Tempo de execução

→ Vários fatores afetam o tempo de execução de um programa:
- ↳ computadores mais / menos potentes
- ↳ instâncias pequenas / grandes
- ↳ sistema operacional
- ↳ linguagem de programação
- ↳ estrutura de dados
- ↳ programador ...

→ Queremos um modelo de computação que independa desses detalhes mas nos dê informação suficiente.
- ↳ Ele realiza passos básicos rapidamente sobre "números pequenos":
  - aritmética
  - lógica
  - controle de fluxo
  - relacional
  - movimentação

Tempo de execução

DEFINIÇÃO: O tempo de execução de um algoritmo é a quantidade de passos básicos executados por ele sobre uma instância de entrada.

→ O tempo tende a crescer conforme o tamanho da entrada cresce, e por isso usamos uma função nesse tamanho para descrevê-lo: $T(n)$, $T(n, m)$

→ O tamanho da entrada varia com o problema
- ↳ Vetor: quantidade de elementos (n)
- ↳ Número: quantidade de bits na representação binária $(\log n)$
- ↳ Dois vetores: quantidade de elementos somados $(n+m)$
- ↳ Matriz: quantidade de elementos (nm)

Exemplo de cálculo do Tempo

BUSCALINEAR(A, n, k)	p_k Se há elem. c/ chave k	Se não há
1 $i = 1$	t	t
2 **enquanto** $i \leq n$ **e** $A[i].\text{chave} \neq k$ **faça**	$5t\,p_k$	$t(n+1) + 4tn$
3 $\quad i = i + 1$	$2t\,(p_k - 1)$	$2tn$
4 **se** $i \leq n$ **e** $A[i].\text{chave} == k$ **então**	$5t$	$5t$
5 $\quad$ **devolve** i	t	
6 **devolve** -1		t

t = tempo de um passo básico

se elem. existe $\Rightarrow t + 5tp_k + 2t(p_k - 1) + 6t = 7tp_k + 5t$ → $12t$ → $7tn + 5t$

se elem. não existe $\Rightarrow t + t(n+1) + 4tn + 2tn + 6t = 7tn + 8t$

Outro exemplo de cálculo do tempo

BUSCABINARIA(A, n, k)	
1 $esq = 1$	t
2 $dir = n$	t
3 **enquanto** $esq < dir$ **faça**	tr
4 $\quad meio = \lfloor (esq + dir)/2 \rfloor$	$4t(r-1)$
5 $\quad$ **se** $k > A[meio].\text{chave}$ **então**	$3t(r-1)$
6 $\quad\quad esq = meio + 1$	$2t(r-1)$
7 $\quad$ **senão**	
8 $\quad\quad dir = meio$	
9 **se** $A[esq].\text{chave} == k$ **então**	$4t$
10 $\quad$ **devolve** esq	t
11 **devolve** -1	

r = qtd de vezes que o teste da linha 3 executa

↳ mas conseguimos calcular r

total $= 10tr - 3t$

$= 10t \lg n - 3t$

Resumo

	Busca Linear	Busca Binária
Elem. em A	$7t p_k + 5t$	$10t r - 3t$
$k = A[1].chave$	$12t$	$10t\lceil lg\, m \rceil + 17t$
$k = A[\frac{m}{2}].chave$	$\frac{7tm}{2} + 5t$	
$k = A[m].chave$	$7tm + 5t$	
Não há elem. em A	$7tm + 8t$	

Análise por casos

DEFINIÇÃO: O tempo de melhor caso de um algoritmo é o menor tempo de execução do algoritmo dentre os tempos de execução de todas as instâncias de um dado tamanho n.

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

$12t$

```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

$10 \lg n + 7t$

Análise por casos

DEFINIÇÃO: O tempo de pior caso de um algoritmo é o maior tempo de execução do algoritmo dentre os tempos de execução de todas as instâncias de um dado tamanho n.

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

$7tn + 8t$

```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

$10t \lg n + 17t$

Análise por casos

→ Se $T(n)$ é o tempo de uma entrada qualquer de tamanho n,

$$\text{TEMPO NO MELHOR CASO} \leq T(n) \leq \text{TEMPO NO PIOR CASO}$$

→ Portanto esses tempos nos dão garantias.

→ $12t \leq$ TEMPO DA BUSCA LINEAR $\leq 7tn + 8t$

→ $10t \log n + 7t \leq$ TEMPO DA BUSCA BINÁRIA $\leq 10t \log n + 17t$

Análise por casos

DEFINIÇÃO: O tempo de caso médio de um algoritmo é a média do tempo de execução de todas as instâncias de tamanho n.

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

→ Consideramos algo sobre a distribuição das entradas e fazemos uma análise probabilística.

→ Pode ser tão ruim quanto o pior caso.

→ Análises de caso médio costumam ser mais complicadas.

Bons algoritmos

DEFINIÇÃO: Um algoritmo é eficiente se seu tempo de execução no pior caso puder ser descrito por uma função que é limitada superiormente por uma função polinomial no tamanho da entrada.

→ Ambas as buscas são eficientes, portanto.
Mas qual é a melhor? Qual é a mais eficiente?

Notação assintótica

→ É uma abstração que nos permite focar no que ocorre com $f(n)$ quando n cresce indefinidamente
  ↳ Termos de menor ordem não importam
  ↳ constantes não importam
  ↳ $f(n) = 2n^3 + 4000n^2 + 10000n - 47$

→ Exemplo: $f(n) = 475n + 34$ e $g(n) = n^2 + 3$
  ↳ $f(n) \leq g(n)$ → verdade se $n \geq 476$ ($f(4) = 1934$, $g(4) = 19$)
  ↳ $f(n) \leq 475\, g(n)$ → verdade se $n \geq 1$

Notação que limita funções superiormente

DEFINIÇÃO: Seja n um inteiro positivo e sejam $f(n)$ e $g(n)$ funções positivas. Dizemos que

$$f(n) = O(g(n)) \quad (\text{ou } f(n) \text{ é } O(g(n)))$$

se existem constantes positivas C e n_0 tais que

$$f(n) \leq C \cdot g(n) \qquad \forall\, n \geq n_0$$

Ex: $f(n) = 7tn + 8t$

$g_1(n) = n$ $\quad g_2(n) = n^2$ $\quad g_3(n) = \sqrt{n}$

Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} \neq \infty$, então f é $O(g)$

Observação

Escrevemos $O(1)$ e $\Omega(1)$ para denotar limites por constantes.

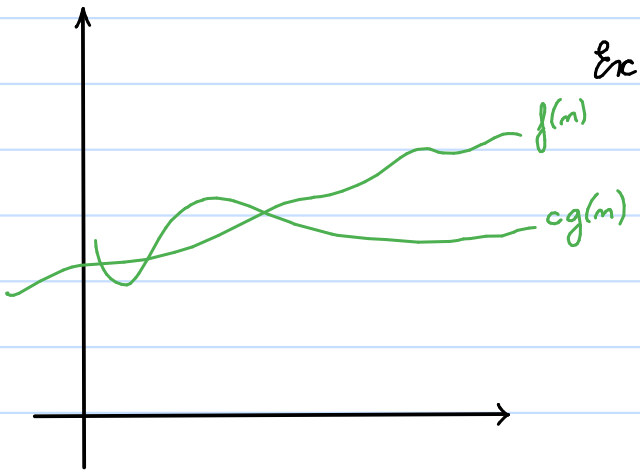
Notação que limita funções inferiormente

DEFINIÇÃO: Seja n um inteiro positivo e sejam $f(n)$ e $g(n)$ funções positivas. Dizemos que

$$f(n) = \Omega(g(n)) \quad \text{(ou } f(n) \text{ é } \Omega(g(n))\text{)}$$

se existem constantes positivas c e n_0 tais que

$$f(n) \geq c \cdot g(n) \qquad \forall\ n \geq n_0$$



Exc: $f(n) = 7tn + 8t$

$g_1(n) = n \qquad g_2(n) = n^2 \qquad g_3(n) = \sqrt{n}$

Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$, então f é $\Omega(g)$

Notação que limita funções justamente

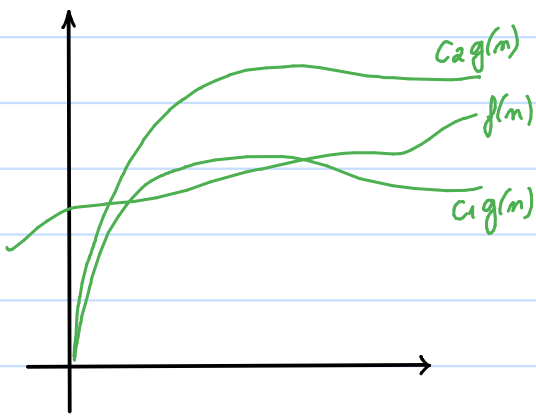
DEFINIÇÃO: Seja n um inteiro positivo e sejam $f(n)$ e $g(n)$ funções positivas. Dizemos que

$$f(n) = \Theta(g(n)) \quad \text{(ou } f(n) \text{ é } \Theta(g(n))\text{)}$$

se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \qquad \forall\ n \geq n_0$$

→ Obs: $f(n)$ é $\Theta(g(n))$ se e somente se $f(n)$ é $O(g(n))$ e é $\Omega(g(n))$



Exc: $f(n) = 7tn + 8t$

$g_1(n) = n \qquad g_2(n) = n^2 \qquad g_3(n) = \sqrt{n}$

Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, então f é $\Theta(g)$

↳ $\neq 0, \infty$

mais exemplos de notação assintótica

$f(n) = 10n^2 + 5n + 3$

$f(n) = 42 \lg n$

↳ $O(n)$, $O(\lg^2 n)$, $O(\lg_{50} n)$, ~~$O(\lg n^2)$~~

Notar que $\log_a n = \Theta(\log_b n)$

$f(n) = 3n^4 - 270n^3 + 10n^2 - 15n$

↳ $3n^4 - 270n^3 + 10n^2 - 15n \leq 3n^4 + n^3 + 10n^2 + n$

$\leq 3n^4 + n^4 + 10n^4 + n^4$

$= 15n^4$ $\therefore c = 15$ e $n_0 = 1$

↳ $3n^4 - 270n^3 + 10n^2 - 15n \geq 3n^4 - 270n^3 - 15n$

$\geq 3n^4 - 270n^3 - 15n^3$

$= 3n^4 - 285n^3$

$\overset{??}{\geq} 3n^4 - 2n^4 = n^4$ $\therefore c = 1$ e $n_0 = 143$

só se $-285n^3 \geq -2n^4 \Rightarrow 285 \leq 2n \Rightarrow n \geq 142{,}5$

$f(n) = 42 \log_2 n$ $\quad 42 \lg n \leq 42\sqrt{n}$ se $n \geq 16$

Notar que $\lg^k n$ é $O(n^t)$

$\lim \frac{\lg n}{n}$ precisa de l'Hôpital

Vida real

$5 \log_2 n + \sqrt{n}$ é $O(\sqrt{n})$?

" é $O(\lg n)$?

" é $O(n)$?

" é $\Omega(n)$?

" é $\Omega(\lg n)$?

" é $\Omega(\sqrt{n})$?

" é $O(n^2)$?

" é $\Omega(n^2)$?

c

$\lg n$

$\sqrt{n}$

n^c $\quad c \geq 1$

$n^c \lg n$

2^n

$n!$

Erro comum

"Notação O é de pior caso, Ω é de melhor caso e Θ é de caso médio"

	Busca Linear	
melhor caso	$12t$	$\Theta(1), O(1), O(n), O(n^2), \Omega(1)$
Caso médio	$\frac{7tn}{2} + \frac{17t}{2}$	$O(n), O(n^3), O(n \lg n), \Omega(n), \Omega(\sqrt{n})$
Pior caso	$7tn + 8t$	$O(n), O(n^2), O(n \lg n), \Omega(1), \Omega(\sqrt{n}), \Omega(\lg n)$

Se $T(n)$ é o tempo do algoritmo em uma entrada qualquer, então

$f(n) \leq$ TEMPO MELHOR CASO $\leq T(n) \leq$ TEMPO PIOR CASO $\leq g(n)$

"O algoritmo tem tempo X" $\equiv$ "Qualquer instância tem tempo X"

Tempo com notação assintótica

→ Não vamos mais calcular as expressões de tempo contando detalhadamente os passos básicos

→ Vamos fazer uma análise mais rápida usando notação assintótica.

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

A BuscaLinear tem tempo $O(n)$ porque o laço executa no máximo n vezes e o corpo do laço leva tempo constante para ser executado.

A BuscaLinear tem tempo $\Theta(n)$ no pior caso. Tem tempo $\Theta(1)$ no melhor caso.

A BuscaBinaria tem tempo $\Theta(\lg n)$, porque quando o laço começa temos um vetor de tamanho n, que é sempre diminuído pela metade e ao final temos um vetor de tamanho 1 (qq caso)

Exemplo completo de solução de problema

Problema *Soma em Vetor*

Entrada: $\langle A, n, k \rangle$, onde $A[1..n]$ é um vetor de tamanho n e k é um inteiro.
Saída: Par de posições i e j, $i \neq j$, tais que $A[i] + A[j] = k$, caso existam, ou o par $-1, -1$, caso contrário.

	1	2	3	4	5	6	7	8	9
A	3	6	2	12	4	9	10	1	5

k = 10

k = 2

Resolvendo problemas

1) Entender o problema
2) Criar um algoritmo
   ↳ Algum outro problema conhecido pode ajudar?
   ↳ Alguma técnica pode ajudar?
3) Verificar se o algoritmo está correto
4) Verificar qual o tempo do algoritmo
5) Será que dá para fazer melhor? Volte para 2.

Problema da soma em vetor reduz para busca

```
SomaK(A, n, k)
1 para j = 1 até n, incrementando faça
2     i = BuscaLinear(A, n, k − A[j])
3     se i ≠ −1 e i ≠ j então
4         devolve i, j
5 devolve −1, −1
```

	1	2	3	4	5	6	7	8	9
A	3	6	2	12	4	9	10	1	5

$k = 16$

① Está correto?

$P(t)$ = "Antes da t-ésima iteração, $i = t$ e $\forall x \in A[1..i-1] \not\exists y \in A$ tal que $x + y = k$"

② Qual o tempo?

$O(n^2)$, porque o laço executa no máximo n vezes e cada execução leva tempo $O(n)$, da BUSCALINEAR.

③ Dá para fazer melhor?

Se $j = 3$, buscamos por $k - A[3]$ (em part. na pos. 7). Se $j = 7$, buscamos por $k - A[7]$ (em part. na pos. 3).

para $j = n$ até 2, decrementando
i = BUSCALINEAR$(A, j-1, k - A[j])$

Resolvendo diretamente

```
SomaK_v2(A, n, k)
1 para i = 1 até n − 1, incrementando faça
2     para j = i + 1 até n, incrementando faça
3         se A[i] + A[j] == k então
4             devolve i, j
5 devolve −1, −1
```

① Está correto?

$R(t)$ = "Antes da t-ésima iteração começar, $j = i + t$ e $\forall i < j' < j$, $A[i] + A[j'] \neq k$".

$R(n - (i+1) + 1 + 1) = R(n - i + 1)$

$\Rightarrow \forall i < j' < i + (n - i + 1) = n + 1$

$P(t)$ = "antes da t-ésima iteração começar, $i = t$ e $\forall 1 \leq i' < i$ $\forall i+1 \leq j \leq n$, $A[i] + A[j] \neq k$"

$P(n) \Rightarrow \forall 1 \leq i' < n, \forall n \leq j \leq n$, $A[i] + A[j] \neq k$.

② Qual o tempo?

$O(n^2)$ porque executa no máx. para todos os valores possíveis de j, que toma até $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \frac{n(n-1)}{2}$, que é $\Theta(n^2)$.

③ Dá para fazer melhor? Sim

extras

Calcular o tempo dos algoritmos a seguir:

```
MULTIPLY (x, y)
    prod = 0                                  θ(1)
    enquanto x > 0 :                          θ(lg x)
        se x é ímpar                          θ(lg x)
            prod = prod + y                   θ(lg x) · θ(lg y)
        x = ⌊x/2⌋                             θ(lg x)
        y = y + y                             θ(lg x) θ(lg y)
    devolve prod                              θ(1)
```

$\sum_{k=1}^{\lg x} \theta(1)$

$\sum_{k=1}^{\lg x} \theta(\lg y)$

```
FUN(n)
    c = 0                                     θ(1)
    i = n                                     θ(1)
    enquanto i > 0 :                          θ(lg n)
        para j = 1 até i, inc                 θ(i) → n + n/2 + n/4 + ... + 1
            c = c + 1
        i = i/2
    devolve c
```

$\sum_{i=1}^{\lg n} \theta(1)$

$\sum_{x=0}^{\lg n} \frac{n}{2^x} = n\left(\frac{1 - (1/2)^{\lg n}}{1 - 1/2}\right)$

$= 2n\left(1 - n^{\lg 1/2}\right)$

$= 2n - 2$

$\lg \frac{1}{2} = \lg 1 - \lg 2 = -1$

extras

Calcular o tempo dos algoritmos a seguir:

```
MULTIPLY (x, y)
    prod = 0
    enquanto x > 0 :
        se x é ímpar
            prod = prod + y
        x = ⌊x/2⌋
        y = y + y
    devolve prod
```

O tempo de MULTIPLY(x, y) é $\Theta(\log x \log y)$ pois o laço executa $\log x$ vezes (x varia de seu valor inicial a 0, sempre sendo dividido por 2) e cada iteração leva tempo $\Theta(\log y)$, para fazer as operações "+ y".

↳ Polinomial no tamanho da entrada

```
FUN(n)
    c = 0
    i = n
    enquanto i > 0 :
        para j = 1 até i, inc
            c = c + 1
        i = i / 2
    devolve c
```

O tempo de FUN(n) é $\Theta(n)$, o laço para executa i vezes para cada valor de i em $\{n, n/2, n/4, \ldots, 1\}$ realizando operações constantes nessas iterações e

$$\sum_{k=1}^{\log_2 n} \frac{n}{2^k} = n\left(\frac{1 - (1/2)^{\log n}}{1 - 1/2}\right)$$

$$= 2n\left(1 - n^{\log 1/2}\right)$$

$$= 2n - 2$$

↳ Exponencial no tamanho da entrada