

Busca

Prof. Denio Duarte

Prof. Geomar Schreiner

Busca

- Uma busca consiste em **recuperar** um ou mais itens armazenados em um repositório de dados.
- Sempre buscamos os dados da mesma forma?

Busca

- Uma busca consiste em **recuperar** um ou mais itens armazenados em um repositório de dados.
- Sempre buscamos os dados da mesma forma?
 - Depende!

Busca

- Uma busca consiste em **recuperar** um ou mais itens armazenados em um repositório de dados.
- Depende:
 - De como os dados estão estruturados
 - Vetor, lista, árvore, arquivo
 - Se os dados estão ou não ordenados
 - Se há duplicidade de chaves

Busca Linear

- Método mais simples de pesquisa
- Varredura serial do conjunto de dados, da primeira até a última posição, comparando a chave de pesquisa com a chave de cada entrada
 - pesquisa **bem-sucedida**: é encontrada uma chave igual
 - pesquisa **malsucedida**: o final da lista é atingido sem que a chave procurada seja encontrada.

Busca Linear

- Método mais simples de pesquisa
- Bem-sucedida:
 - O valor do elemento encontrado; ou
 - o índice (vetor) ou endereço (ponteiro) do elemento
- Qual seria a complexidade de tempo na ordem O (pior caso)?

Busca Linear

- Método mais simples de pesquisa
- Bem-sucedida:
 - O valor do elemento encontrado; ou
 - o índice (vetor) ou endereço (ponteiro) do elemento
- Qual seria a complexidade de tempo na ordem O (pior caso)?
 - $O(n)$ – n é o tamanho da lista de entrada

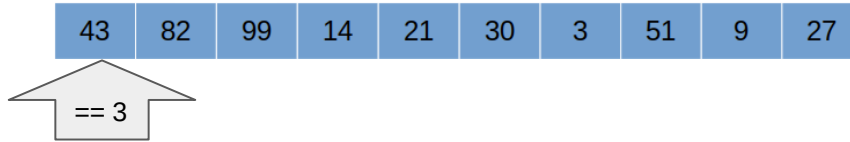
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave de busca**)

43	82	99	14	21	30	3	51	9	27
----	----	----	----	----	----	---	----	---	----

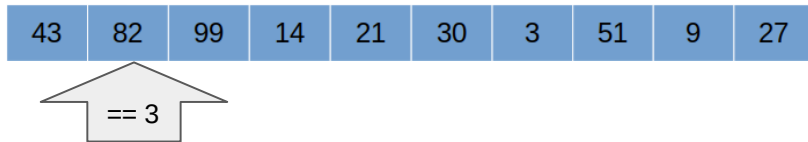
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)



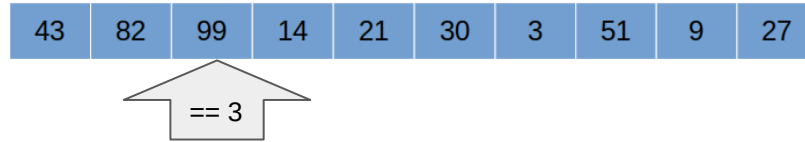
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (*chave*)



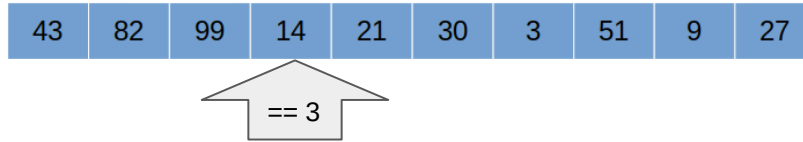
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)



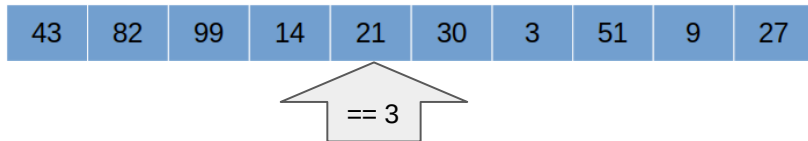
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)



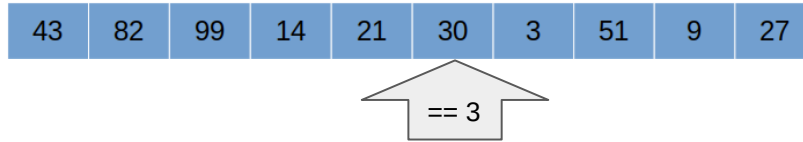
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)



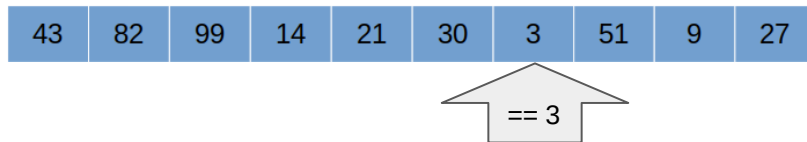
Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)



Busca Linear

- Exemplo: vamos considerar primeiramente um vetor
 - Buscar o valor 3 (**chave**)

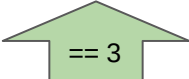


Busca Linear

- Exemplo: vamos considerar primeiramente um vetor

- Buscar o valor 3 (**chave**)

43	82	99	14	21	30	3	51	9	27
----	----	----	----	----	----	---	----	---	----



- Retorna 3 ou a posição (6)

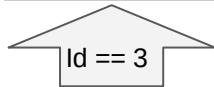


A função de busca, em caso de malsucedida,
deve retornar um valor que indique o insucesso,
por exemplo, -1

Busca Linear

- E se fosse um vetor de struct
 - Buscar Funcionário com **Id = 3**

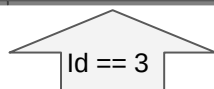
Id: 1 Nome: Andrei Salario: 340.00	Id: 2 Nome: Geomar Salario: 340.00	Id: 6 Nome: Marinho Salario: 360.00
--	--	---



Busca Linear

- E se fosse um vetor de struct
 - Buscar Funcionário com **Id = 3**

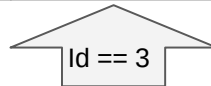
Id: 1 Nome: Andrei Salario: 340.00	Id: 2 Nome: Geomar Salario: 340.00	Id: 6 Nome: Marinho Salario: 360.00
--	--	---



Busca Linear

- E se fosse um vetor de struct
 - Buscar Funcionário com **Id = 3**

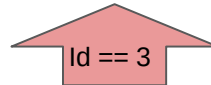
Id: 1 Nome: Andrei Salario: 340.00	Id: 2 Nome: Geomar Salario: 340.00	Id: 6 Nome: Marinho Salario: 360.00
--	--	---



Busca Linear

- E se fosse um vetor de struct
 - Buscar Funcionário com **Id = 3**

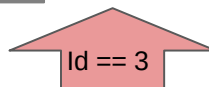
Id: 1 Nome: Andrei Salario: 340.00	Id: 2 Nome: Geomar Salario: 340.00	Id: 6 Nome: Marinho Salario: 360.00
--	--	---



Busca Linear

- E se fosse um vetor de struct
 - Buscar Funcionário com **Id = 3**

Id: 1 Nome: Andrei Salario: 340.00	Id: 2 Nome: Geomar Salario: 340.00	Id: 6 Nome: Marinho Salario: 360.00
--	--	---

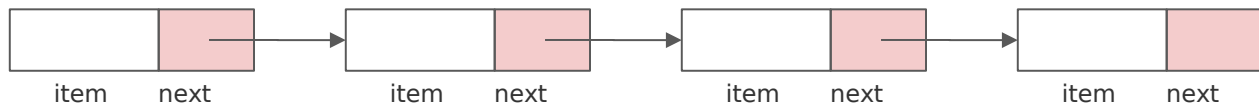


Pesquisa malsucedida.
Nem toda pesquisa
precisa retornar algo.

A função de busca, em caso de **malsucedida**,
deve retornar um valor que indique o insucesso,
por exemplo, **-1**

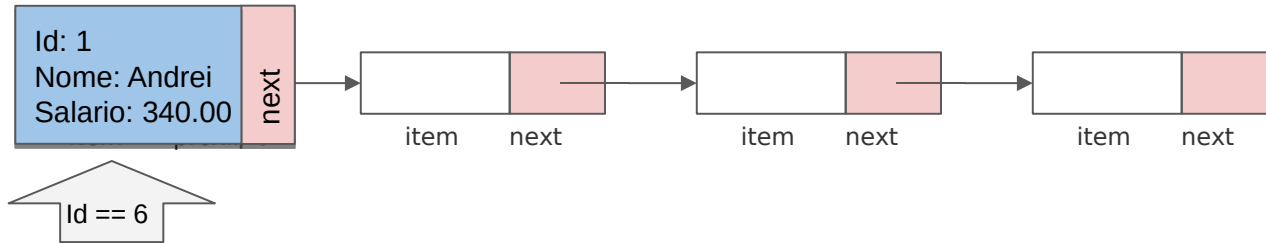
Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



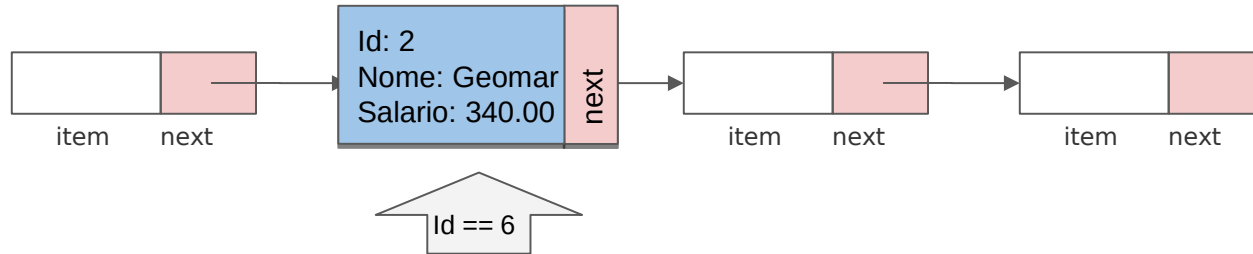
Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



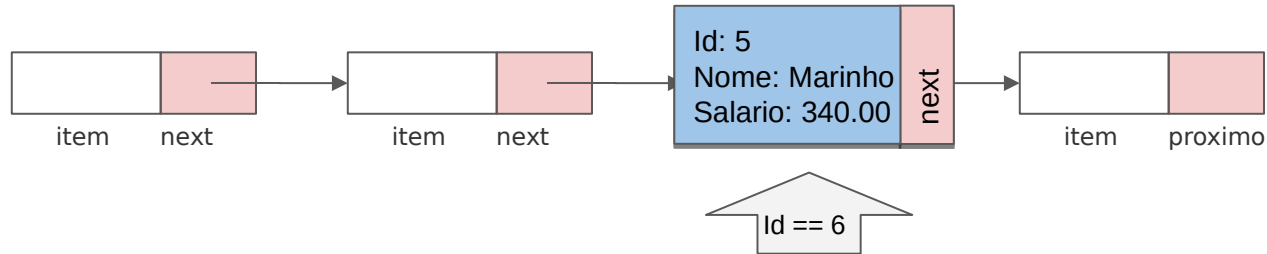
Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



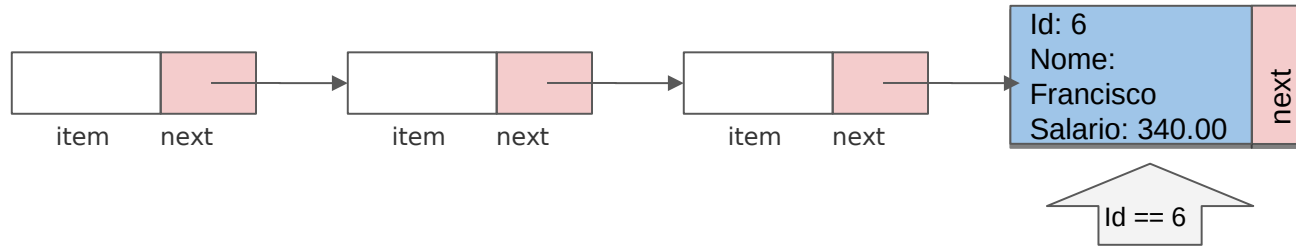
Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



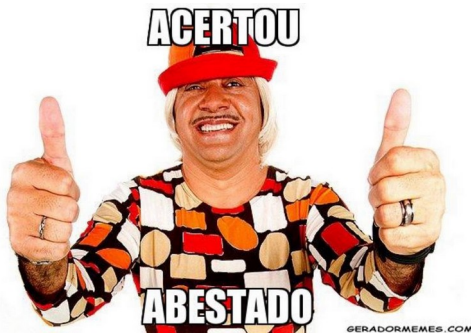
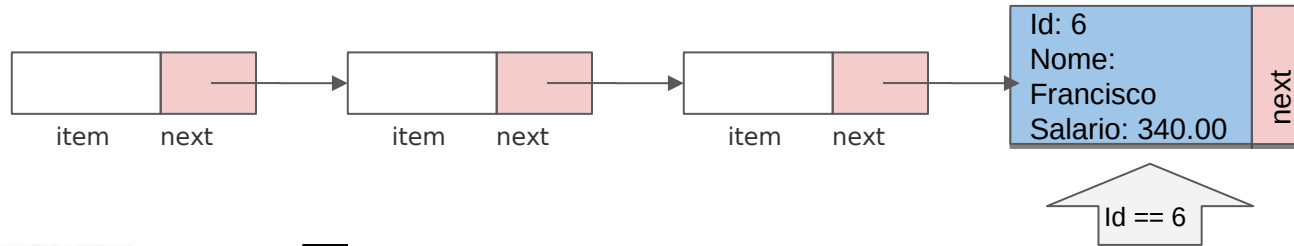
Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



Busca Linear

- E se fosse uma lista encadeada
 - Buscar Funcionário com **id = 6** (chave de busca)



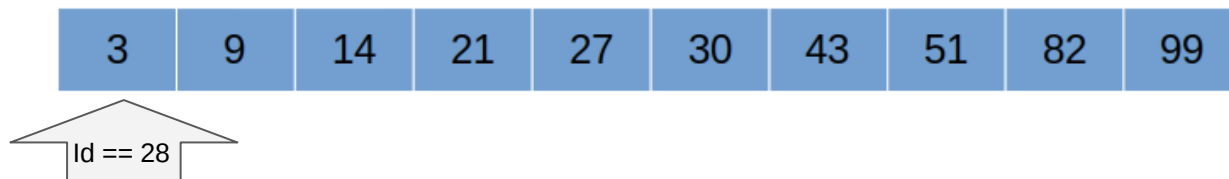
Retorna os dados da estrutura, o ponteiro ou **NULL** no caso de **malsucedida**.

Busca Linear

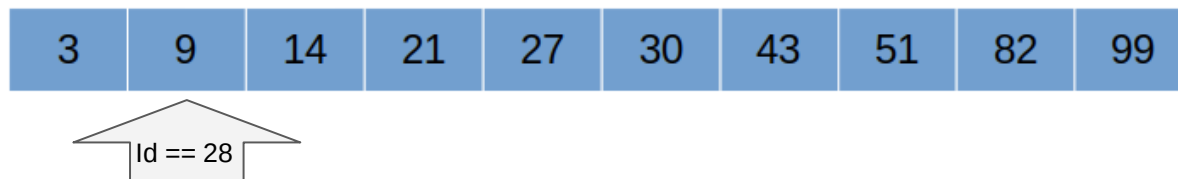
- Esse método funciona para qualquer tipo de estrutura
 - Simples e robusto
 - Oneroso pois passa por todos os elementos – $O(n)$
 - Se os valores da estrutura estiverem ordenados conseguiríamos otimizar?

3	9	14	21	27	30	43	51	82	99
0	1	2	3	4	5	6	7	8	9

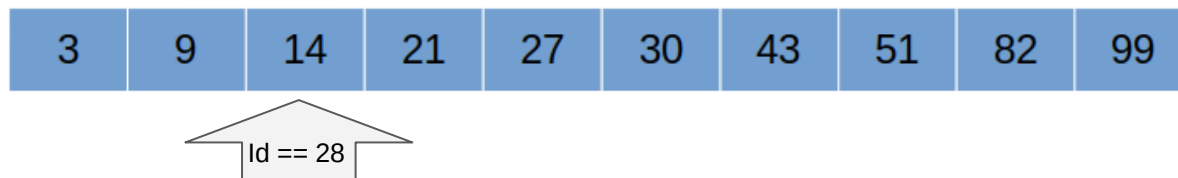
Busca Linear



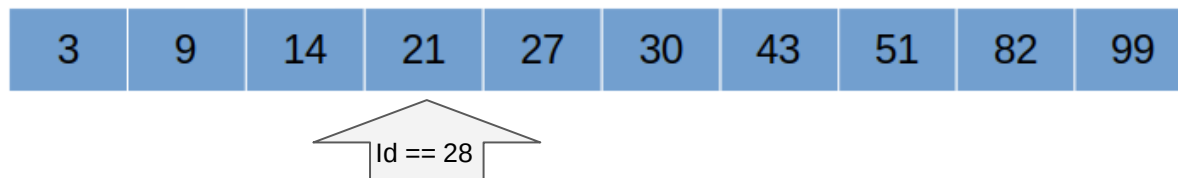
Busca Linear



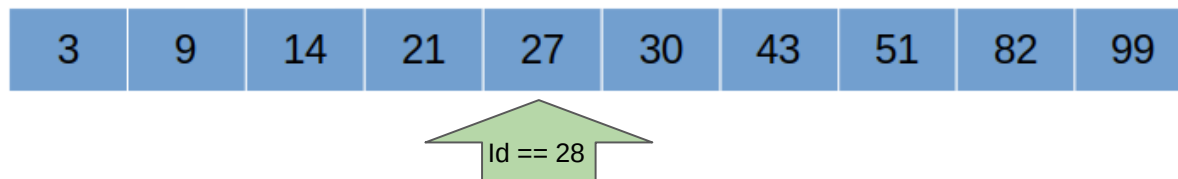
Busca Linear



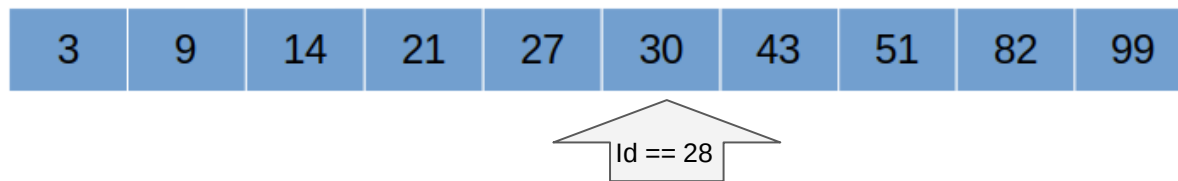
Busca Linear



Busca Linear

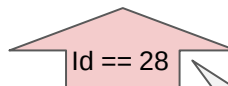


Busca Linear



Busca Linear

3	9	14	21	27	30	43	51	82	99
---	---	----	----	----	----	----	----	----	----



Como o valor agora
é maior do que eu
procuro, posso
parar a varredura

Busca Binária

- Considerando um vetor **pré ordenado**, podemos tirar vantagem para otimizar o código
- Num vetor ordenado, podemos adotar uma estratégia mais sofisticada e eficiente: **busca binária**
- Divisão e conquista: a cada passo, analisa apenas parte do problema.

Busca Binária

- Divisão e conquista: a cada passo, analisa apenas parte do problema.
 - Caso 1. O elemento do meio corresponde à chave procurada
 - a busca termina com sucesso.
 - Caso 2. A chave buscada é menor do que o elemento do meio
 - a busca continua na primeira metade do vetor.
 - Caso 3: A chave buscada é maior do que o elemento do meio
 - a busca continua na segunda metade do vetor.
 - Repete até encontra ou não a chave

Busca Binária

- Buscar pelo elemento 43

3	9	14	21	27	30	43	51	82	99
0	1	2	3	4	5	6	7	8	9
inicio									fim

$$\text{meio} = (\text{fim} - \text{inicio}) / 2 \Rightarrow (9 - 0) / 2 = 4.5 \Rightarrow 4$$

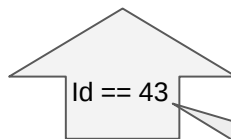
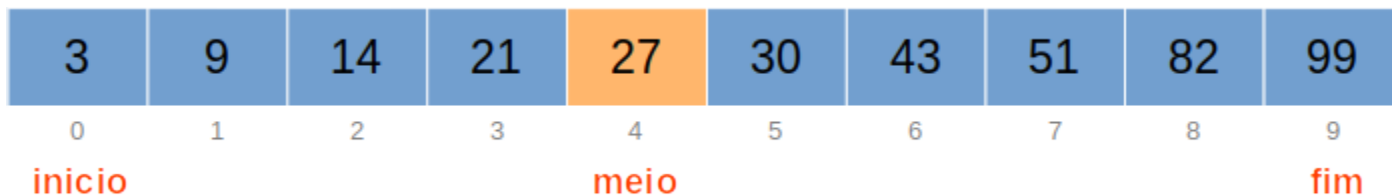
Busca Binária

- Buscar pelo elemento 43

3	9	14	21	27	30	43	51	82	99
0	1	2	3	4	5	6	7	8	9
inicio				meio					fim

Busca Binária

- Buscar pelo elemento 43



$43 > 27$

Então ignoramos o que está a esquerda do meio, e chamamos a função para a parte da direita.

Busca Binária

- Buscar pelo elemento 43



Id == 43

$43 < 51$
Então ignoramos o que está à direita do meio, e chamamos a função para a parte da esquerda.

Busca Binária

- Buscar pelo elemento 43



Busca Binária

- Buscar pelo elemento 43



início
meio

fim

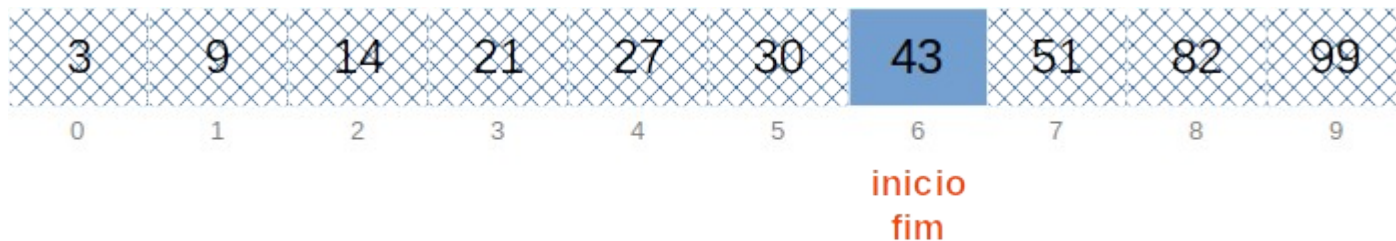


$43 > 30$

Então ignoramos o que está a esquerda do meio, e chamamos a função para a parte da direita.

Busca Binária

- Buscar pelo elemento 43

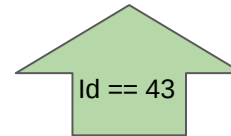


Busca Binária

- Buscar pelo elemento 43



inicio
fim
meio



FINALMENTE

0'43'



Busca Binária

- É um algoritmo relativamente simples
 - Utiliza o paradigma dividir para conquistar
- Possui um desempenho superior a busca linear
 - Como a lista é sempre dividida por dois, ou seja, $n/2/2/2\dots$ até a divisão retornar 1
 - A complexidade de tempo é $O(\log_2 n)$ (lembre busca linear $O(n)$)

Busca Binária

- Requisito:
 - Depende de um **vetor ordenado** para funcionar
 - O vetor pode ser de qualquer tipo
 - É **possível** implementá-lo em uma **lista encadeada** mas perde eficiência
- Implementação mais usual utiliza **recursividade**

Busca Binária

- Algoritmo Iterativo

```
int buscaBinaria(int *l, int NITEMS, int key){
    int inicio=0, fim=NITEMS, meio=(inicio+fim)/2;
    while (inicio<=fim)
    {
        if (l[meio]==key)
            break;
        if (l[meio]<key)
            inicio=meio+1;
        else
            fim=meio-1;
        meio=(inicio+fim)/2;
    }
    if (inicio > fim)
        return -1;
    else
        return l[meio];
}
```

Busca Binária

- Algoritmo Recursivo

```
int buscaBinaria (int *vet, int inicio, int fim, int chave){  
    int meio;  
    if (inicio > fim)  
        return -1;  
    meio = (inicio+fim)/2;  
  
    if (vet[meio] == chave) {  
        return meio;  
    }  
  
    if (chave > vet[meio] )  
        return buscaBinaria(vet, meio+1, fim, chave);  
    else  
        return buscaBinaria(vet, inicio, meio-1, chave);  
}
```