

# Estruturas de Dados

Ordenação

# Ordenação (*Sorting*)

- Em diversas aplicações devemos ordenar os dados para operar de forma mais eficiente
- Duas Abordagens:
  - Inserir os elementos respeitando a ordenação (Ordenação por Construção)
  - Aplicar um algoritmo de ordenação em um conjunto já criado
- Os algoritmos de ordenação devem ser eficientes em termos de tempo e espaço

# Ordenação de Vetores

- **Entrada:** vetor com os elementos a serem ordenados
- **Saída:** o mesmo vetor com os seus elementos na ordem especificada
- **Ordenação:** pode ser aplicada a qualquer dado com ordem bem definida
  - Vetor com dados simples (*int*, *float*, etc)

Entrada: 

4	2	5	1
---	---	---	---

+

=

Saída: 

1	2	4	5
---	---	---	---

Relação de Ordem: >

# Ordenação de Vetores (Cont.)

- **Ordenação:** pode ser aplicada a qualquer dado com ordem bem definida
  - Vetor com dados complexos (*structs*)
    - Chave da ordenação escolhida entre os campos
    - Em geral, ineficiente pois toda a estrutura deve ser trocada
  - Vetor de ponteiros para dados complexos
    - Troca da Ordem dos Elementos = Troca de ponteiros

# Algoritmos de Ordenação

- Algoritmos Iterativos de Ordenação:
  - Ordenação Bolha (*Bubble Sort*)
  - Ordenação por Inserção (*Insertion Sort*)
  - etc.
- Algoritmos Recursivos de Ordenação (Abordagem Dividir para Conquistar):
  - Ordenação Rápida (*Quick Sort*)
  - Ordenação por Intercalação (*Merge Sort*)
  - etc.

# Ordenação Bolha (Bubble Sort)

A idéia geral da ordenação bolha é colocar os elementos "maiores" nos seus lugares corretos

4	2	5	1
---	---	---	---

1	2	4	5
---	---	---	---

Quando dois elementos estão fora de ordem, troque-os de posição até que o i-ésimo elemento de maior valor do vetor seja levado para as posições finais do vetor

Maior Elemento

4	2	5	1
---	---	---	---

2	4	5	1
---	---	---	---

2	4	5	1
---	---	---	---

2	4	1	5
---	---	---	---

2o. Maior Elemento

2	4	1	5
---	---	---	---

2	4	1	5
---	---	---	---

2	1	4	5
---	---	---	---

3o. Maior Elemento

2	1	4	5
---	---	---	---

1	2	4	5
---	---	---	---

# Ordenação Bolha

```
void bolha(int n, int *v){
    int i, j, temp;
    for(i=n-1; i>0; i--){
        for(j=0; j<i; j++){
            if(v[j]> v[j+1]){
                temp = v[j]; // Troca
                v[j] = v[j+1];
                v[j+1] = temp;
            }
        }
    }
}
```

**i=3**

<b>j=0</b>	4	2	5	1
<b>j=1</b>	2	4	5	1
<b>j=2</b>	2	4	5	1
	2	4	1	5

**i=2**

2	4	1	5
2	4	1	5
2	1	4	5

**i=1**

2	1	4	5
1	2	4	5

# Ordenação Bolha (Versão II)

Quando o vetor já está ordenado em alguma passagem

```
void bolha(int n, int *v) {  
    int i, j, temp, troca;  
    for(i=n-1; i>0; i--) {  
        troca = 0;  
        for(j=0; j<i; j++)  
            if(v[j]> v[j+1]) {  
                temp = v[j]; // Troca  
                v[j] = v[j+1];  
                v[j+1] = temp;  
                troca = 1;  
            }  
        if(troca==0)  
            return;  
    }  
}
```



# Ordenação Bolha pelo campo IRA

```
#include<stdio.h>
typedef struct aluno{
    char nome[81];
    float ira;
} Aluno;
#define MAX 10000
void inicializa(int n, Aluno **alunos);
void imprime(int n, Aluno **alunos, int i);
void bolhaIRA(int n, Aluno **alunos);
void imprime_todos(int n, Aluno **alunos);
void atualiza(int n, Aluno **alunos, int i);
void exclui(int n, Aluno **alunos, int i);
int main (void){
    Aluno* alunos[MAX]; int i;
    inicializa(MAX,alunos);
    for(i=0;i<MAX;i++)
        atualiza(MAX,alunos,i);
    bolhaIRA(MAX,alunos);
    imprime_todos(MAX,alunos);
    for(i=0;i<MAX;i++)
        exclui(MAX,alunos,i);
    return 0; }
```

# Ordenação Bolha pelo campo IRA

```
void bolhaIRA(int n, Aluno **alunos){
    int i, j;
    int troca = 0;
    Aluno *tempAluno;
    for(i=n-1; i>0; i--){
        troca = 0;
        for(j=0; j<i; j++){
            if(alunos[j]->ira > alunos[j+1]->ira){
                tempAluno = alunos[j];
                alunos[j] = alunos[j+1];
                alunos[j+1] = tempAluno;
                troca = 1;
            }
        }
        if(troca==0)
            return;
    }
}
```

# Complexidade

- A complexidade de um algoritmo pode ser medida pelo esforço computacional
  - Número de Comparações, Atribuições etc.
- No caso da ordenação do método bolha, a complexidade pode ser estimada pelo número de comparações (semelhante ao número de trocas)
  - Na primeira passada:  $n-1$  comparações
  - Na segunda passada:  $n-2$  comparações
  - ...
- $T = (n-1) + (n-2) + \dots + 1 = n \cdot (n-1) / 2$
- O Algoritmo é no pior caso de Ordem Quadrática  $O(n^2)$  e no melhor caso Linear  $O(n)$
- Na prática, a ordenação bolha **não** é utilizada

# Ordenação Rápida (*Quick Sort*)

- Escolha um elemento arbitrário  $x$ , o pivô
- Rearrume o vetor de tal forma que  $x$  fique na posição correta  $v[i]$ 
  - $x$  deve ocupar a posição  $i$  do vetor sse  
todos os elementos  $v[0], \dots, v[i-1]$  são menores que  $x$  e  
todos os elementos  $v[i+1], \dots, v[n-1]$  são maiores que  $x$
- Chame recursivamente o algoritmo para ordenar os (sub-)vetores  $v[0], \dots, v[i-1]$  e  $v[i+1], \dots, v[n-1]$
- Continue até que os vetores que devem ser ordenados tenham 0 ou 1 elemento

# Ordenação Rápida (*Quick Sort*)

Rearruração do vetor para o pivô de  $x=v[0]$

do início para o final, compare  $x$  com  $v[1]$ ,  
 $v[2]$ ,... até encontrar  $v[a]>x$

do final para o início, compare  $x$  com  $v[n-1]$ ,  
 $v[n-2]$  ... até encontrar  $v[b]\leq x$

troque  $v[a]$  e  $v[b]$

continue para o final a partir de  $v[a+1]$  e para  
o início a partir de  $v[b-1]$ . Termine quando  
os pontos de busca se encontram ( $b < a$ )

a posição correta de  $x=v[0]$  é a posição  $b$  e  
 $v[0]$  e  $v[b]$  são trocados

4	2	5	1	$a=1$ $b=3$
---	---	---	---	----------------

4	2	5	1	$a=2$ $b=3$
---	---	---	---	----------------

4	2	5	1	$a=2$ $b=3$
---	---	---	---	----------------

4	2	1	5	$a=3$ $b=2$
---	---	---	---	----------------

4	2	1	5	$a=3$ $b=2$
---	---	---	---	----------------

$2 < 3$

1	2	4	5	$a=3$ $b=2$
---	---	---	---	----------------

# Ordenação Rápida (*Quick Sort*)

25	48	37	12	57	86	33	92	a=1 b=7
----	----	----	----	----	----	----	----	------------

25	48	37	12	57	86	33	92	a=1 b=7
----	----	----	----	----	----	----	----	------------

...

25	48	37	12	57	86	33	92	a=1 b=3
----	----	----	----	----	----	----	----	------------

25	12	37	48	57	86	33	92	a=2 b=2
----	----	----	----	----	----	----	----	------------

25	12	37	48	57	86	33	92	a=2 b=1
----	----	----	----	----	----	----	----	------------

12	25	37	48	57	86	33	92	a=2 b=1
----	----	----	----	----	----	----	----	------------

Rearruração do vetor para o pivô de  $x=v[0]$

do início para o final, compare  $x$  com  $v[1]$ ,  $v[2]$ ,... até encontrar  $v[a]>x$

do final para o início, compare  $x$  com  $v[n-1]$ ,  $v[n-2]$  ... até encontrar  $v[b]<=x$

troque  $v[a]$  e  $v[b]$

continue para o final a partir de  $v[a+1]$  e para o início a partir de  $v[b-1]$ . Termine quando os pontos de busca se encontram ( $b<a$ )

a posição correta de  $x=v[0]$  é a posição  $b$  e  $v[0]$  e  $v[b]$  são trocados

Ordena

Ordena

12	25	37	48	57	86	33	92
----	----	----	----	----	----	----	----

# Ordenação Rápida (*Quick Sort*)

```

void rapida(int n, int *v){
    if(n>1){
        int x = v[0];
        int a =1;    int b = n-1;
        while(1){
            while(a<n && v[a] <=x) a++;
            while(v[b]>x) b--;
            if(a<b){
                int temp = v[a];
                v[a] = v[b];
                v[b] = temp;
                a++;b--;
            }else
                break;
        }
        v[0] = v[b];
        v[b] = x;
        rapida(b,v);
        rapida(n-a,&v[a]);
    }
}
    
```

4	2	5	1	a=1 b=3
---	---	---	---	------------

4	2	5	1	a=2 b=3
---	---	---	---	------------

4	2	5	1	a=2 b=3
---	---	---	---	------------

4	2	1	5	a=3 b=2
---	---	---	---	------------

4	2	1	5	a=3 b=2
---	---	---	---	------------

**2>3**

1	2	4	5
---	---	---	---

1	2	4	5
---	---	---	---

# Complexidade da Ordenação Rápida

Esforço computacional:

- **melhor caso:**
  - pivô representa o valor mediano do conjunto dos elementos do vetor
  - após mover o pivô em sua posição, restarão dois subvetores para serem ordenados, ambos com o número de elementos reduzido à metade, em relação ao vetor original
  - algoritmo é  $O(n \log(n))$
- **pior caso:**
  - pivô é o maior elemento e algoritmo recai em ordenação bolha
- **caso médio:**
  - algoritmo é  $O(n \log(n))$



# Resumo

- **Ordenação Bolha (*Bubble Sort*)**
  - quando dois elementos estão fora de ordem, troque-os de posição até que o  $i$ -ésimo elemento de maior valor do vetor seja levado para as posições finais do vetor
  - continue o processo até que todo o vetor esteja ordenado
  - **Complexidade no Pior Caso  $O(n^2)$ . Não** é usado na prática
- **Ordenação Rápida (*Quick sort*)**
  - coloque um elemento arbitrário  $x$ , o pivô, em sua posição  $k$
  - chame recursivamente o algoritmo para ordenar os (sub-)vetores  $v[0], \dots, v[k-1]$  e  $v[k+1], \dots, v[n-1]$
  - continue até que os vetores que devem ser ordenados tenham 0 ou 1 elemento
  - **Complexidade no Caso Médio  $O(n \log(n))$ , porém no Pior Caso  $O(n^2)$**

# Referência

**Introdução a Estruturas de Dados**, Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Editora Campus, 2004.