MCTA028-15: Programação Estruturada

Aula 10: Ordenação (Primeira Parte)

Wagner Tanaka Botelho
wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com
Universidade Federal do ABC (UFABC)
Centro de Matemática, Computação e Cognição (CMCC)

Introdução

Introdução

- A ordenação é o ato de colocar um conjunto de dados em determinada ordem predefinida. Por exemplo,
  - **Fora de ordem:** 5, 2, 1, 3, 4;
  - **Ordenado:** 1, 2, 3, 4, 5.
- A ordenação permite que o acesso aos dados seja feita de forma mais eficiente;
- Os tipos de ordenação mais comuns são:
  - **Numérica:** 1, 2, 3, 4, 5;
  - **Alfabética:** Ana, André, Bianca, Ricardo.

Introdução

- A ordenação também pode ser:
  - **Crescente:**
    - 1, 2, 3, 4, 5;
    - Ana, André, Bianca, Ricardo.
  - **Decrescente:**
    - 5, 4, 3, 2, 1.
    - Ricardo, Bianca, André, Ana.
- Existem VÁRIOS algoritmos de ordenação;
- Nesta aula, vamos ver:
  - **`Bubble Sort;`**
  - **`Selection Sort;`**
  - **`Insertion Sort.`**

Um algoritmo de ordenação é aquele que coloca os elementos de dada sequência em certa ordem predefinida.

Bubble Sort

Bubble Sort

- Também é conhecido como ordenação por “bolha”;
- Um dos algoritmos de ordenação mais conhecidos.

O algoritmo trabalha de forma a movimentar, uma posição por vez, o maior valor existente na porção não ordenada de um *array* para a sua respectiva posição no *array* ordenado. Isso é repetido até que todos os elementos estejam nas suas posições correspondentes.

Bubble Sort

- É um algoritmo simples, de fácil entendimento e implementação;
- Está entre os mais difundidos métodos de ordenação existentes;
- A eficiência do algoritmo diminui drasticamente à medida que o número de elementos no *array* aumenta:
  - Não é recomendado para aplicações que envolvam grandes quantidades de dados ou que precisem de velocidade.

23	6	-4	32	5

+ Primeira iteração:

i=0	23>6?	23	6	-4	32	5
i=1	23>-4?	6	23	-4	32	5
i=2	23>32?	6	-4	23	32	5
i=3	32>5?	6	-4	23	32	5
i=4	FINAL:	6	-4	23	5	32

MAIOR NÚMERO!

+ Segunda iteração:

i=0	6>-4?	6	-4	23	5	32
i=1	6>23?	-4	6	23	5	32
i=2	23>5?	-4	6	23	5	32
i=3	FINAL:	-4	6	5	23	32

MAIOR NÚMERO!

+ Terceira iteração:

i=0	-4>6?	-4	6	5	23	32
i=1	6>5?	-4	6	5	23	32
i=2	FINAL:	-4	5	6	23	32

MAIOR NÚMERO!

+ Quarta iteração:

i=0	FINAL:	-4	5	6	23	32

Sem Alterações!! Ordenação Concluída!

-4	5	6	23	32

Ex_01.c

```c
#include<stdio.h>
int * bubbleSort(int *vet, int N){
    int sim, aux;

    do{
        sim = 0;
        for(int i=0; i<N-1; i++){
            if(vet[i] > vet[i+1]){
                aux=vet[i];
                vet[i]=vet[i+1];
                vet[i+1]=aux;
                sim=1;
            }
        }
        N--;
    }while(sim!=0);
    return vet;
}

void main(){
    int vetor[3]={23, 6, -4};

    vetor[3] = bubbleSort(vetor,3);

    for (int
        printf
    }
}
```

Retornando um vetor

O vet retornado na função bubbleSort() é armazenado no vetor[]

Ex_02.c

```c
#include<stdio.h>
int * bubbleSort(int *vet, int N){
    int sim, aux;

    do{
        sim = 0;
        for(int i=0; i<N-1; i++){
            if(vet[i] > vet[i+1]){
                aux=vet[i];
                vet[i]=vet[i+1];
                vet[i+1]=aux;
                sim=1;
            }
        }
        N--;
    }while(sim!=0);
    return vet;
}

void main
    int v

    int

    pVet

    pVet = bubbleSort(pVet, 3);

    for (int i=0;i<3;i++){
        printf("%i ", pVet[i]);
    }
}
```

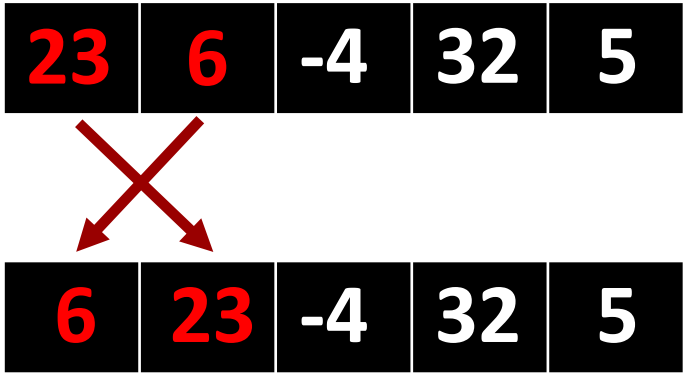
Retornando um vetor

O vet retornado na função bubbleSort() é armazenado no ponteiro pVet.

Ex_01.c

```c
#include<stdio.h>
int * bubbleSort(int *vet, int N){
    int sim, aux;

    do{
        sim = 0;
        for(int i=0; i<N-1; i++){
            if(vet[i] > vet[i+1]){
                aux=vet[i];
                vet[i]=vet[i+1];
                vet[i+1]=aux;
                sim=1;
            }
        }
        N--;
    }while(sim!=0);
    return vet;
}

void main(){
    int vetor[3]={23, 6, -4};

    vetor[3] = bubbleSort(vetor,3);

    for (int i=0;i<3;i++){
        printf("%i ", vetor[i]);
    }
}
```

Comparar se o anterior é maior do que o próximo. Se sim, trocar!



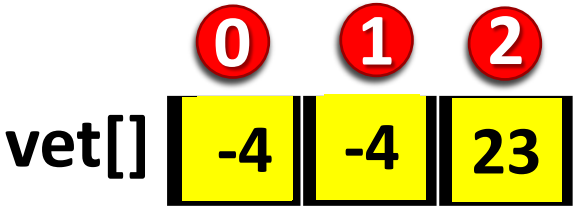
O algoritmo finaliza quando sim==0. A ideia é otimizar o algoritmo, caso nenhuma troca seja realizada.

Ex_01.c

```c
#include<stdio.h>
int * bubbleSort(int *vet, int N){
    int sim, aux;

    do{
        sim = 0;
        for(int i=0; i<N-1; i++){
            if(vet[i] > vet[i+1]){
                aux=vet[i];
                vet[i]=vet[i+1];
                vet[i+1]=aux;
                sim=1;
            }
        }
        N--;
    }while(sim!=0);
    return vet;
}
```

vet=[23, 6, -4]

N=3



Linha	vet[]	i	N	sim	aux
6				~~0~~	
7		~~0~~			
9					~~23~~
10	[0]=6				
11	[1]=23				
12				~~1~~	
7		~~1~~			
9					~~23~~
10	[1]=-4				
11	[2]=23				
12				~~1~~	
7		~~2~~			
15			2		
6				0	
7		0			
9					6
10	[0]=-4				

Ex_01.c

```c
#include<stdio.h>
int * bubbleSort(int *vet, int N){
    int sim, aux;

    do{
        sim = 0;
        for(int i=0; i<N-1; i++){
            if(vet[i] > vet[i+1]){
                aux=vet[i];
                vet[i]=vet[i+1];
                vet[i+1]=aux;
                sim=1;
            }
        }
        N--;
    }while(sim!=0);
    return vet;
}
```

i=0

aux= 6

N=2

0	1	2
vet[] -4	6	23

Linha	vet[]	i	N	sim	aux
11	[1]=6				
12				~~1~~	
7		~~1~~			
15			~~1~~		
6				0	
7		0			
15			0		

Selection Sort

Selection Sort

- É conhecido como ordenação por “seleção”;
- Algoritmo de ordenação de FÁCIL implementação;
- O nome “seleção” é pelo fato do algoritmo, a cada passo, “selecionar” o MELHOR elemento (maior ou menor, dependendo do tipo de ordenação) para ocupar aquela posição do *array*;
- Na prática, o algoritmo possui um desempenho quase sempre superior quando comparado com o `Bubble Sort`.

Selection Sort

O algoritmo `Selection Sort` divide o array em duas partes: a parte ordenada à esquerda do elemento analisado, e a parte que ainda não foi ordenada, à direita do elemento. Para cada elemento do array, começando do primeiro, o algoritmo procura na parte não ordenada (direita) o menor valor (ordenação crescente) e troca os dois valores de lugar. Em seguida, o algoritmo avança para a próxima posição do array e esse processo é feito até que todo o array esteja ordenado.

23	6	-4	3	5

0	1	2	3	4	
23	6	-4	3	5	Procura o menor valor a direita de vet[0] (vet[2]<vet[0]).
-4	6	23	3	5	Troca os valores.
-4	6	23	3	5	Procura o menor valor a direta de vet[1] (vet[3]<vet[1]).
-4	3	23	6	5	Troca os valores.
-4	3	23	6	5	Procura o menor valor a direta de vet[2] (vet[4]<vet[2]).
-4	3	5	6	23	Troca os valores.
-4	3	5	6	23	Procura o menor valor a direta de vet[3]. Nenhuma posição é menor!
-4	3	5	6	23	FIM! Vetor ordenado!

Ex_03.c

```c
#include<stdio.h>
int *selectionSort(int *vet, int N){
    int i=0, j=0, menor=0, troca=0;
    for(i=0;i<N-1;i++){
        menor=i;
        for(j=i+1;j<N;j++){
            if(vet[j] < vet[menor]){
                menor=j;
            }
        }
        if(i!=menor){
            troca=vet[i];
            vet[i]=vet[menor];
            vet[menor]=troca;
        }
    }
    return vet;
}

void main(){
    int vetor[4]={23, 6, -4, 3};

    vetor[4] = selectionSort(vetor,4);

    for (int i=0;i<4;i++){
        printf("%i ", vetor[i]);
    }
}
```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\10\C

```
-4 3 6 23
Process returned 3 (0x3)   execution time
Press any key to continue.
```

*Ex_03.c

```c
#include<stdio.h>
int *selectionSort(int *vet, int N){
    int i=0, j=0, menor=0, troca=0;
    for(i=0;i<N-1;i++){
        menor=i;
        for(j=i+1;j<N;j++){
            if(vet[j] < vet[menor]){
                menor=j;
            }
        }
        if(i!=menor){
            troca=vet[i];
            vet[i]=vet[menor];
            vet[menor]=troca;
        }
    }
    return vet;
}
```

vet=[23, 6, -4, 3]

N=4

Linha	vet	N	i	j	menor	troca
4			0			
5					0	
6				1		
8					1	
6				2		
8					2	
6				3		
6				4		
12						23
13	[0]=-4					
14	[2]=23					
4			1			
5					1	
6				2		
6				3		
8					3	
6				4		

*Ex_03.c

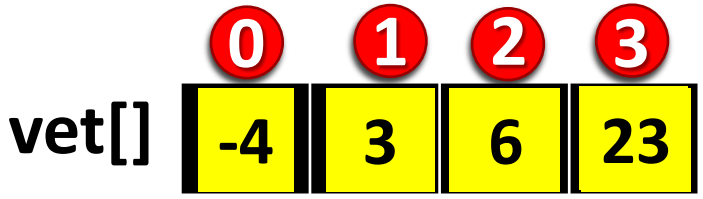
```c
#include<stdio.h>
int *selectionSort(int *vet, int N){
    int i=0, j=0, menor=0, troca=0;
    for(i=0;i<N-1;i++){
        menor=i;
        for(j=i+1;j<N;j++){
            if(vet[j] < vet[menor]){
                menor=j;
            }
        }
        if(i!=menor){
            troca=vet[i];
            vet[i]=vet[menor];
            vet[menor]=troca;
        }
    }
    return vet;
}
```

vet=[23, 6, -4, 3]

N=4

i=1

menor=3



Linha	vet	N	i	j	menor	troca
12						6
13	[1]=3					
14	[3]=6					
4			2			
5					2	
6				3		
8					3	
6				4		
12						23
13	[2]=6					
14	[3]=23					
4			3			

Insertion Sort

Insertion Sort

- É conhecido como ordenação por “inserção”;
- Também é um algoritmo de ordenação simples;
- Tem esse nome, pois se assemelha ao processo de ordenação de um conjunto de cartas de baralhos com as mãos:
  - Pega uma carta de cada vez e a “insere” em seu devido lugar, sempre deixando as cartas da mão em ORDEM.

Insertion Sort

- O algoritmo é mais eficiente do que o `Selection Sort` e o `Bubble Sort`;
- Trata-se de um dos mais rápidos algoritmos de ordenação para conjuntos pequenos de dados;
- **Estável:**
  - A ordem dos elementos IGUAIS não muda durante a ordenação;
- ***Online***:
  - Pode ordenar elementos na medida em que os recebe, ou seja, **NÃO** precisa ter todo o conjunto de dados para coloca-los em ordem.

Insertion Sort

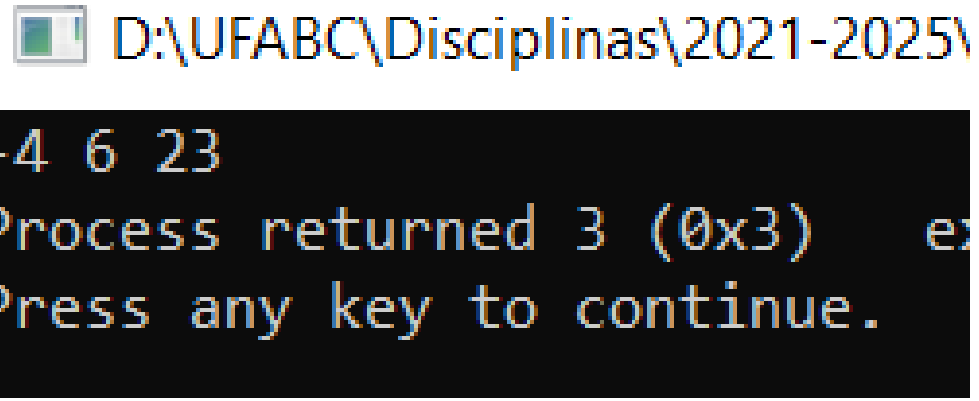
O algoritmo `Insertion Sort` percorre um *array* e, para cada posição X, verifica se o seu valor está na posição correta. Isso é feito andando para o começo do *array*, a partir da posição X, e movimentando uma posição para frente os valores que são maiores do que o valor da posição X. Desse modo, teremos uma posição livre para inserir o valor da posição X em seu devido lugar.

23	6	-4	3	5

	0	1	2	3	4	
i=1	23	6	-4	3	5	Desloca os valores à esquerda que são maiores do que vet[1].
	6	23	-4	3	5	
i=2	6	23	-4	3	5	Desloca os valores à esquerda que são maiores do que vet[2].
	-4	6	23	3	5	
i=3	-4	6	23	3	5	Desloca os valores à esquerda que são maiores do que vet[3].
	-4	3	6	23	5	
i=4	-4	3	6	23	5	Desloca os valores à esquerda que são maiores do que vet[4].
	-4	3	5	6	23	

Ex_04.c

```c
#include<stdio.h>
int *insertionSort(int *vet, int N){
    int i=0, j=0, atual=0;
    for(i=0;i<N;i++){
        atual=vet[i];
        j=i;
        while(j>0&&atual<vet[j-1]){
            vet[j]=vet[j-1];
            j--;
        }
        vet[j]=atual;
    }
    return vet;
}

void main(){
    int vetor[3]={23, 6, -4};

    vetor[3] = insertionSort(vetor, 3);

    for (int i=0;i<3;i++){
        printf("%i ", vetor[i]);
    }
}
```

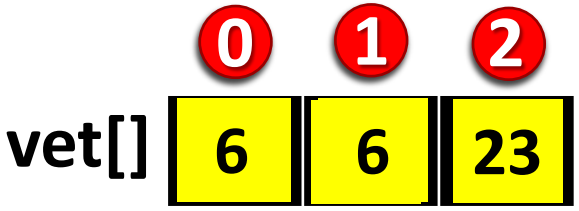


Ex_04.c

```
1  #include<stdio.h>
2  int *insertionSort(int *vet, int N){
3      int i=0, j=0, atual=0;
4      for(i=0;i<N;i++){
5          atual=vet[i];
6          j=i;
7          while(j>0&&atual<vet[j-1]){
8              vet[j]=vet[j-1];
9              j--;
10         }
11         vet[j]=atual;
12     }
13     return vet;
14 }
```

vet=[23, 6, -4]

N=3



Linha	vet	N	i	j	atual
4			0		
5					23
6				0	
11	[0]=23				
4			1		
5					6
6				1	
8	[1]=23				
9				0	
11	[0]=6				
4			2		
5					-4
6				2	
8	[2]=23				
9				1	
8	[1]=6				
9				0	

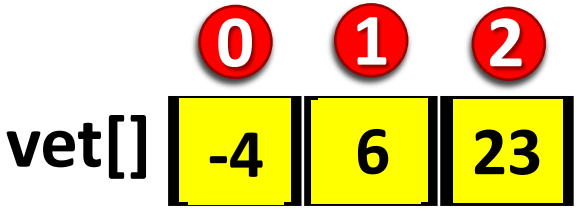
Ex_04.c

```
1  #include<stdio.h>
2  int *insertionSort(int *vet, int N){
3      int i=0, j=0, atual=0;
4      for(i=0;i<N;i++){
5          atual=vet[i];
6          j=i;
7          while(j>0&&atual<vet[j-1]){
8              vet[j]=vet[j-1];
9              j--;
10         }
11         vet[j]=atual;
12     }
13     return vet;
14 }
```

vet=[23, 6, -4]

N=3

i=2



Linha	vet	N	i	j	atual
11	[0]=-4				
4			3		

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011;
- BACKES, André. Estrutura de Dados Descomplicada em Linguagem C. GEN LTC. 2016.