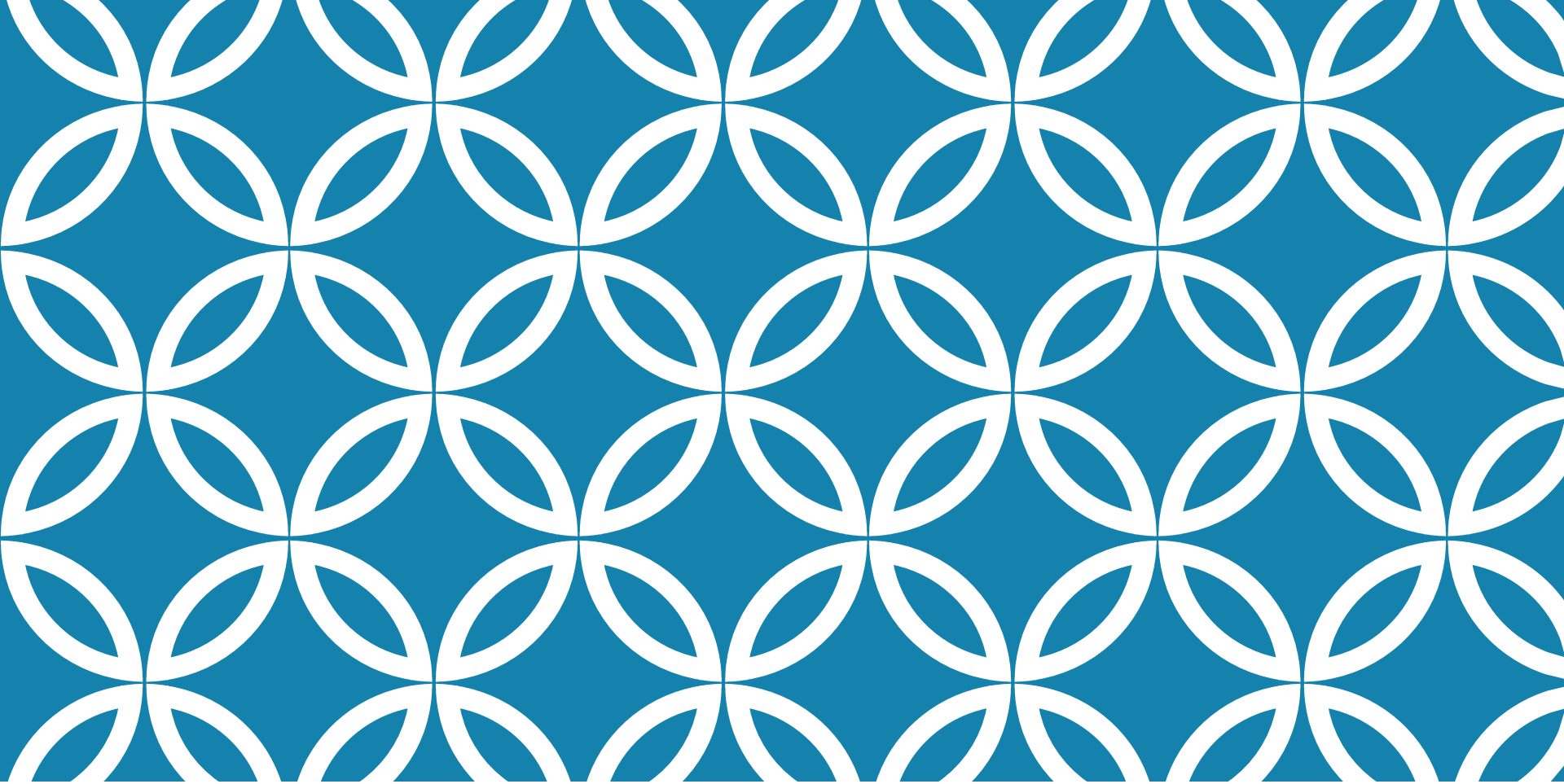


ESTRUTURAS DE DADOS - VETORES

Prof. André Vignatti

DEFINIÇÕES USADAS NESSA AULA

```
const  min_r=0;  max_r=50;
       min_i=1;  max_i=10;

type  vetor_r= array  [min_r..max_r]  of  real;
      vetor_i= array  [min_i..max_i]  of  integer;
```

SOMA DE VETORES

	1	2	3	4	5	6
v:	2	6	1	5	3	3
	+	+	+	+	+	+
w:	1	3	2	4	3	5
	=	=	=	=	=	=
v+w:	3	9	3	9	6	8

```
procedure somar_vetores (var v, w, soma_v_w: vetor_i; tam: integer);
(* este procedimento considera que os dois vetores tem o mesmo tamanho *)
var i: integer;

begin
      for i:= 1 to tam do
            soma_v_w[i]:= v[i] + w[i];
end;
```

- ***v*** e ***w*** poderiam ser passados por **valor**
- ***v*** e ***w*** poderiam ter tamanhos diferentes (e a procedure verifica se tem tamanhos iguais)

```
procedure soma_vetores (var v: vetor_i; tam_v: integer;
                        var w: vetor_i; tam_w: integer;
                        var soma_v_w: vetor_i; tam_soma: integer);
```

PRODUTO ESCALAR

	1	2	3	4	5	6
v:	2	6	1	0	3	3
	×	×	×	×	×	×
w:	1	0	2	4	3	5
	=	=	=	=	=	=
	2	0	2	0	9	15

Depois, somar os números: $2 + 0 + 2 + 0 + 9 + 15 = \mathbf{28}$

```
function prod_escalar (var v, w: vetor_r; tam: integer): real;
var i: integer;
    soma: real;
begin
      soma:= 0;
      for i:= 1 to tam do
            soma:= soma + v[i] * w[i];
      prod_escalar:= soma;
end;
```

BUSCANDO POR UM ELEMENTO

Problema: saber se um elemento ***x*** está dentro de um conjunto ***S***

- Esse é um problema **central** da computação
  - **Aplicações:** busca por livros, busca por cpf, busca por arquivos, ...
- Buscas ainda aparecerão em Alg2, Alg3, Grafos, Análise de Algoritmos, ...

Alg1 - **busca em vetor:** saber se ***x*** está dentro de um vetor ***v***

Pois vetor é a única **estrutura dados** que conhecemos até agora...

Problema: dado ***x*** real, e vetor ***v*** de reais, fazer uma função que devolve a posição onde ***x*** está em ***v***, ou 0 se não estiver presente.

```
function busca_simples (x: real; var v: vetor_r; n: integer): integer;
var i: integer;
begin
      busca_simples:= 0;
      for i:= 1 to n do
            if v[i] = x then
                  busca_simples:= i;
end;
```

MELHORIA: PARAR QUANDO ENCONTRAR

```
function busca_simples_v2 (x: real; var v: vetor_r; n: integer): integer;
var i: integer;
    achou: boolean;

begin
      achou:= false;
      i:= 1;
      while (i <= n) and not achou do
      begin
            if v[i] = x then
                  achou:= true;
            i:= i + 1;
      end;
      if achou then
            busca_simples_v2:= i - 1;
end;
```

Qual a melhoria no desempenho?

Postura: otimista, pessimista ou realista?

MELHORIA: SENTINELA

Ideia: colocar x no final, evitando algumas comparações no laço

```
function busca_com_sentinela (x: real; var v: vetor_r; n: integer): integer;
var i: integer;

begin
      v[n+1]:= x;
      i:= 1;
      while v[i] <> x do
            i:= i + 1;
      if i <= n then
            busca_com_sentinela:= i
      else
            busca_com_sentinela:= 0;
end;
```

dá pra fazer melhor???

BUSCA – RESUMO ATÉ AGORA

busca em vetor: *x* está dentro do vetor *v*?

algoritmos vistos até agora:

1) buscar em todas posições
2) buscar até encontrar
3) buscar com sentinela

é possível melhorar?

BUSCA EM VETOR ORDENADO

é possível melhorar se o vetor estiver ordenado!

- ideia: busca enquanto v[i] < x

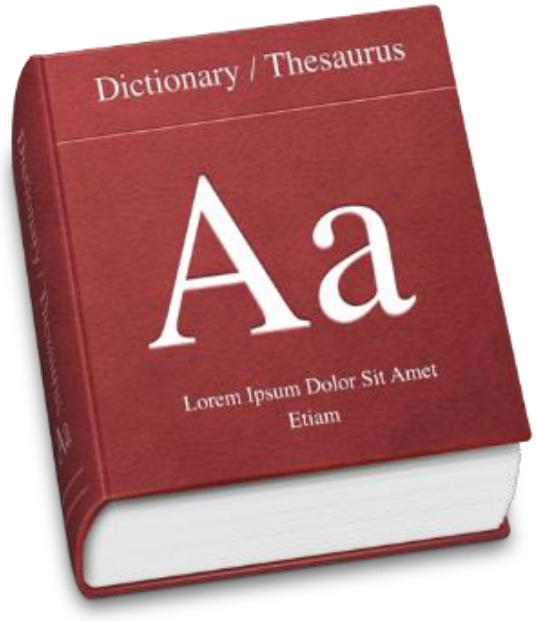
```
function busca_vetor_ordenado (x: real; var v: vetor_r; n: integer): integer;
var i: integer;

begin
      v[n+1]:= x;
      i:= 1;
      while v[i] < x do
            i:= i + 1;
      if (v[i] = x) and (i <= n) then
            busca_vetor_ordenado:= i
      else
            busca_vetor_ordenado:= 0;
end;
```

BUSCA BINÁRIA

ninguém usaria no “mundo real” o algoritmo anterior!

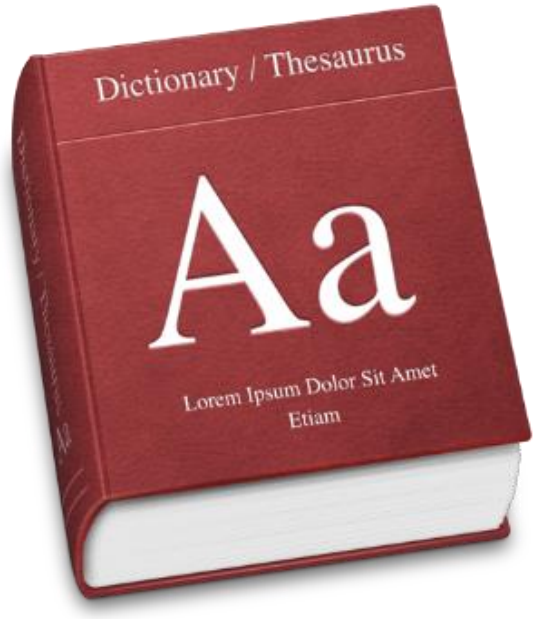
Como você buscaria uma palavra no dicionário?



BUSCA BINÁRIA

ninguém usaria no “mundo real” o algoritmo anterior!

Como você buscaria uma palavra no dicionário?



abre **no meio**:

- se estiver **antes**, procura no 1ª metade
- se estiver **depois**, procura na 2ª metade

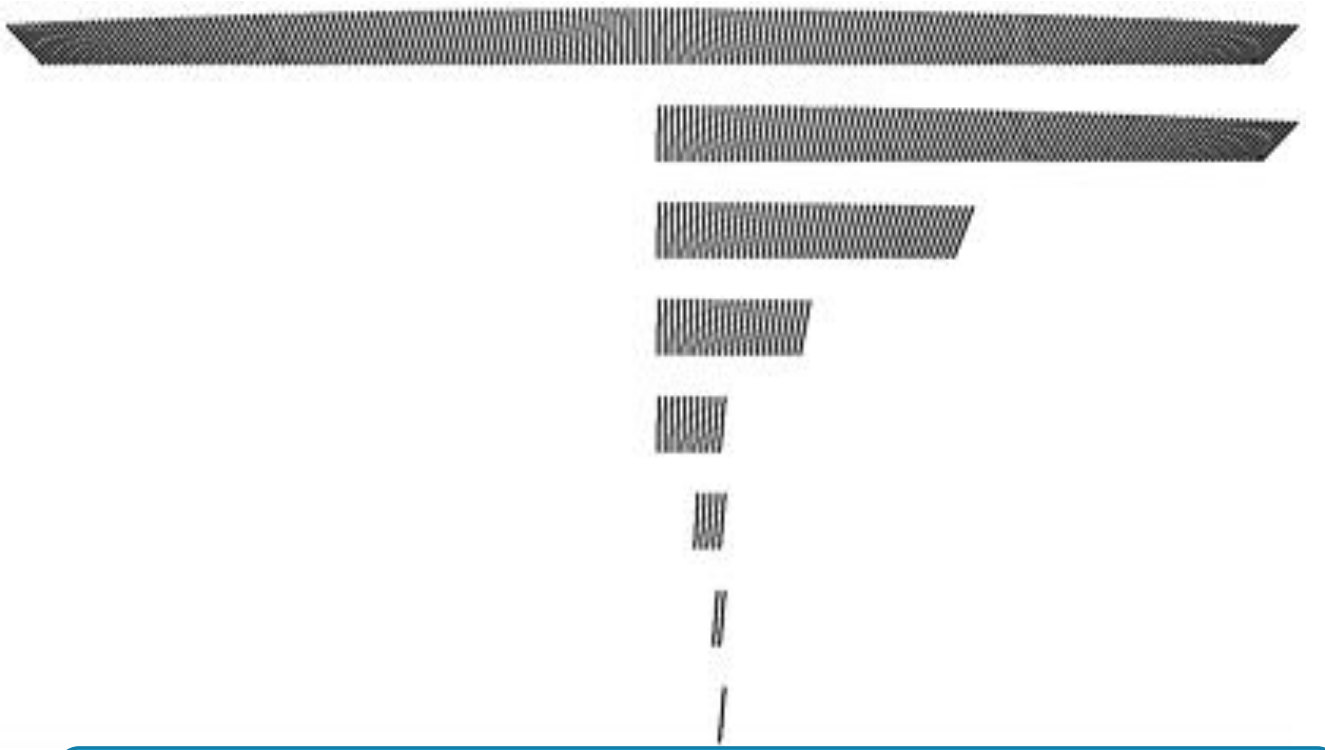
BUSCA BINÁRIA

```
function busca_binaria (x: real; var v: vetor_r; n: integer): integer;
var inicio, fim, meio: integer;

begin
      inicio:=1;                          (* aponta para o inicio do vetor *)
      fim:= n;                            (* aponta para o fim do vetor    *)
      meio:= (inicio + fim) div 2;        (* aponta para o meio do vetor   *)
      while (v[meio] <> x) and (fim >= inicio) do
      begin
            if v[meio] > x then           (* vai jogar uma das duas metades fora  *)
                  fim:= meio - 1          (* metade superior foi jogada fora      *)
            else
                  inicio:= meio + 1;      (* metade inferior foi jogada fora      *)
            meio:= (inicio + fim) div 2;  (* recalcula apontador para meio *)
      end;
      (* o laco termina quando achou ou quando o fim ficou antes do inicio *)
      if v[meio] = x then
            busca_binaria:= meio
      else
            busca_binaria:= 0;
end;
```

TEMPO DE EXECUÇÃO

no **pior caso**, quantas comparações até encontrar um elemento no vetor com **n** páginas?



reformulando: quantas vezes deve-se elevar 2 para que 2^i seja igual a n?

TEMPO DE EXECUÇÃO

comparações no pior caso:

- busca sequencial: n
- busca binária: $\log_2 n$

moral da história: algoritmos inteligentes são muito mais rápidos!

se soubesse disso, não testaria as lâmpadas uma a uma para descobrir onde está a bomba...

n	$\log_2 n$
2	1
4	2
8	3
16	4
32	5
...	...
1024	10
...	...
1 milhão	$\approx$ 20
...	...
1 bilhão	$\approx$ 30

COMPARANDO

Versão	$n = 10$	$n = 10^2$	$n = 10^4$	$n = 10^8$
Busca simples (fig. 10.16)	20	200	20000	200000000
Busca com sentinela (fig. 10.17)	10	100	10000	100000000
Busca em vetor ordenado (fig. 10.18)	10	100	10000	100000000
Busca binária (fig. 10.20)	3	5	10	19

vale a pena ordenar o vetor?