempenho muito inferior.



Ordem de Inserção

Essa ordenação facilita operações como busca, inserção e exclusão, proporcionando uma eficiência média de $O(\log n)$.



A wide-angle photograph of a dense forest. The foreground is filled with the dark green, conical shapes of evergreen trees. As the eye moves towards the background, the trees fade into a thick, white mist or fog that hangs low over the forest floor. The scene is bathed in a soft, dappled light that filters through the canopy, creating a peaceful and somewhat mysterious atmosphere.

O Uso de Árvores

Tabelas Hash

CHAVE E VALOR

Uma tabela hash é uma estrutura que mapeia chaves para valores aplicando uma função hash a chave para determinar o “slot” onde o dado será armazenado.



Características

As **operações fundamentais** em uma **hash** são: **insert** para adicionar elementos na tabela, **delete** para remover os elementos e **search** localizar o valor associado a uma chave específica.

Complexidade

A ideia por trás da tabela hash é relativamente simples. No entanto, a implementação e o gerenciamento de colisões podem adicionar complexidade à estrutura.

Função Hash

A eficácia de uma tabela hash em grande parte depende da qualidade da função hash utilizada.

Eficiência

Tabelas hash oferecem acesso eficiente (em média, tempo constante) aos valores com base em suas chaves.



Colisões



```
int hash(int elemento, int n){  
    return elemento % n;  
}
```

Chaves para inserir: 12, 22, 32, 7, 17

- $\text{hash}(12) = 12 \% 10 = 2$
- $\text{hash}(22) = 22 \% 10 = 2$
- $\text{hash}(32) = 32 \% 10 = 2$
- $\text{hash}(7) = 7 \% 10 = 7$
- $\text{hash}(17) = 17 \% 10 = 7$

Tabela Hash:

0:

1:

2: -> [12] -> [22] -> [32]

3:

4:

5:

6:

7: -> [7] -> [17]

8:

9:

O Uso de Tabelas Hash



Alguma pergunta ?