

---

# **Pesquisa de Dados em Tabelas ou Busca**

Pesquisa sequencial e pesquisa binária

# Pesquisa de Dados em Tabelas

- Métodos para localizar entradas em tabelas, dado o valor de uma chave primária como argumento de pesquisa

	Chave Primária	Info
1	7	...
2	9	...
3	14	...
4	35	...
5	78	...

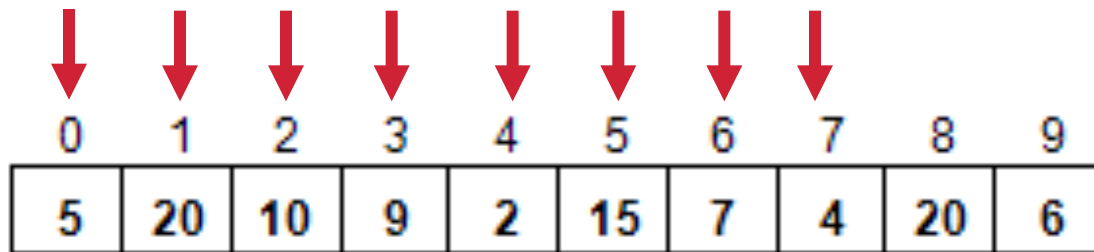
- Localizar: informações relativa às chaves: 35, 12

- ◆ Fazer uma varredura serial de um array, comparando o argumento de pesquisa com o elemento de cada posição, até ser encontrado um que seja igual (sucesso) ou até que seja atingido o final do array (não foi encontrado).

## Pesquisa Sequencial em Array não ordenado

```
public int pesquisaSequencial(int[] tab, int arg) {  
    for (int i = 0; i < tab.length; i++)  
        if (tab[i] == arg)  
            return i;  
    return -1;  
}
```

Exemplo: procurando o elemento 4



0	1	2	3	4	5	6	7	8	9
5	20	10	9	2	15	7	4	20	6

## Pesquisa Sequencial em Array Ordenado

```
public int pesquisaSequencialOrdenada(int[] tab, int arg) {  
    for (int i = 0; (i < tab.length) && (tab[i] <= arg); i++)  
        if (tab[i] == arg)  
            return i;  
    return -1;  
}
```

Exemplo: procurando o elemento 11

0	1	2	3	4	5	6	7	8	9
2	4	5	7	9	10	12	13	15	16

# Pesquisa Binária

- Método para ser aplicado em **arrays ordenados**
- Reduz o nro. de elementos a serem considerados pela metade
- Exemplo: Pesquisa do elemento com valor 17

0	1	2	3	4	5	6	7	8	9	10
1	3	5	7	9	11	13	15	17	19	21

Diagram illustrating binary search on a sorted array. The array contains values [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21] at indices 0 to 10. The search target is 17. The first comparison (1) is at index 5 (value 11). The second comparison (2) is at index 8 (value 17).

# Pesquisa Binária

- Técnica
  - Consiste na comparação do argumento de pesquisa (*arg*) com o elemento localizado no endereço médio da tabela.
  - Se *arg* for maior do que o elemento contido naquele endereço, o processo é repetido para a metade superior da tabela;
  - se for menor, para a metade inferior;
  - se for igual, a busca se encerra com sucesso.
- A área de pesquisa é reduzida à metade do número de elementos a cada vez

# Uma pesquisa binária

**Argumento de pesquisa: 32**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	15	17	30	32	34	40	50	80	90	95	97	99	101	105
			↑	↑	↑		↑							
			2 <sup>a</sup>	4 <sup>a</sup>	3 <sup>a</sup>		1 <sup>a</sup>							



## Outra pesquisa

Argumento de pesquisa: 54

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200
	↑	↑	↑	↑					
	2 <sup>a</sup>	3 <sup>a</sup>	4 <sup>a</sup>	1 <sup>a</sup>					

# Mais uma pesquisa

**Argumento de pesquisa: 100**

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200
				↑			↑	↑	
				1 <sup>a</sup>			2 <sup>a</sup>	3 <sup>a</sup>	

# Algoritmo Pesquisa Binária

- Localizar, por busca binária, a posição ocupada pelo elemento de valor *arg* em um vetor *tab*, ordenado

**Parâmetros:**

*tab*: array onde será feita a pesquisa

*arg*: argumento de pesquisa

**Retorno :**    **-1**   não encontrou

**≠ -1**   elemento está na posição

# Algoritmo Pesquisa Binária

```
public int pesquisaBinaria(int[] tab, int arg) {  
    int inf = 0;  int sup = tab.length - 1;  
    while (inf <= sup) {  
        int med = (inf + sup) / 2; // divisão inteira  
        if (arg == tab[med])  
            return med;  
        else if (arg < tab[med])  
            sup = med - 1; // procura na 1a. metade  
        else  
            inf = med + 1; // procura na 2a. metade  
    }  
    return -1;  
}
```

**Exemplo:**  
busca do  
elemento 43

**inf sup**

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200

↑  
**3º**