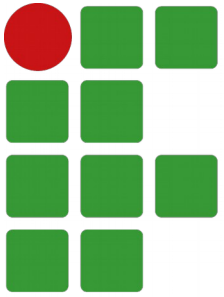
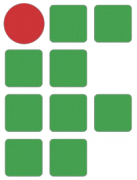


INSTITUTO FEDERAL
Ceará

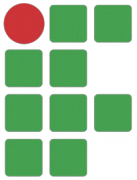
Busca e Ordenação

Introdução



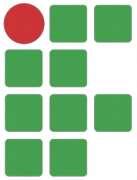
- A busca de um elemento em um conjunto é uma operação básica em Computação
  - Banco de Dados
- Dependendo da forma de armazenamento na memória (primária, secundária)
  - Diferentes estratégias podem ser aplicadas

Introdução



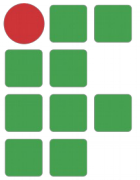
- Memória primária
  - Busca sequencial
  - Busca binária
- Ordenação (ou não)
- Valores duplicados (ou não)

Busca Sequencial



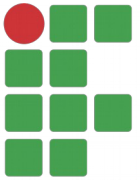
- Temos:
  - Um número inteiro n >= 0
  - Vetor de inteiros v[0… n-1]
  - Número inteiro x
- Precisamos encontrar o índice i, onde v[i] = x
  - Retorna i (OU)
  - Retorna -1

Busca Sequencial



```
int buscaSequencial(int *v, int n, int elem) {
    for (int i=0; i < n; i++) {
        if (elem == v[i]) {
            return i;
        }
    }
    return -1;
}
```

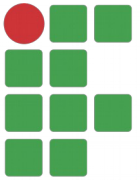
Teste de Mesa



```
int buscaSequencial(int *v, int n, int elem) {
    for (int i=0; i < n; i++) {
        if (elem == v[i]) {
            return i;
        }
    }
    return -1;
}
```

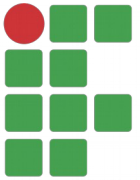
i = 0	-1	2	4	6	7	8
i = 1	-1	2	4	6	7	8
i = 2	-1	2	4	6	7	8
i = 3	-1	2	4	6	7	8
i = 4	-1	2	4	6	7	8
i = 5	-1	2	4	6	7	8

Busca Sequencial Recursiva



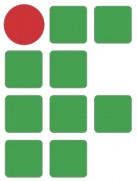
- Condição de parada:
  - Vetor vazio → n = 0
- Fórmula geral
  - Verificar se o elemento está no vetor (x == v[n-1])
  - Se não for encontrado
    - Verificar no elemento antecedente

Busca Sequencial Recursiva



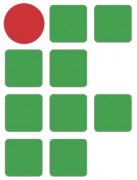
```
int buscaSequencial_R(int *v, int n, int elem) {
    if (n == 0) {
        return -1;
    } else {
        if (elem == v[n-1]) {
            return n-1;
        }
        return buscaSequencial_R(v, n-1, elem);
    }
}
```

Busca Sequencial em Vetor Ordenado



- Ordem:
  - Crescente
  - Decrescente
- Descobrir se:
  - Um elemento está presente
  - Em qual posição um elemento pode ser inserido para continuar ordenado

Busca Sequencial em Vetor Ordenado

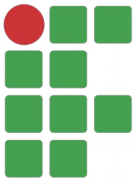


- A princípio o mesmo algoritmo pode ser usado
  - A resposta será encontrada
  - Porém **ineficiente** do ponto de vista do consumo de tempo
  - No pior caso, a comparação será realizada com **todos os elementos** (assim como no vetor não-ordenado)
  - Ele pode ser otimizado

Teste de Mesa

i = 0	1	2	4	6	9	18
i = 1	1	2	4	6	9	18
i = 2	1	2	4	6	9	18
i = 3	1	2	4	6	9	18
i = 4	1	2	4	6	9	18
i = 5	1	2	4	6	9	18

Busca Sequencial em Vetor Ordenado



```
int buscaOrdenada(int *v, int n, int elem) {
    for (int i=0; i<n; i++) {
        if (elem == v[i]) {
            return i;
        } else {
            if (elem < v[i]) {
                return -1;
            }
        }
    }
    return -1;
}
```

Exercícios

Procurar um número inteiro em um vetor com valores duplicados:

a) Imprimir todos os índices do número procurado

b) Imprimir somente o primeiro índice (interrompa a busca)

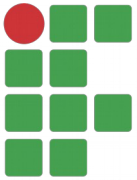
2	2	8	6	10	5	7	5	19

Exercícios

Localizar posição para inserir elemento em vetor ordenado

1	2	6	8	10	15	17	20	29

Exercícios



- Atualizador de valores
  - Percorra um vetor e troque o valor pesquisado por outro fornecido

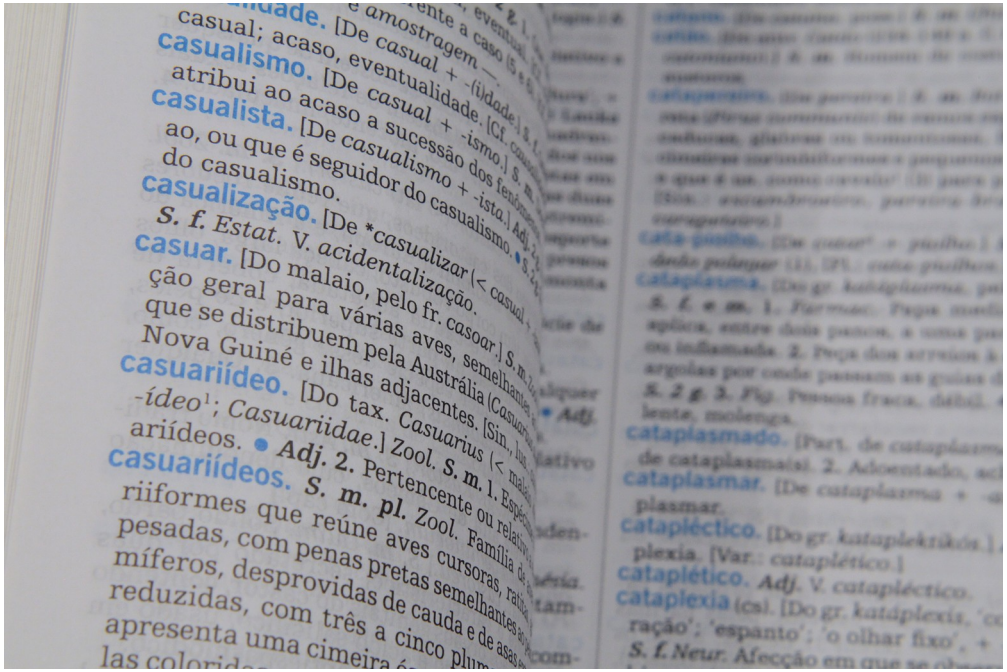
```
troca (int *v, int n, int elem, int novo_elem) {
    int indice = pesquisa (v, n, elem)
    //Realizar a troca de valores
}
```

Exercícios

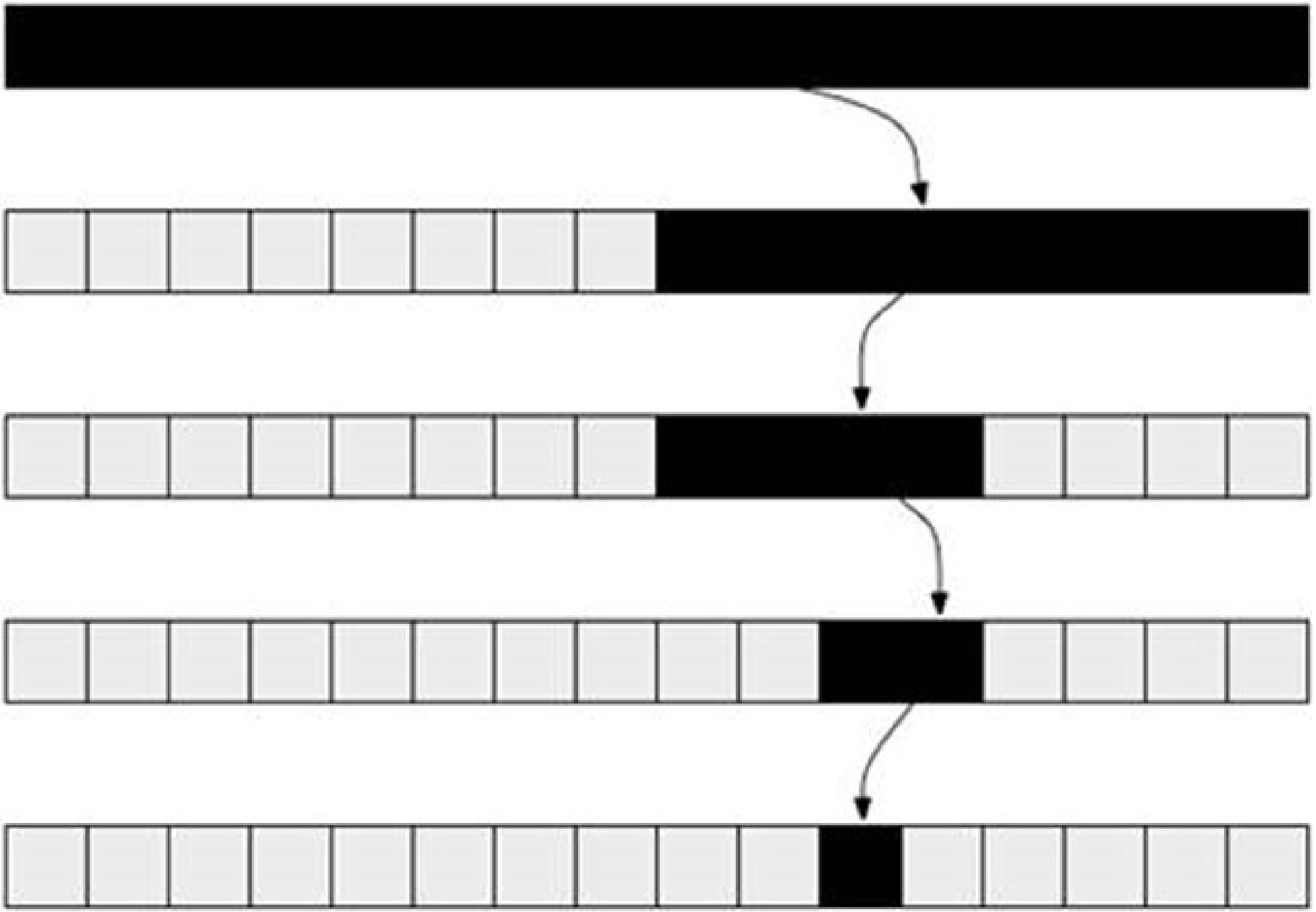
- Escreva um programa que leia uma sequência com N números inteiros e imprime a sequência eliminando os elementos repetidos.
  - Sequência não-ordenada
  - Sequência ordenada

Busca Binária

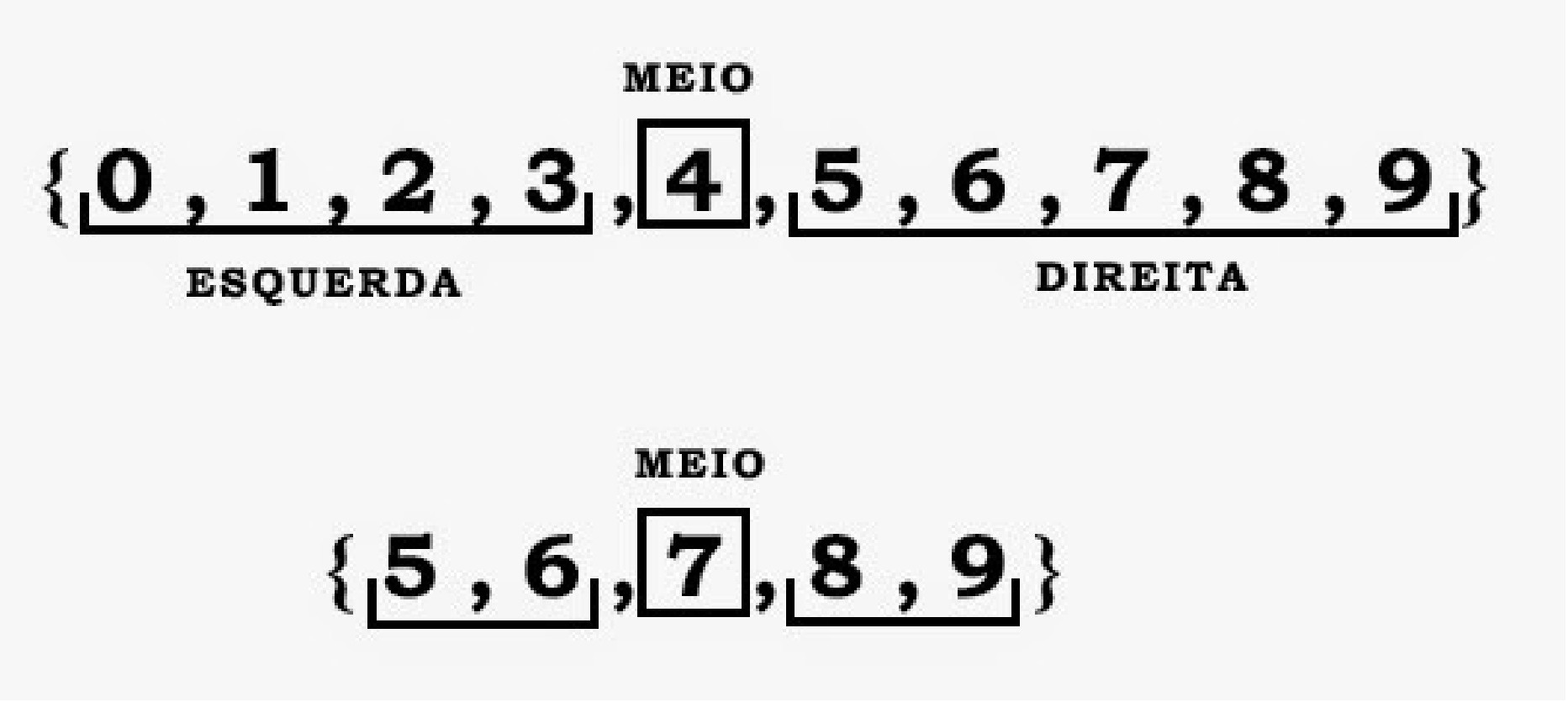
- Podemos fazer o mesmo trabalho de forma mais eficiente
  - Da mesma forma que procuramos uma palavra no dicionário
  - Realiza sucessivas DIVISÕES



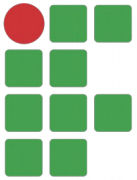
Busca Binária



Busca Binária

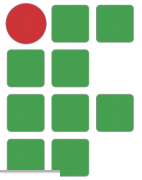


Busca Binária



```c
int busca binaria(int *v, int n, int elem) {
    int inicio, meio, final;
    inicio = 0;
    final = n-1;
    while (inicio <= final) {
        meio = (inicio+final)/2;
        if (elem < v[meio]) {
            final = meio-1; //busca na metade da esqueda
        } else {
            if (elem > v[meio]) {
                inicio = meio+1; //busca na metade da direita
            } else {
                return meio; //encontrado
            }
        }
    }
    return -1; //não encontrado
}
```

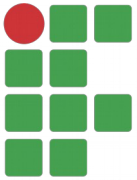
Busca Binária Recursiva



```
int busca_binaria_r(int *v, int e, int d, int elem) {

    if (d >= e) {
        int meio = (e + d)/2;
        if (elem > v[meio]) {
            return busca_binaria_r(v, meio+1, d, elem);
        } else {
            if (elem < v[meio]) {
                return busca_binaria_r(v, e, meio-1, elem);
            }
            else {
                return meio;
            }
        }
    }
    return -1;
}
int main() {
    int vetor_ordenado[] = {1,3,5,6,7,9,10,15,19,23};
    int indice = busca_binaria_r(vetor_ordenado, 0, 9, 10);
    if (indice != -1) {
```

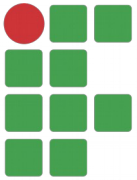
Busca em Vetor de Struct



```c
struct aluno {
    int matricula;
    char nome[30];
    float n1, n2, n3;
};
```

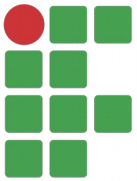
matricula nome[30] n1, n2, n3	matricula nome[30] n1, n2, n3	matricula nome[30] n1, n2, n3	matricula nome[30] n1, n2, n3	matricula nome[30] n1, n2, n3
v[0]	v[1]	v[2]	v[3]	v[4]

Busca em Vetor de Struct



- Para acessar as variáveis do struct
  - v[i].matricula
  - v[i].nome
- Comparação de char*
  - #include<string.h>
  - if (strcmp (nome, v[i].nome) == 0
  - < 0 → v[i].nome < nome
  - > 0 → v[i].nome > nome
  - == 0 → v[i].nome = nome

Funções



```
struct aluno v[4]= {
        {2,"joao",9.5,8.0,9.2},
        {3, "pedro", 7.8, 8.4, 9.9},
        {1, "mariana", 7.8, 8.3, 9.9},
        {4, "beatriz", 7.0, 8.9, 9.5}};
```

```
int buscaSequencialMatricula(struct aluno *alunos, int n, int matricula)
```

```
int buscaSequencialNome(struct aluno *alunos, int n, char * name)
```