Recursão e busca

Rafael Alves da Costa

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
FATEC Carapicuíba

Aula 6 - Estrutura de dados

09/2025

Sumário

Recursão

Recursão

Recursão é um método para resolver problemas que envolve quebrar o problema em subproblemas cada vez menores até atingir um problema simples o bastante, que possa ser resolvido trivialmente. Em geral a recursão envolve uma função que chama ela mesma. Embora possa parecer pouco na superfície, a recursão nos permite escrever soluções elegantes para problemas que podem ser, de outra forma, muito difíceis de programar[1].

[1] Miller, B. N.; Ranum, D. L. Problem solving with algorithms and data structures using python, 2Ed. Franklin, Beedle and Associates Inc., 2011.

As Três Leis da Recursão

Como os **robôs** de **Asimov**, todos os algoritmos recursivos devem obedecer três leis importantes[1]:

(1) Um algoritmo recursivo deve possuir um caso base (base case)

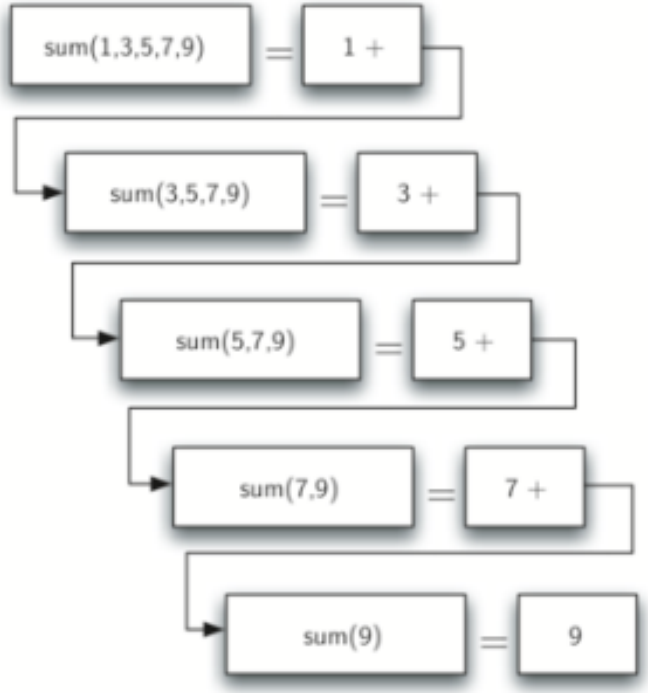
(2) Um algoritmo recursivo deve modificar o seu estado e se aproximar do caso base.

(3) Um algoritmo recursivo deve chamar a si mesmo, recursivamente.

- **Vamos ver um exemplo que faz tudo isso!!!**

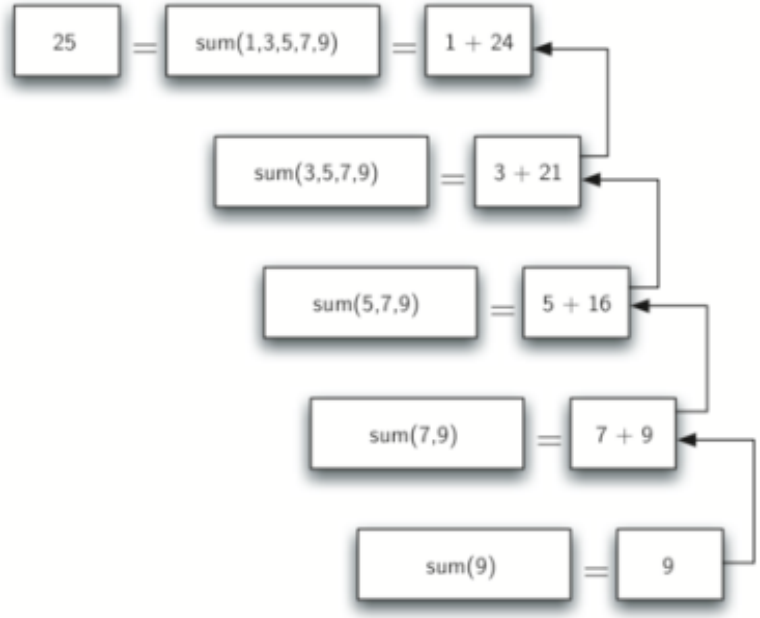
Exemplo de recursividade 1

Calculando a Soma de Uma Lista de Números:



Série de chamadas recursivas para somar

Fonte: IME-USP



Returns das chamadas recursivas para somar

Fonte: IME-USP

- Vamos ver como funciona no site!!!

Fatorial de um Número Inteiro

- O fatorial de um inteiro positivo n, denotado por $n!$, é definido como o produto dos inteiros de 1 até n.
- Formalmente:

$$n! = \begin{cases} 1 & \text{se } n = 0, \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{se } n \geq 1. \end{cases}$$

- Exemplos:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120.$$

- Importância:
  - $n!$ representa o número de maneiras de permutar n itens distintos.
  - Exemplo: Para $\{a, b, c\}$, existem $3! = 6$ arranjos possíveis.

Recursão no Cálculo do Fatorial

- A recursão consiste em definir uma função em termos dela mesma.
- Definição recursiva do fatorial:

$$n! = \begin{cases} 1 & \text{se } n = 0, \\ n \cdot (n-1)! & \text{se } n \geq 1. \end{cases}$$

- **Base Case (Caso Base):**

$$0! = 1.$$

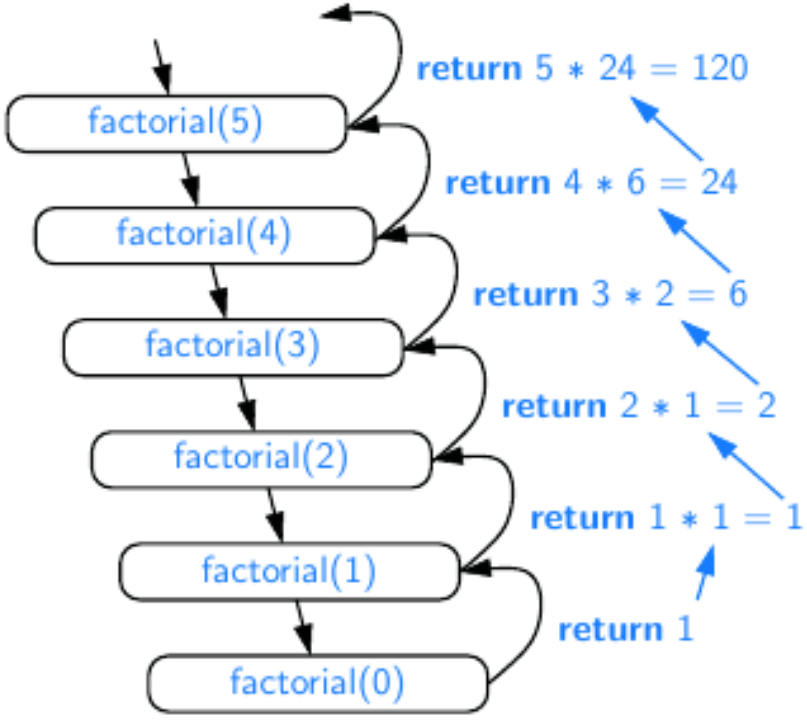
  É o ponto de parada da recursão.
- **Recursive Case (Caso Recursivo):**

$$n! = n \cdot (n-1)! \quad \text{para } n \geq 1.$$

  O fatorial de n depende do fatorial de $n-1$.

Exemplo de recursividade 2

Calculando Fatorial por Recursão:



Traço de recursão para a chamada factorial(5). Fonte: Goodrich, M. T., et. al. (2014). Data structures and algorithms in Java. John wiley & sons.

- Cada entrada do traço corresponde a uma chamada recursiva. Cada nova chamada de método recursivo é indicada por uma seta para baixo direcionada para uma nova invocação. Quando o método retorna, desenha-se uma seta mostrando esse retorno e o valor de retorno pode ser indicado ao lado dessa seta.

Busca

Busca sequencial

- **Busca** é o processo algorítmico de **encontrar** um item específico **numa coleção** de itens. Uma busca tipicamente devolve True ou False, indicando se o item está presente ou não. Ela também pode ser modificada em algumas situações para retornar o elemento encontrado.[2]

```
>>> 15 in [3,5,2,4,1]
False
>>> 3 in [3,5,2,4,1]
True
>>>
```

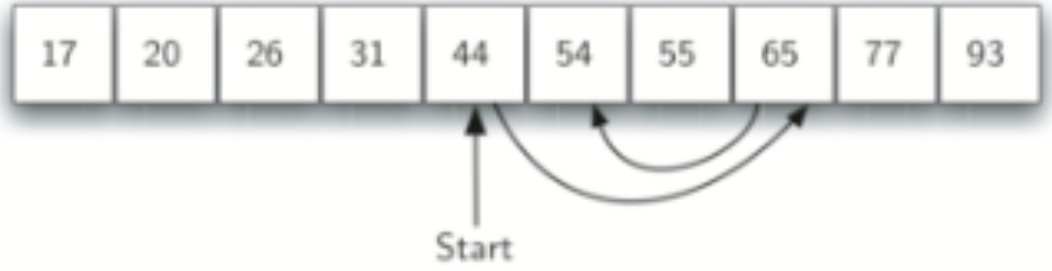
Busca exemplo
Fonte: IME-USP

- Busca sequencial
- Vamos investigar o funcionamento da busca sequencial no notebook!

Busca binária

Na **busca sequencial**, quando iniciamos a comparação a partir do primeiro item, há no máximo mais **n-1** elementos a serem buscados se o primeiro item não era o que estávamos procurando. Em vez de procurar o item sequencialmente, uma **busca binária** irá começar examinando o **item do meio**. Se esse elemento é o que estamos buscando, a procura terminou. Se não for o item correto, podemos utilizar **o fato da lista estar ordenada** para **eliminar metade dela**. Se o item que estamos procurando for maior que o elemento do meio, sabemos que a metade inferior (contando com o item do meio) não precisa mais ser levada em consideração. O item, se estiver na lista, necessariamente está na metade superior[2].

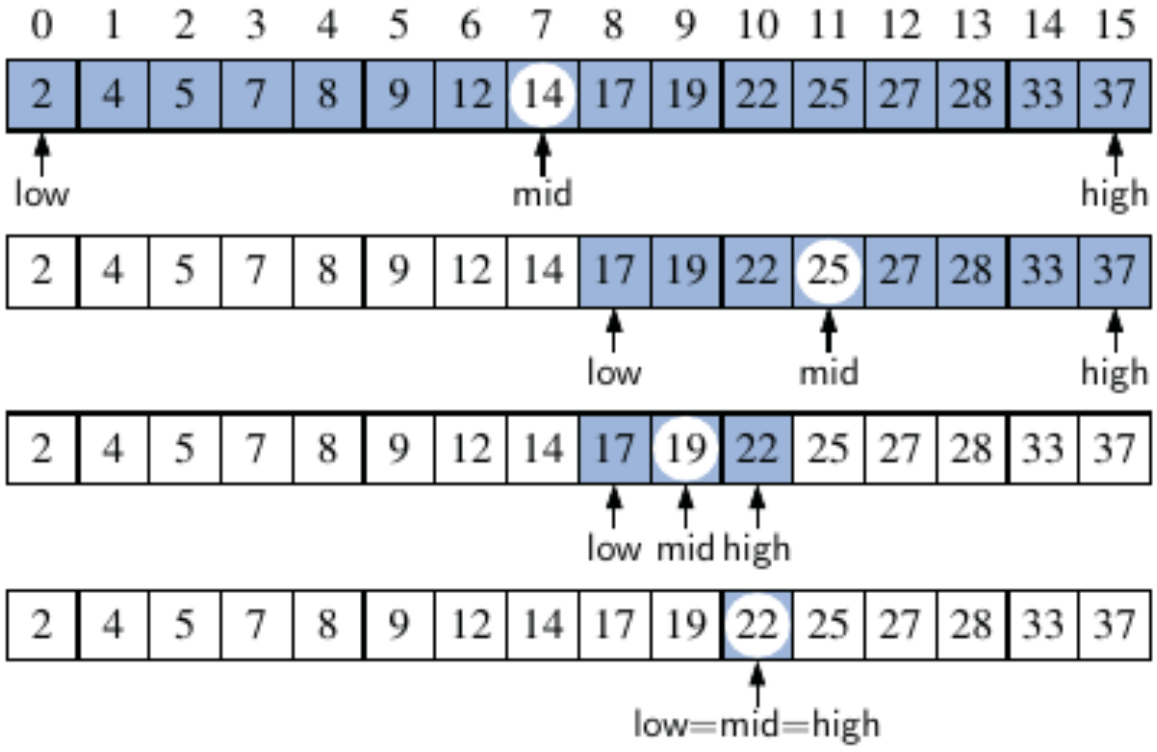
- **Exemplo:** Métodos dividir para conquistar!



Busca binária exemplo

IME-USP

Exemplo de Busca binária



Exemplo de uma busca binária para o valor-alvo 22 em um array ordenado com 16 elementos.

Fonte: Goodrich, M. T., et. al. (2014). Data structures and algorithms in Java. John wiley & sons.

Análise da Busca binária

Objetivo: Avaliar o tempo de execução do algoritmo de busca binária.

Observação Principal:

- Em cada chamada recursiva, um número constante de operações é executado.
- Logo, o tempo total depende essencialmente do número de chamadas recursivas.

Redução Recursiva

Fato Crucial:

- Seja `low` e `high` o intervalo do subarranjo atual.
- A cada chamada recursiva, o número de candidatos (elementos no subarranjo) é reduzido *pela metade*.

$$\texttt{high} - \texttt{low} + 1 \quad \rightarrow \quad \frac{\texttt{high} - \texttt{low} + 1}{2}.$$

Exemplo de Redução:

$$\begin{aligned}
&\text{Após a } 1^{\text{a}} \text{ chamada:} && \leq \frac{n}{2},\\
&\text{após a } 2^{\text{a}} \text{ chamada:} && \leq \frac{n}{4},\\
&\ldots\\
&\text{após a } j\text{-ésima chamada:} && \leq \frac{n}{2^j}.
\end{aligned}$$

Conclusão 1: $\mathcal{O}(\log n)$

Número Máximo de Chamadas:

$$\frac{n}{2^r} < 1 \implies r > \log n.$$

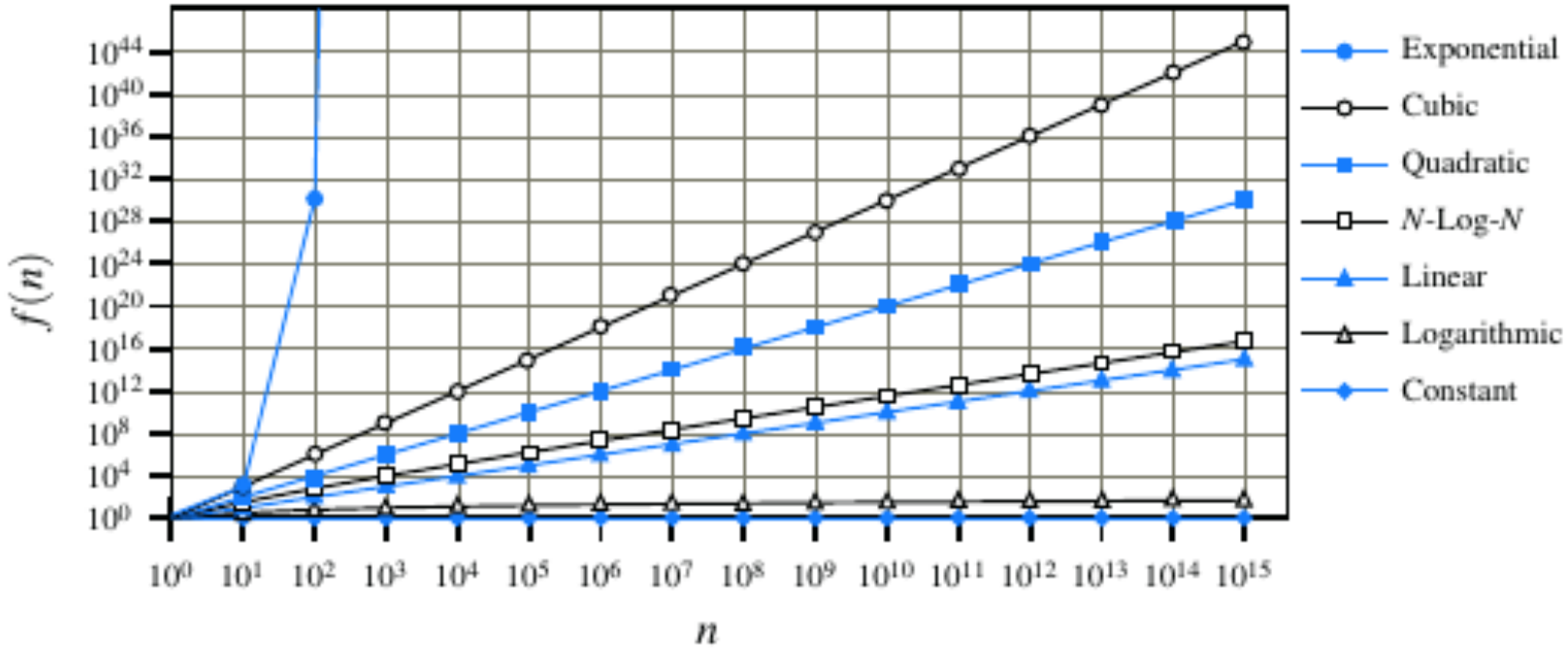
Assim, o menor inteiro r satisfazendo essa condição é:

$$r = \lfloor \log n \rfloor + 1.$$

Consequência:

- A busca binária realiza no máximo $\lfloor \log n \rfloor + 1$ chamadas recursivas.
- Logo, seu tempo de execução é $\mathcal{O}(\log n)$.

Conclusão 2: Comparando taxas de crescimento



constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

Considerações

Considerações

- **Recordando!!!**
  - Recursão.
  - Busca.
  - Exercícios.

    Referências:
  - Goodrich, M. T., et. al. (2014). Data structures and algorithms in Java. John wiley & sons.
  - Miller, B. N.; Ranum, D. L. Problem solving with algorithms and data structures using python, 2Ed. Franklin, Beedle and Associates Inc., 2011