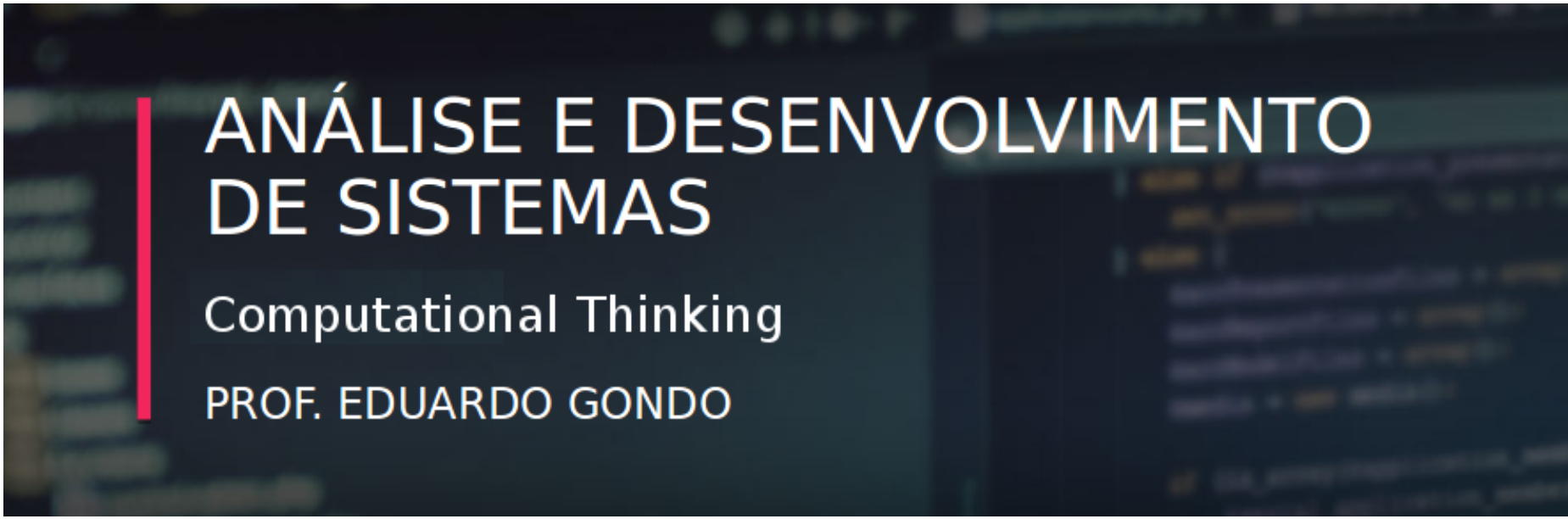
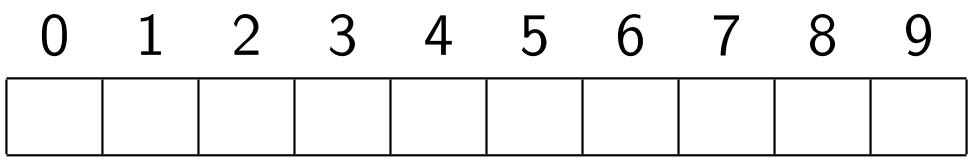

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

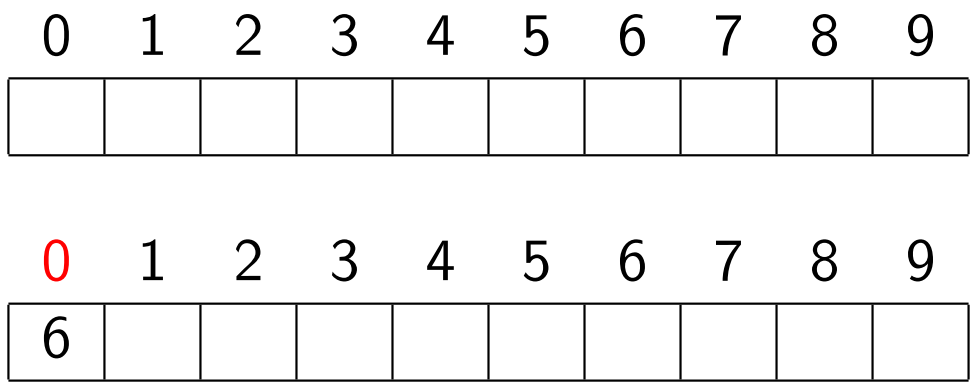
Busca simples - problema

Problema: dado um vetor *vet* de números reais e um número real x qualquer, devolva uma posição de x em *vet* se $x \in vet$ ou -1 se $x \notin vet$.
Vamos simular a busca no vetor abaixo procurando o número 14:



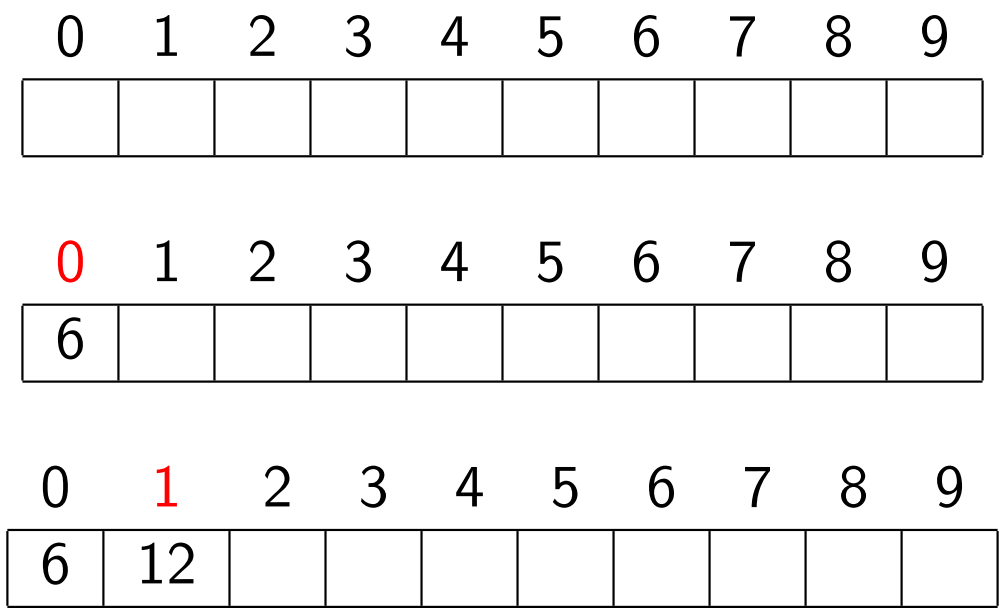
Busca simples - problema

Problema: dado um vetor *vet* de números reais e um número real x qualquer, devolva uma posição de x em *vet* se $x \in vet$ ou -1 se $x \notin vet$.
Vamos simular a busca no vetor abaixo procurando o número 14:



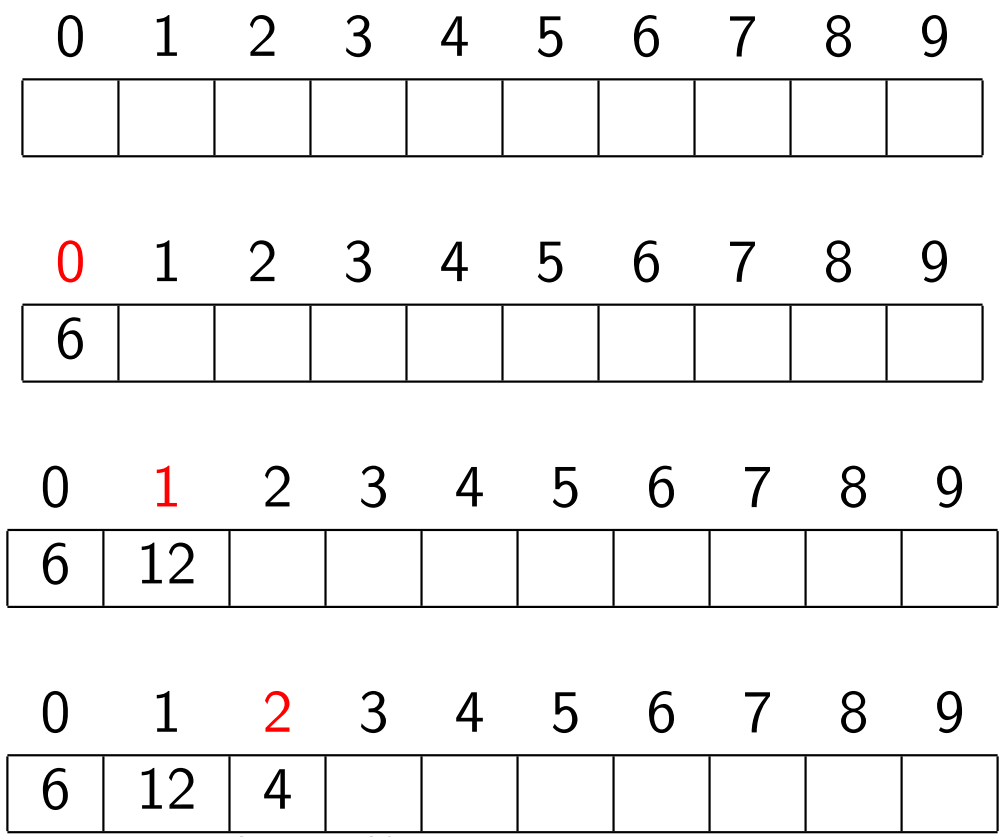
Busca simples - problema

Problema: dado um vetor *vet* de números reais e um número real x qualquer, devolva uma posição de x em *vet* se $x \in vet$ ou -1 se $x \notin vet$.
Vamos simular a busca no vetor abaixo procurando o número 14:



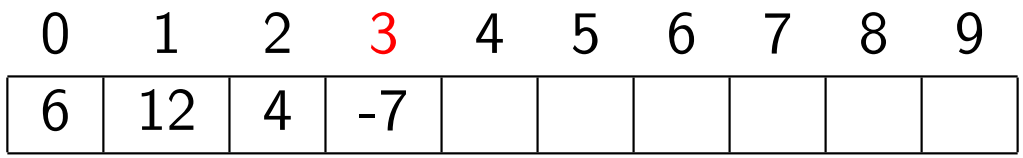
Busca simples - problema

Problema: dado um vetor *vet* de números reais e um número real x qualquer, devolva uma posição de x em *vet* se $x \in vet$ ou -1 se $x \notin vet$.
Vamos simular a busca no vetor abaixo procurando o número 14:



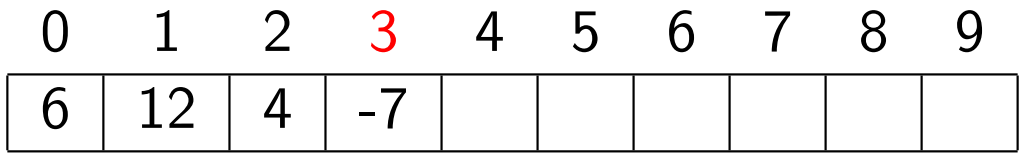
Busca simples - problema

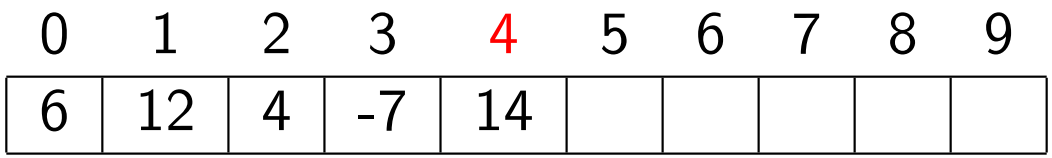
Continuando a busca do 14:



Busca simples - problema

Continuando a busca do 14:





Neste ponto a repetição pára e retorna a posição 4 do vetor que é a posição que contém a primeira ocorrência do 14 no vetor.

Vejamos duas implementações do algoritmo de busca usando while e depois usando o for.

Busca simples - algoritmo

Veja abaixo um método que faz a busca dos dados, note que ele recebe o vetor de números reais e o número *x* a ser procurado. O método retorna a posição da primeira ocorrência de *x* em *vet*.

```
1  public int busca(double[] vet, double x) {
2       int i = 0;
3       while (i < vet.length && vet[i] != x)
4            i++;
5       if (i == vet.length)
6            return -1;
7       else
8            return i;
9  }
```

Busca simples - Versão com for

```
1  public int busca(double[] vet, double x) {
2      for(int i = 0; i < vet.length; i++) {
3          if (vet[i] == x)
4              return i;
5      }
6      return -1;
7  }
```

Ambas as soluções são equivalentes em termos de performance, pois no for quando encontramos o elemento x o método termina.

Busca - considerações

- note que estamos interessados na posição de x em *vet*
- o valor x pode ocorrer mais de uma vez no vetor *vet*, mas nosso método retorna apenas uma posição
- o pior caso, ou seja, a situação que o algoritmo vai levar mais tempo de processamento é quando o valor x não pertence à *vet*
- nesse caso procuramos em todo vetor a procura da informação e não encontramos
- o tempo de execução de pior caso é proporcional ao tamanho do vetor
- cada comparação de um elemento do vetor com x elimina apenas um elemento
- será que pode ser melhor, ou seja, cada comparação eliminar vários elementos do vetor?

Busca - busca binária

- a resposta é sim! MAS...

Busca - busca binária

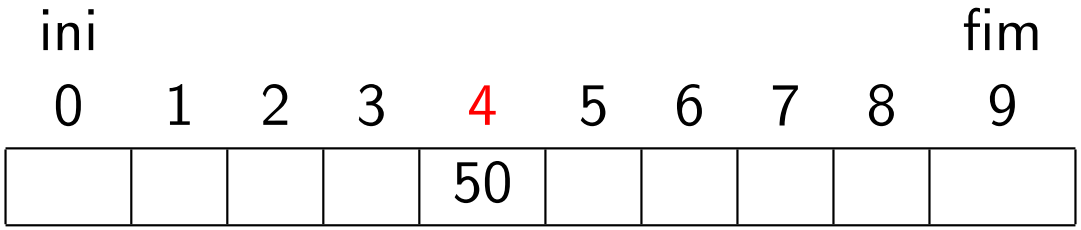
- a resposta é sim! MAS...
- apenas se os dados do vetor estiverem ordenados, daí podemos consultar como uma lista telefônica
- a idéia principal do algoritmo é com uma comparação eliminar grande parte dos dados do vetor

chamando o elemento procurado de x e o vetor de vet, podemos descrever o algoritmo de busca binária do seguinte modo:

```
1   enquanto existir elementos em vet faça {
2     pegue o elemento central do vetor vet, seja t este
          elemento
3     se t > x então
4         elimine todos os elementos do vetor vet após a
              posição central
5     se t < x então
6         elimine todos os elementos do vetor vet antes da
              posição central
7     se t == x então
8         retorne a posição de t
9   } //fim enquanto
```

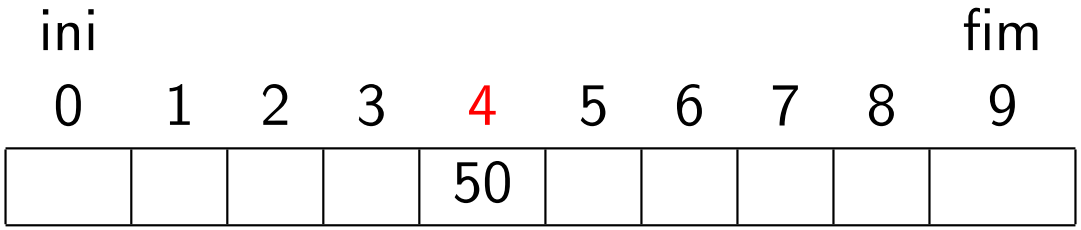
Busca binária - simulação

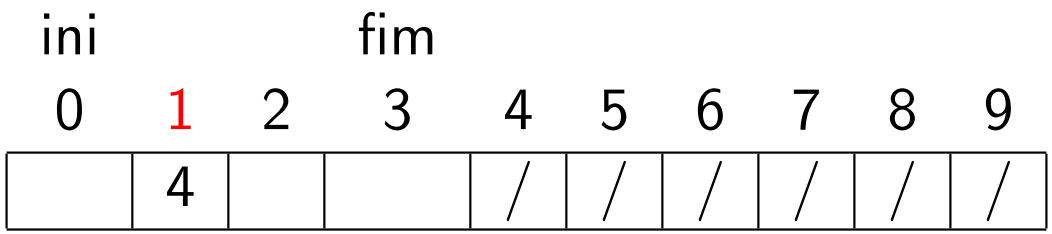
Continuando a busca do 14:



Busca binária - simulação

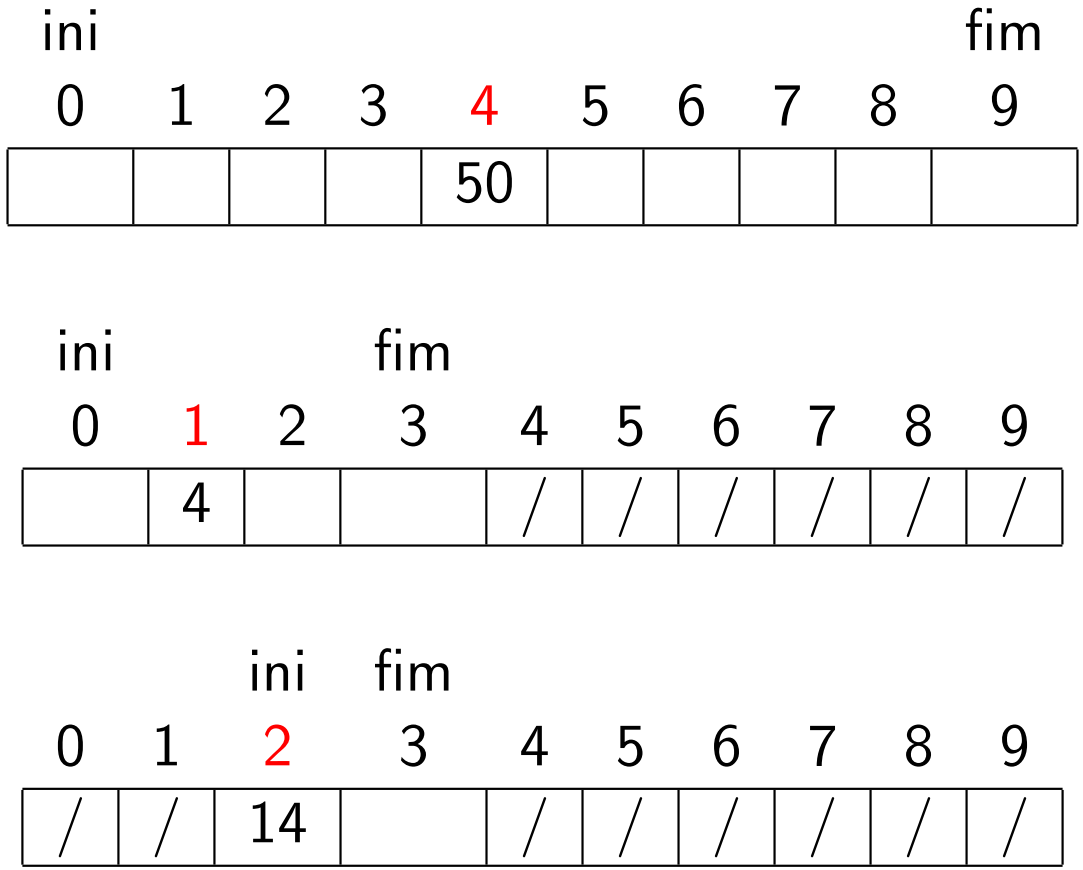
Continuando a busca do 14:





Busca binária - simulação

Continuando a busca do 14:



Busca - busca binária

Segue abaixo a implementação do busca binária em Java

```
1
2  public int buscaBinaria(double[] vet, double x) {
3      int inicio = 0;
4      int fim = vet.length - 1;
5      while (inicio <= fim) {
6          int meio = (inicio + fim) / 2; //meio do vetor
7          if (vet[meio] > x)
8                fim = meio - 1;
9          else if (vet[meio] < x)
10               inicio = meio + 1;
11         else
12               return meio;
13     }
14     return -1;
15 }
```

Busca binária - observações

vejamos algumas considerações sobre a implementação

- crie apontadores para marcar as extremidades dos vetor que vocês estão considerando
- como sugestão indico os inteiros inicio e fim
- no começo do algoritmo inicio = 0 e fim = v.length - 1
- para obter o elemento central faça a média entre inicio e fim
- se inicio <= fim então ainda tenho elementos para examinar
- será que inicio > fim em algum momento do meu algoritmo?

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Deitel e Deitel - Java como Programar
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados