Estrutura de Dados

Aula 13

Pesquisa (busca) e Ordenação (classificação)

Pesquisa de Dados

Dado um conjunto de elementos identificados por uma chave, localizar o elemento que corresponde a uma chave dada.

Ex: Busca em uma lista de funcionários os dados do funcionário que possui código igual a 1432

- Cada elemento do conjunto é chamado de **registro**;
- Cada registro é subdividido em **campos**;
- Existe uma **chave** associada a cada registro usada para diferenciar os registros entre si.

0	102	Rosa
1	107	Ana
2	95	Clara
3	120	Bia
4	122	Maria
5	155	Joana
6	115	Paula
7	99	Amélia
8	100	Bruna
9	104	Alice

Pesquisa de Dados

Objetivo: Localizar um dado elemento com o menor **custo**

Tipos de Pesquisa (busca)

1. Pesquisa sequencial
2. Pesquisa binária
3. Pesquisa por meio de árvore de busca binária

1. Pesquisa Sequencial

Percorrer registro por registro em busca da chave. A pesquisa termina quando o elemento for encontrado ou quando todos os registros foram acessados.

```
V: Vetor que contém os dados
Chave: dado consultado
N :Quantidade de elementos do vetor

Início
| Para indice=0 até N-1 faça
| | Se V[indice].chave = Chave então
| | |   Retorna indice    /* Chave encontrada */
| | FimSe
| FimPara
| Retorna -1          /* Chave não encontrada */
Fim
```

- Quantas comparações para achar o código 120? **4**
- Quantas comparações para determinar que uma chave não existe? **10 (n)**
- Quantas comparações no melhor caso? **1**

0	102	Rosa
1	107	Ana
2	95	Clara
3	120	Bia
4	122	Maria
5	155	Joana
6	115	Paula
7	99	Amélia
8	100	Bruna
9	104	Alice

- Em média são executadas **(n+1)/2** comparações

O (n)

Notação O (Big O)

Medida de eficiência de um algoritmo. Relaciona a quantidade de operações necessárias para executá-lo e a quantidade de elementos que ele manipula.

Ex:

Método: **incluiNoInicio**

Quantas operações são executadas para uma lista 10 nodos?

E para uma lista de **n** nodos?

Linear: **O(n)**

Método: **consulta** (por posição)

Quantas operações são executadas para uma lista 10 nodos?

E para uma lista de **n** nodos?

Constante: **O(1)**

0	102	Rosa
1	107	Ana
2	95	Clara
3	120	Bia
4	122	Maria
5	155	Joana
6	115	Paula
7	99	Amélia
8	100	Bruna
9	104	Alice

Notação O (Big O)

		n		
		10	100	1000
constante	$O(1)$	1	1	1
logaritmica	$O(\log_2 n)$	3	7	10
linear	$O(n)$	10	100	1000
logarímica linear	$O(n \log_2 n)$	33	664	9970
Polinomial $O(n^k)$	$O(n^2)$	100	10.000	1.000.000
	$O(n^3)$	1000	1.000.000	1.000.000.000
Exponencial	$O(2^n)$	1024	$1,268x10^{30}$	$1,072x10^{301}$
Fatorial	$O(n!)$	368800	100!	1000!

Pesquisa Binária

- Método aplicado a um conjunto de dados ordenados pela chave de busca;
- Exige acesso aleatório;
- O (log_2 n)

Comparar o elemento buscado com o elemento central do vetor

Se for igual achou
Se for menor buscar na metade inferior do vetor
Se for maior buscar na metade superior do vetor
Calcular um novo elemento central e repetir a busca para a respectiva metade

0	99	Rosa
1	102	Ana
2	115	Clara
3	120	Bia
4	122	Maria
5	130	Joana
6	135	Paula
7	137	Amélia
8	140	Bruna
9	145	Alice

Pesquisa Binária

Chave: 135

Inicio = 0

Fim = 9

Meio = 4

Inicio	0	99	Rosa
	1	102	Ana
	2	115	Clara
	3	120	Bia
→	4	122	Maria
	5	130	Joana
	6	135	Paula
	7	137	Amélia
	8	140	Bruna
Fim	9	145	Alice

Chave: 135

Inicio = 5

Fim = 6

Meio = 5

	0	99	Rosa
	1	102	Ana
	2	115	Clara
	3	120	Bia
	4	122	Maria
Inicio →	5	130	Joana
Fim	6	135	Paula
	7	137	Amélia
	8	140	Bruna
	9	145	Alice

Chave: 135

Inicio = 5

Fim = 9

Meio = 7

	0	99	Rosa
	1	102	Ana
	2	115	Clara
	3	120	Bia
	4	122	Maria
Inicio	5	130	Joana
	6	135	Paula
→	7	137	Amélia
	8	140	Bruna
Fim	9	145	Alice

Chave: 135

Inicio = 6

fim = 6

Meio = 6;

	0	99	Rosa
	1	102	Ana
	2	115	Clara
	3	120	Bia
	4	122	Maria
	5	130	Joana
Inicio Fim →	6	135	Paula
	7	137	Amélia
	8	140	Bruna
	9	145	Alice

Achou na posição 6

Pesquisa Binária

```
Início
| Enquanto inicio <= fim faça
| | meio = (inicio+fim)/2;
| | Se V[meio].chave = chave então
| | |  Retorna meio;          /* Chave encontrada */
| | Senão
| | |  Se Chave < v[meio].chave então
| | |  |   fim = meio-1
| | |  Senão
| | |  |   inicio = meio+1
| | |  FimSe
| | FimSe
| FimEnquanto
|Retorna -1                  /* Chave não encontrada */
Fim
```

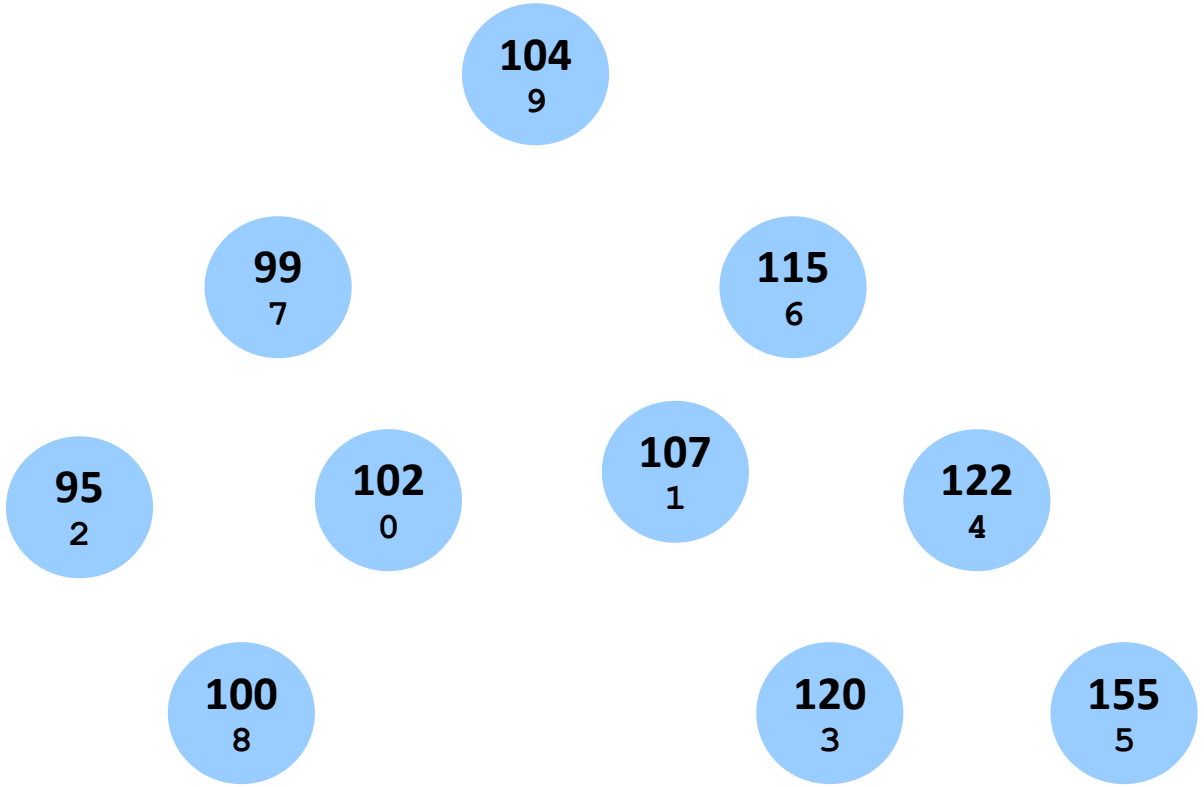
V: Vetor que contém os dados

Chave: dado consultado

inicio :índice do primeiro

Fim: índice do último

Pesquisa por meio de Árvore de Busca Binária



0	102	Rosa
1	107	Ana
2	95	Clara
3	120	Bia
4	122	Maria
5	155	Joana
6	115	Paula
7	99	Amélia
8	100	Bruna
9	104	Alice

Ordenação

Consiste na determinação da ordem dos elementos de um conjunto E de tal forma que:

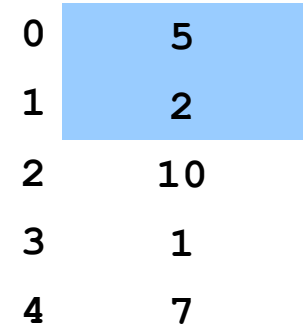
$E_1 < E_2 < \ldots < E_n$ para ordem crescente ou
$E_1 > E_2 > \ldots > E_n$ para ordem decrescente

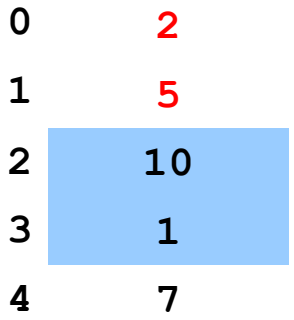
Método da Bolha (*Bubble Sort*)

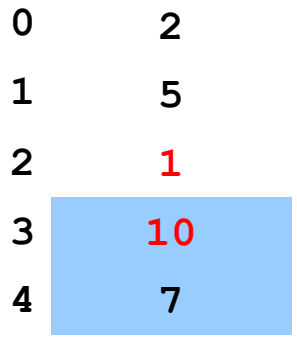
O (n²)

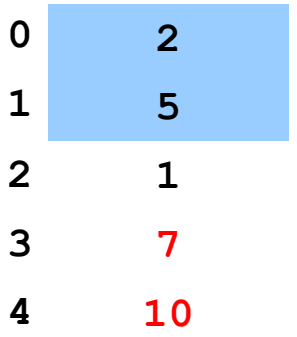
1. Cada elemento é comparado com o próximo
2. Se o elemento estiver fora de ordem a troca é realizada
3. Repetir o passo 1 até que não ocorram trocas

Método da bolha (*Bubble Sort*)

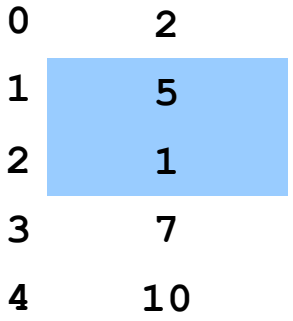


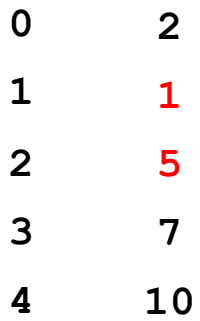




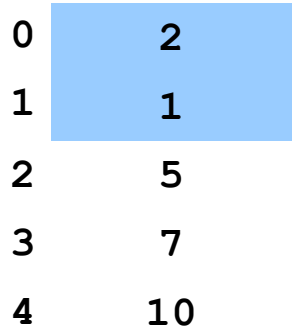


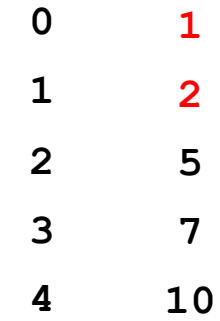
Houve troca?



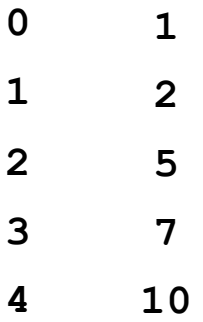


Houve troca?





Houve troca?



Houve troca?

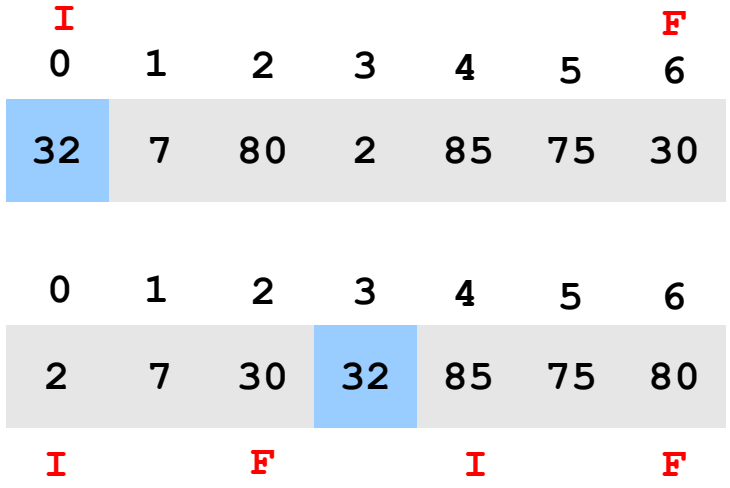
Método da Bolha (*Bubble Sort*)

```
Início
  Faça
  | Assinalar que não houve troca
  | Para cada elemento do vetor (exceto o último)
  | | Se o elemento estiver fora de ordem em
  | | |                    Relação ao próximo então
  | | | Trocar o elemento com o próximo
  | | | Assinalar que houve troca
  | | FimSe
  | FimPara
  Enquanto houver troca
Fim
```

Método da Troca e Partição (Quick *Sort*)

$O(n \log_2 n)$

```
void quickSort(int v[], int inicio, int fim) {
    int indicePivo;
    if (fim > inicio) {
        indicePivo = particiona(v,inicio,fim);
        quickSort(v,inicio,indicePivo-1);
        quickSort(v,indicePivo+1,fim);
    }
}
```



Depois do particionamento

Método da troca e partição (Quick *sort*)

```
int particiona(int v[], int inicio, int fim) {
  int esq,dir,pivo,aux;

  esq = inicio;  dir = fim;  pivo = v[inicio];
  while (esq < dir) {
    while (v[esq] <= pivo && esq < fim)
      esq++;
    while (v[dir] > pivo)
      dir--;
    if (esq < dir)
      troca(&v[esq], &v[dir]);
  }
  v[inicio] = v[dir];
  v[dir] = pivo;
  return dir;
}
```

esq						dir
0	1	2	3	4	5	6
32	7	80	2	85	75	30

0	esq
6	dir
32	pivo

Método da Troca e Partição (Quick *Sort*)

	esq					dir
0	1	2	3	4	5	6
32	7	80	2	85	75	30

32	pivo

		esq				dir
0	1	2	3	4	5	6
32	7	80	2	85	75	30

```
while (v[esq] <= pivo && esq<fim)
   esq++;
```

		esq				dir
0	1	2	3	4	5	6
32	7	80	2	85	75	30

```
while (v[dir] > pivo)
   dir--;
```

Método da Troca e Partição (Quick *Sort*)

		esq				dir
0	1	2	3	4	5	6
32	7	80	2	85	75	30

		esq				dir
0	1	2	3	4	5	6
32	7	30	2	85	75	80

			esq			dir
0	1	2	3	4	5	6
32	7	30	2	85	75	80

				esq		dir
0	1	2	3	4	5	6
32	7	30	2	85	75	80

32 pivo

```
if (esq < dir)
   troca(&v[esq], &v[dir]);
```

```
while (v[esq] <= pivo && esq<fim)
   esq++;
```

Método da Troca e Partição (Quick *Sort*)

				esq		dir
0	1	2	3	4	5	6
32	7	30	2	85	75	80

				esq	dir	
0	1	2	3	4	5	6
32	7	30	2	85	75	80

				dir		
				esq		
0	1	2	3	4	5	6
32	7	30	2	85	75	80

			dir	esq		
0	1	2	3	4	5	6
32	7	30	2	85	75	80

32 pivo

```
while (v[dir] > pivo)
   dir--;
```

```
if (esq < dir)
   troca(&v[esq], &v[dir]);
```

Método da Troca e Partição (Quick *Sort*)

			dir	esq		
0	1	2	3	4	5	6
32	7	30	2	85	75	80

32 pivo

			dir	esq		
0	1	2	3	4	5	6
2	7	30	2	85	75	80

```
v[inicio] = v[dir];
```

			dir	esq		
0	1	2	3	4	5	6
2	7	30	32	85	75	80

```
v[dir] = pivo;
```

```
return dir;
```

3

Método da Troca e Partição (Quick *Sort*)

```
particiona(v,0,6);
```

0	1	2	3	4	5	6
2	7	30	32	85	75	80

```
particiona(v,0,2);
```

0	1	2	3
2	7	30	32

```
particiona(v,4,6);
```

4	5	6
80	75	85

```
particiona(v,1,2);
```

0	1	2	3	4	5	6
2	7	30	32	75	80	85

```
particiona(v,4,5);
```

0	1	2	3	4	5	6
2	7	30	32	75	80	85

Estrutura de Dados

Aula 13

Pesquisa (busca) e Ordenação (classificação)