A notação O (1)

$$O(g(n)) = \{f(n) : \exists_{c,n_0>0} \text{ tais que } \forall_{n\geq n_0}\ 0 \leq f(n) \leq c\, g(n)\}$$

$$O(n) = \{f(n) : \exists_{c,n_0>0} \text{ tais que } \forall_{n\geq n_0}\ 0 \leq f(n) \leq c\, n\}$$

$$n = O(n) \qquad 2n+5 = O(n) \qquad \log n = O(n) \qquad n^2 \neq O(n)$$

$$O(n^2) = \{f(n) : \exists_{c,n_0>0} \text{ tais que } \forall_{n\geq n_0}\ 0 \leq f(n) \leq c\, n^2\}$$

$$n^2 = O(n^2) \qquad 4n^2+n = O(n^2) \qquad n = O(n^2) \qquad n^3 \neq O(n^2)$$

$$O(\log n) = \{f(n) : \exists_{c,n_0>0} \text{ tais que } \forall_{n\geq n_0}\ 0 \leq f(n) \leq c\, \log n\}$$

$$1+\log n = O(\log n) \qquad \log n^2 = 2\log n = O(\log n) \qquad n \neq O(\log n)$$

Escreve-se $f(n) = O(g(n))$ em vez de $f(n) \in O(g(n))$

Lê-se $f(n)$ é O de $g(n)$

A notação O (2)

$$\Omega(g(n)) = \{f(n) : \exists_{c,n_0>0} \text{ tais que } \forall_{n\geq n_0}\ 0 \leq c\,g(n) \leq f(n)\}$$

$$n = \Omega(n) \quad n^2 = \Omega(n) \quad \log n \neq \Omega(n^2)$$

$$\Theta(g(n)) = \{f(n) : \exists_{c_1,c_2,n_0>0} \text{ t.q. } \forall_{n\geq n_0}\ 0 \leq c_1\,g(n) \leq f(n) \leq c_2\,g(n)\}$$

$$3n^2 + n = \Theta(n^2) \quad n \neq \Theta(n^2) \quad n^2 \neq \Theta(n)$$

$$o(g(n)) = \{f(n) : \forall_{c>0}\ \exists_{n_0>0} \text{ tal que } \forall_{n\geq n_0}\ 0 \leq f(n) < c\,g(n)\}$$

$$n = o(n^2) \quad n^2 \neq o(n^2)$$

$$\omega(g(n)) = \{f(n) : \forall_{c>0}\ \exists_{n_0>0} \text{ tal que } \forall_{n\geq n_0}\ 0 \leq c\,g(n) < f(n)\}$$

$$n = \omega(\log n) \quad n^2 = \omega(\log n) \quad \log n \neq \omega(\log n)$$

A notação O (3)

Traduzindo...

$f(n) = O(g(n))$ $f(n)$ não cresce mais depressa que $g(n)$

$f(n) = o(g(n))$ $f(n)$ cresce mais devagar que $g(n)$

$f(n) = \Omega(g(n))$ $f(n)$ não cresce mais devagar que $g(n)$

$f(n) = \omega(g(n))$ $f(n)$ cresce mais depressa que $g(n)$

$f(n) = \Theta(g(n))$ $f(n)$ e $g(n)$ crescem com o mesmo ritmo

Pseudo-código

Exemplo

PESQUISA-LINEAR(V, k)

```
n <- |V|                                 // inicialização
i <- 1

while i <= n and V[i] != k do            // pesquisa
    i <- i + 1

if i <= n then                           // resultado:
    return i                             //  - sucesso
else
    return -1                            //  - insucesso
```

`|V|`	nº de elementos de um vector — $O(1)$
`V[1..|V|]`	elementos do vector
`and` e `or`	só é avaliado o segundo operando se necessário
`variável.campo`	acesso a um campo de um "objecto"

Análise da complexidade (1)

Exemplo

Análise da complexidade temporal, no pior caso, da função PESQUISA-LINEAR, por linha de código

1. Obtenção da dimensão de um vector, afectação: operações com complexidade constante

$$O(1) + O(1) = O(1)$$

2. Afectação: $O(1)$
3. Acessos a `i`, `n`, `V[i]` e `k`, comparações e saltos condicionais com complexidade constante

$$4\,O(1) + 2\,O(1) + 2\,O(1) = O(1)$$

   Executada, no pior caso, $|\mathtt{V}|+1$ vezes

$$(|\mathtt{V}| + 1) \times O(1) = O(|\mathtt{V}|)$$

Análise da complexidade (2)

Exemplo

4. Acesso a `i`, soma e afectação: $O(1) + O(1) + O(1) = O(1)$
   Executada, no pior caso, `|V|` vezes

$$|\mathtt{V}| \times O(1) = O(|\mathtt{V}|)$$

5. Acesso a `i` e `n`, comparação e salto condicional com complexidade constante

$$2\,O(1) + O(1) + O(1) = O(1)$$

6. Saída de função com complexidade constante: $O(1)$
8. Saída de função com complexidade constante: $O(1)$

Análise da complexidade (3)

Exemplo

Juntando tudo

$$
\begin{aligned}
O(1) + O(1) + O(|\mathtt{V}|) + O(|\mathtt{V}|) + O(1) + \max\{O(1), O(1)\} = \\
= 4\,O(1) + 2\,O(|\mathtt{V}|) = \\
= O(|\mathtt{V}|)
\end{aligned}
$$

No pior caso, a função PESQUISA-LINEAR tem complexidade temporal linear na dimensão do vector `V`

Se n for a dimensão do vector `V`, a função tem complexidade temporal

$$O(n)$$

Isto significa que, para um *input* de dimensão n, o tempo que a função demora a executar, no pior caso, é

$$T(n) = O(n)$$

Ainda a pesquisa linear

De um valor num vector ordenado

PESQUISA-LINEAR-ORD(V, k)

```
n <- |V|                               // inicialização
i <- 1

while i <= n and V[i] < k do           // pesquisa
    i <- i + 1

if i <= n and V[i] = k then            // resultado:
    return i                           //  - sucesso
else
    return -1                          //  - insucesso
```

Pesquisa dicotómica ou binária

De um valor num vector ordenado

PESQUISA-DICOTÓMICA(V, k)

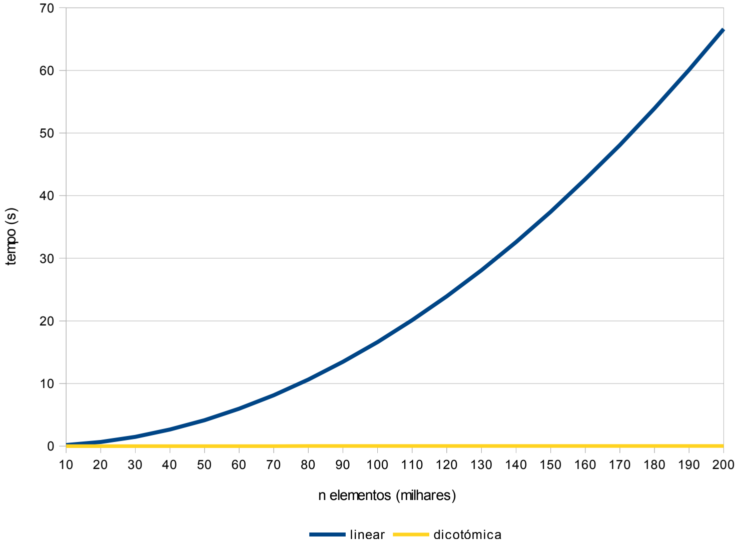
```
n <- |V|
return PESQUISA-DICOTÓMICA-REC(V, k, 1, n)
```

PESQUISA-DICOTÓMICA-REC(V, k, i, f)

```
if i > f then
    return -1                   // intervalo vazio

m <- (i + f) / 2

if k < V[m] then
    return PESQUISA-DICOTÓMICA-REC(V, k, i, m - 1)

if k > V[m] then
    return PESQUISA-DICOTÓMICA-REC(V, k, m + 1, f)

// k = V[m]
return m
```

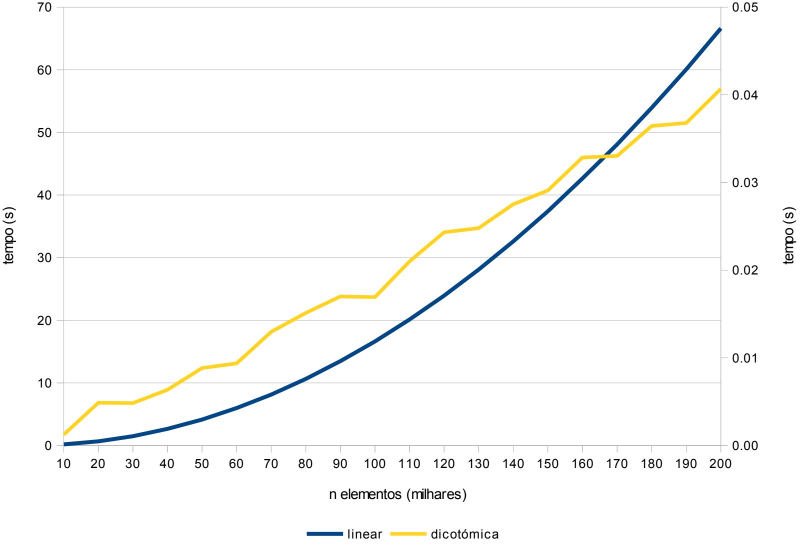
Pesquisa linear e dicotómica

Dos *n* elementos de um vector



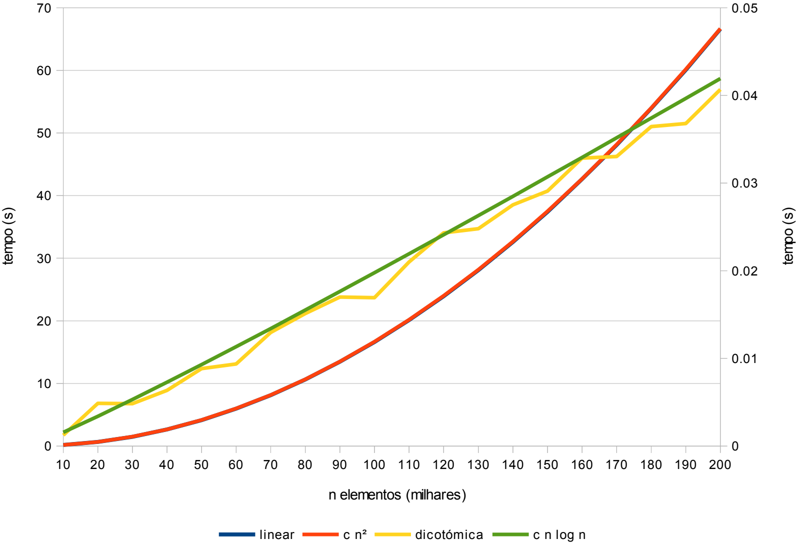
Pesquisa linear e dicotómica (com escalas diferentes)

Dos n elementos de um vector



Pesquisa linear e dicotómica (com escalas diferentes)

Dos n elementos de um vector



Tries

A estrutura de dados *trie*

Uma *trie* é uma árvore cujos nós têm filhos que correspondem a símbolos do alfabeto das chaves

Uma chave está contida numa *trie* se o percurso que ela induz na *trie*, a partir da sua raiz, termina num nó que marca o fim de uma chave

As *tries* apresentam algumas características que as distinguem de outras estruturas de dados

1. A complexidade das operações não depende do número de elementos que ela contém
2. As chaves não têm de estar explicitamente contidas na *trie*
3. As operações não se baseiam em comparações entre chaves

Uma *trie*

Exemplo

Representação de uma *trie* com as chaves (palavras) cal, cor, saco, sal e salto