Algoritmos

O que é um algoritmo?

Problema: encontrar elemento

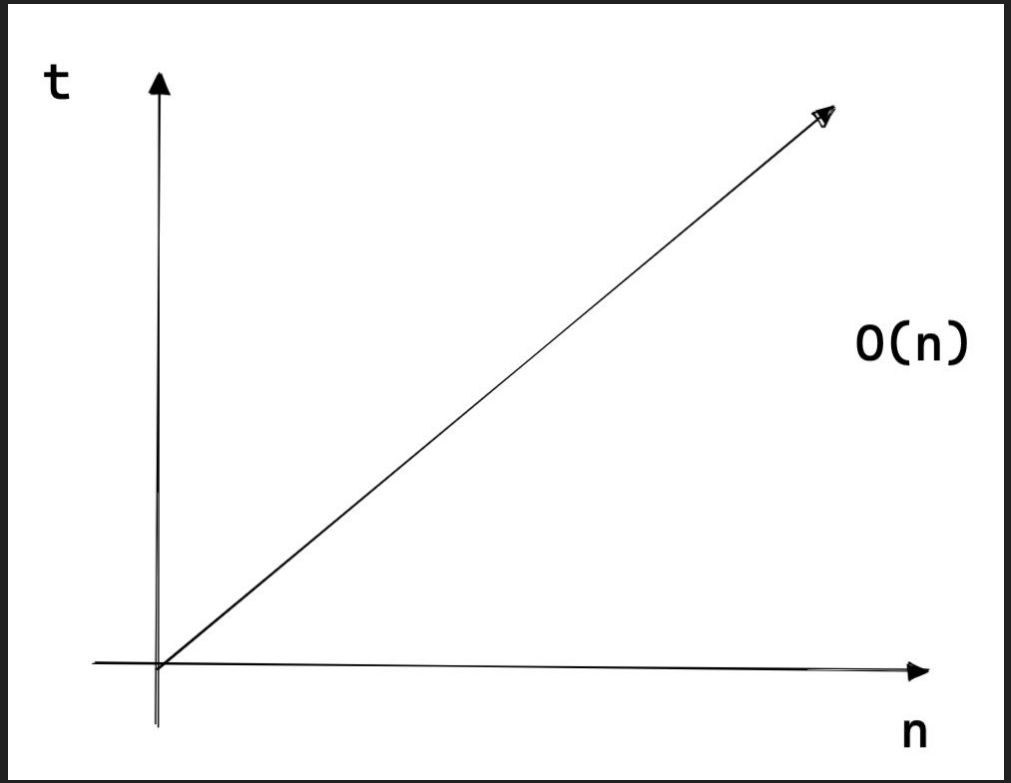
Implementação: Busca Linear

Busca Aleatória

Corretude vs Eficiência

Analisando complexidade

Notação Big-O (pior caso) – Busca Linear



Conseguimos melhorar?

Jogo: acerte o número!

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

Resposta: 77

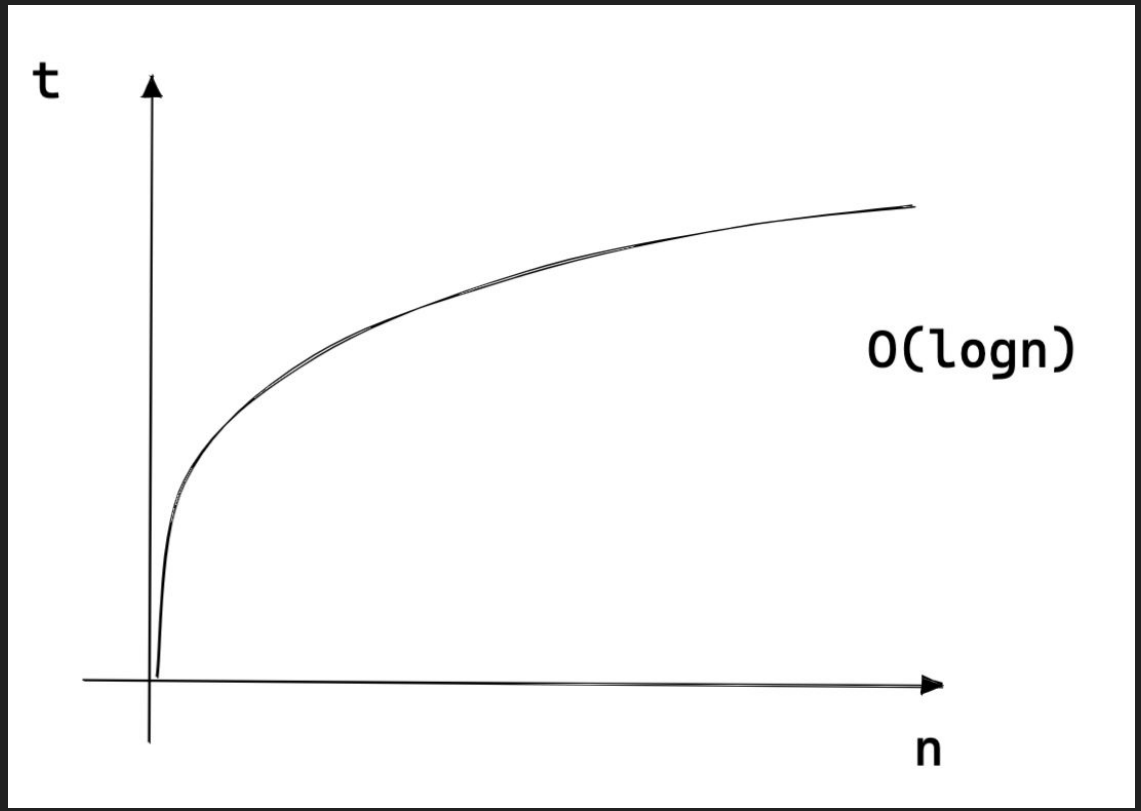
Quantos passos precisaríamos no pior caso?

Resposta: 7 passos

E se tivéssemos 1.000.000 números possíveis?

Resposta: 20 passos

O(logn) – Busca Binária



Implementação: Busca Binária

Comparar execução

Busca simples vs binária

busca_simples.py 99.999.999 8

busca_binaria.py 99.999.999 8

(10**8 = 100 milhões)

Comparando tempo de execução com *time.perfcounter()*

Randomizando entradas

random.randint(a, b)

Analisando complexidade: Horários Disponíveis

- Versão simplificada do exercício do Módulo 14
  - Link para solução completa
- Qual a complexidade desse algoritmo?

```
while start < end:
    if not Agendamento.objects.filter(data_horario=start).exists():
        horarios_disponiveis.add(start)
    start = start + delta

return horarios_disponiveis
```

Utilizando o módulo *timeit*

Código e *CLI*

List vs *Set* (*timeit*)

HashMap / HashTable

Por que não utilizar sempre Set?

Análise de Memória (espaço)

sys.getsizeof(obj)

**Set não garante ordem*

Sempre pense nos *tradeoffs*!
Tempo x Espaço

** Legibilidade!*

Exemplos de algoritmos e complexidade

- Busca *hash* – O(1)
- Busca binária – O(logn)
- Busca linear – O(n)
- Horários disponíveis – O(n^2)
- Permutações – O(n!)
  - Caixeiro Viajante

Ferramentas utilizadas

- *time.perfcounter()*
- *timeit.timeit (CLI)*
- *random.randint(a, b)*
- *sys.getsizeof(object)*