BC1424
Algoritmos e Estruturas de Dados I

Aula 11:
Métodos simples de ordenação
(SelectionSort, InsertionSort, BubbleSort)

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Ordenação

- Ordenar corresponde ao **processo de re-arranjar um conjunto de objetos** em ordem ascendente ou descendente (geralmente considerado no primeiro passo para resolver um problema prático).
- O objetivo principal da ordenação **é facilitar a recuperação posterior** de itens do conjunto ordenado.
- As ordens mais utilizadas são a numérica e lexicográfica.
- **Diversos algoritmos** de ordenação serão estudados e implementados...

O problema de ordenar

- Um vetor v[0..n-1] é crescente se v[0] ≤ v[1] ≤ … ≤ [n-1]

- O problema de ordenação de um vetor consiste em **rearranjar (ou seja, permutar) os elementos de um vetor v[0..n-1] de tal modo qu ele se torne crescente.**

- **Vetores ordenados:**
  - {1, 1, 1, 1, 1, 1, 1, 1}
  - {0, 1, 1, 1, 2, 3, 4, 4, 4, 4, 4, 4, 100}

Verificar se um vetor v[0..n-1] é crescente

Vetor crecente?

```c
#include <stdio.h>

int VerificaCrescente(int v[], int n) {
    int i;

    for (i=0; i<n-1; i++)
        if (v[i]>v[i+1])
            return 0;
    return 1;
}

int main()
{
    int v[] = {0, 1, 1, 1, 2, 3, 4, 4, 4, 4, 4, 4, 100};
    int n=sizeof(v)/sizeof(v[0]);

    printf("%d\n", VerificaCrescente(v, n) );
}
```

Crie uma versão recursiva da função VerificaCrescente

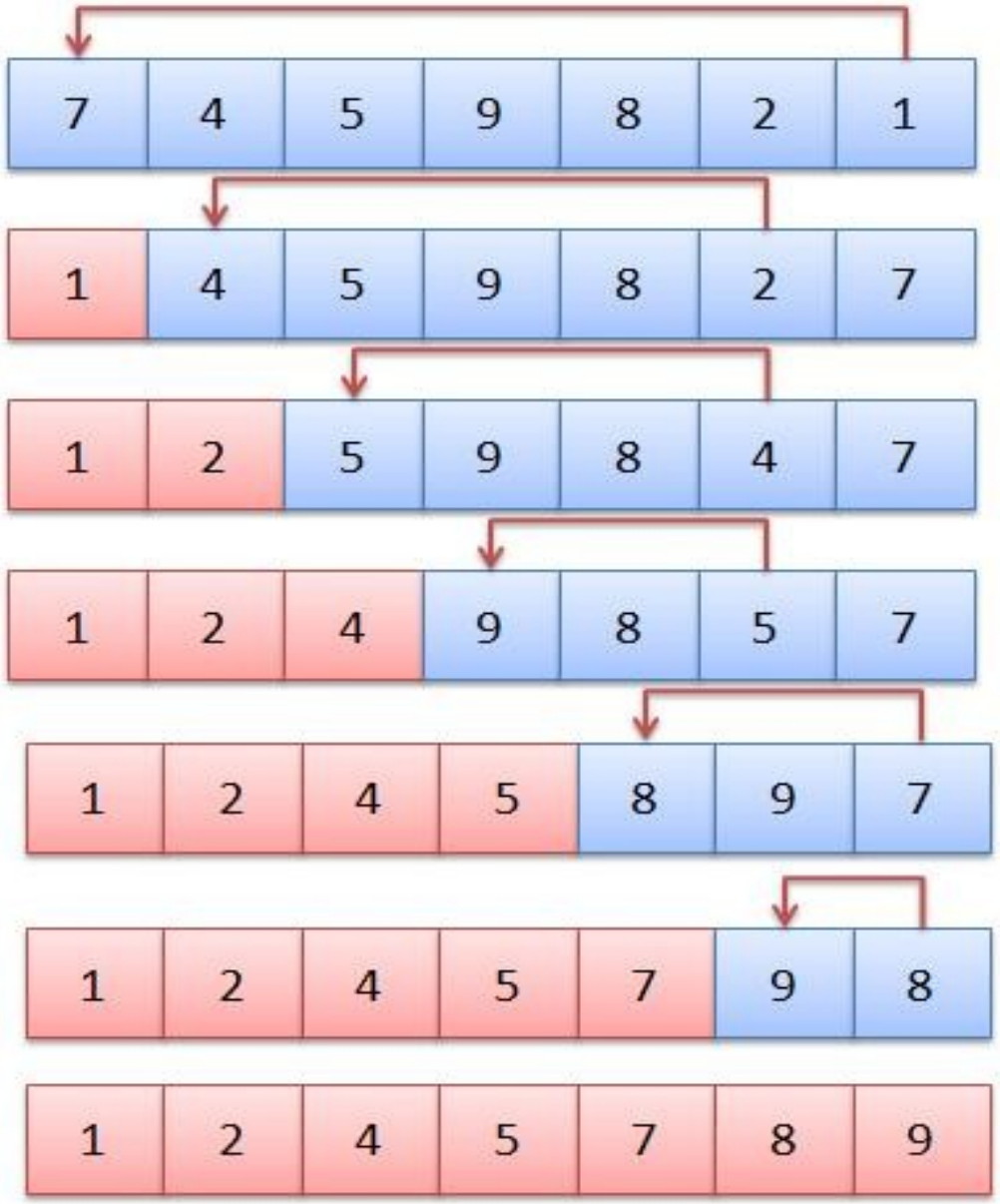
Vetor crecente?

```
int VerificaCrescenteRec(int v[], int n) {
    if (n==1)
        return 1;
    if (v[n-1]>v[n])
        return 0;
    else
        return VerificaCrescenteRec(v, n-1);
}
```

Complexidade computacional: No pior caso = O(n)

SelectionSort:
Algoritmo de seleção

SelectionSort



Animação: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

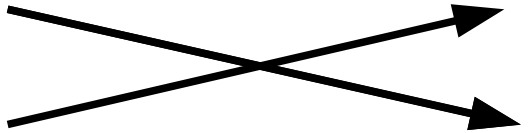
SelectionSort.c (tidia-ae)

```c
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

SelectionSort.c (tidia-ae)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



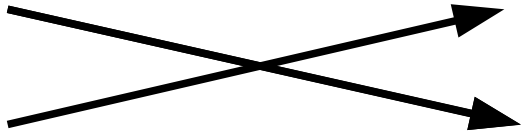
A = 'Maria'
B = 'Joao'

A = B
B = A

SelectionSort.c (tidia-ae)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



A = 'Maria'
B = 'Joao'

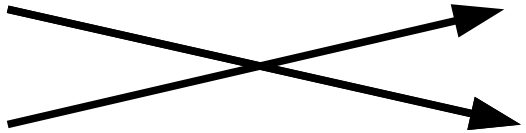
A = B
B = A

A = 'Maria'
B = 'Maria'

SelectionSort.c (tidia-ae)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



A = 'Maria'
B = 'Joao'

aux = A
A = B
B = aux

aux = 'Joao'
A = 'Maria'
B = 'Joao'

SelectionSort.c (tidia-ae)

```c
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

Complexidade computacional: No pior caso?

SelectionSort

- Quanto tempo o algoritmo consome para fazer o serviço?
  (A complexidade computacional é proporcional ao número de comparações v[iMin]>v[j])

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

SelectionSort

- Quanto tempo o algoritmo consome para fazer o serviço?
  (A complexidade computacional é proporcional ao número de comparações v[iMin]>v[j])

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

SelectionSort

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = (n–1)(n)/2

Tempo = $n^2/2 – n/2$

SelectionSort

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = (n–1)(n)/2

Tempo = $n^2/2 – n/2$

Se o vetor tiver n=1000 elementos, o número de comparações (tempo) será proporcional a: **499500**.

SelectionSort

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = (n–1)(n)/2

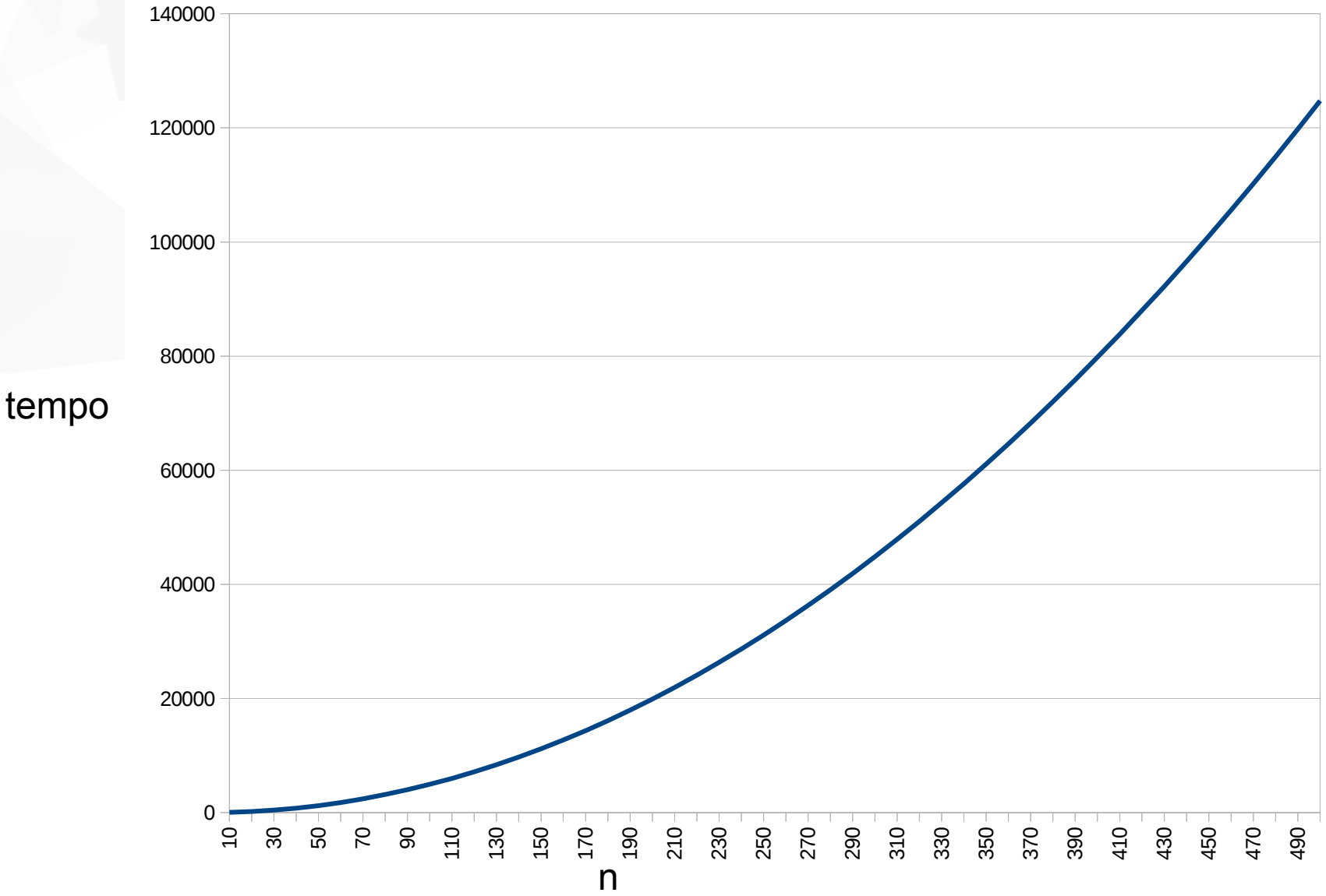
Tempo = $n^2/2 – n/2$

Se o vetor tiver n=1000 elementos, o número de comparações (tempo) será proporcional a: **499500**.

Se o computador fizer no máximo 10 comparações por segundo, o tempo gasto será de: 49950 segundos = 832 min = **13 horas**

SelectionSort

Complexidade computacional: No pior caso = $O(n^2)$



SelectionSort

- Crie uma versão recursiva do **SelectionSort**

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

SelectionSort

```
void SelectionSortRec (int v[], int n, int indice) {
    int j, aux, iMin;

    if (indice>=n-1)
        return;

    iMin = indice;

    for (j=indice+1; j<n; j++) {
        if (v[iMin]>v[j])
            iMin = j;
    }

    if (iMin!=indice) {
        aux = v[iMin];
        v[iMin] = v[indice];
        v[indice] = aux;
    }

    return SelectionSortRec (v, n, indice+1);
}
```

Chamado inicial: `SelectionSortRec(v, n, 0);`

SelectionSort

V = [9, 8, 7, 6, 5, 4, 3, 2, 1] Indice = 0

[1, 8, 7, 6, 5, 4, 3, 2, 9] Indice = 1

[1, 2, 7, 6, 5, 4, 3, 8, 9] Indice = 2

[1, 2, 3, 6, 5, 4, 7, 8, 9] Indice = 3

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 4

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 5

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 6

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 7

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 8

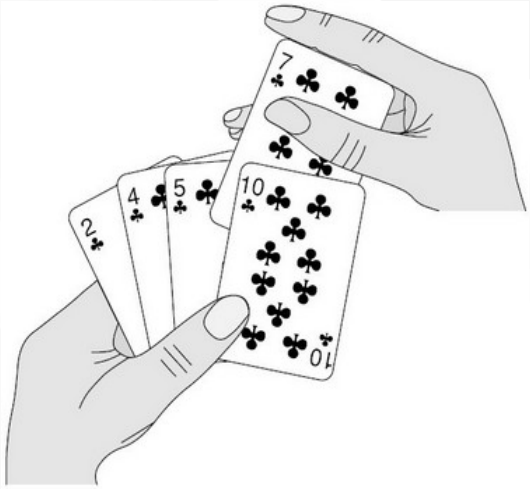
Indice >= n-1
8>=9-1

SelectionSort (versão elegante)

```
// Versão João Gabriel Almeida
void SelectionSortRec2 (int v[], int n) {
    int j, iMin, aux;

    if (n==1)
        return;
    else {
        iMin = 0;
        for (j=1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=0) {
            aux = v[iMin];
            v[iMin] = v[0];
            v[0] = aux;
        }

        SelectionSortRec2(v+1, n-1);
    }
}
```

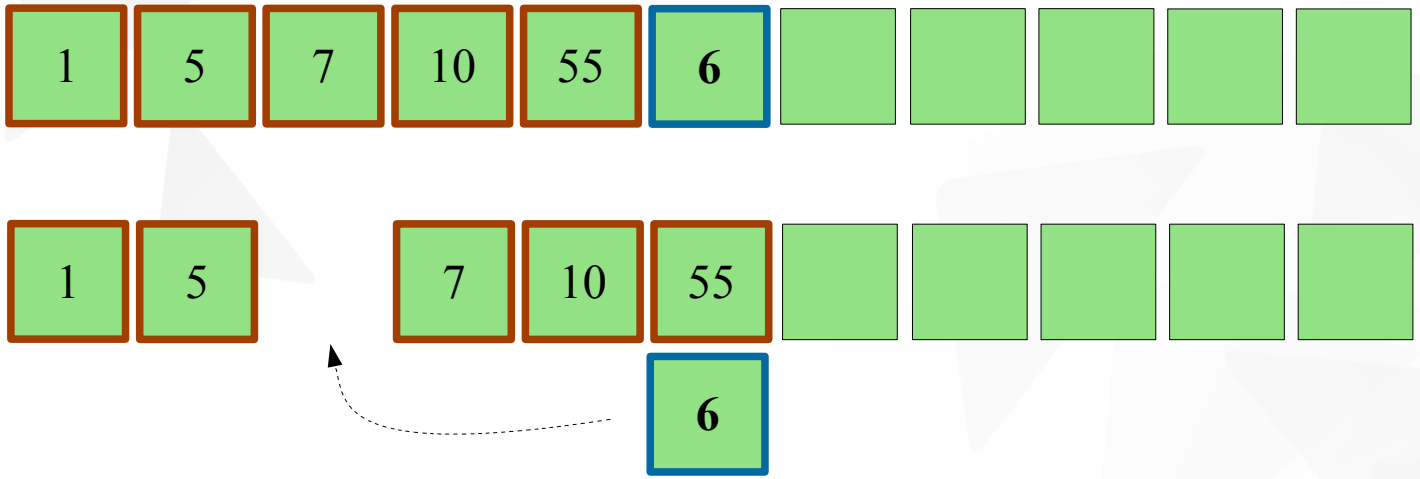
InsertionSort:
Algoritmo de inserção

InsertionSort



Método preferido dos jogadores de cartas

Em cada passo, a partir do i=2, o I-ésimo elemento da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.

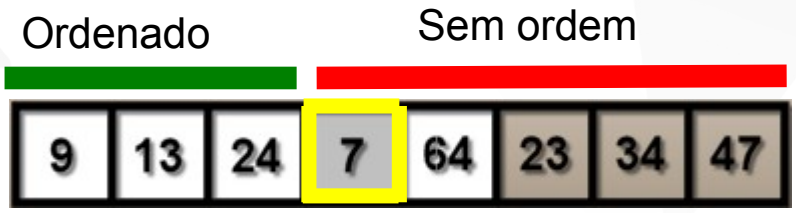


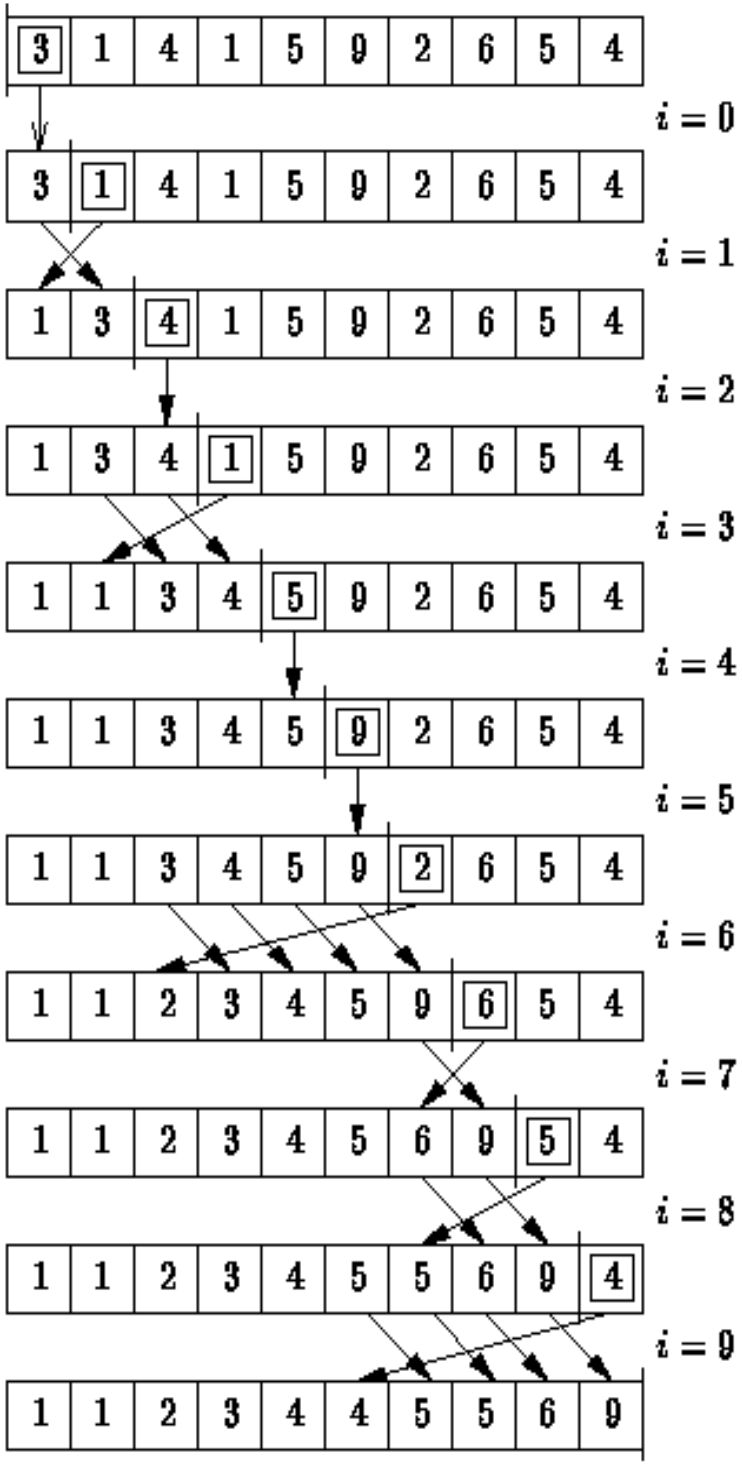
Animação: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

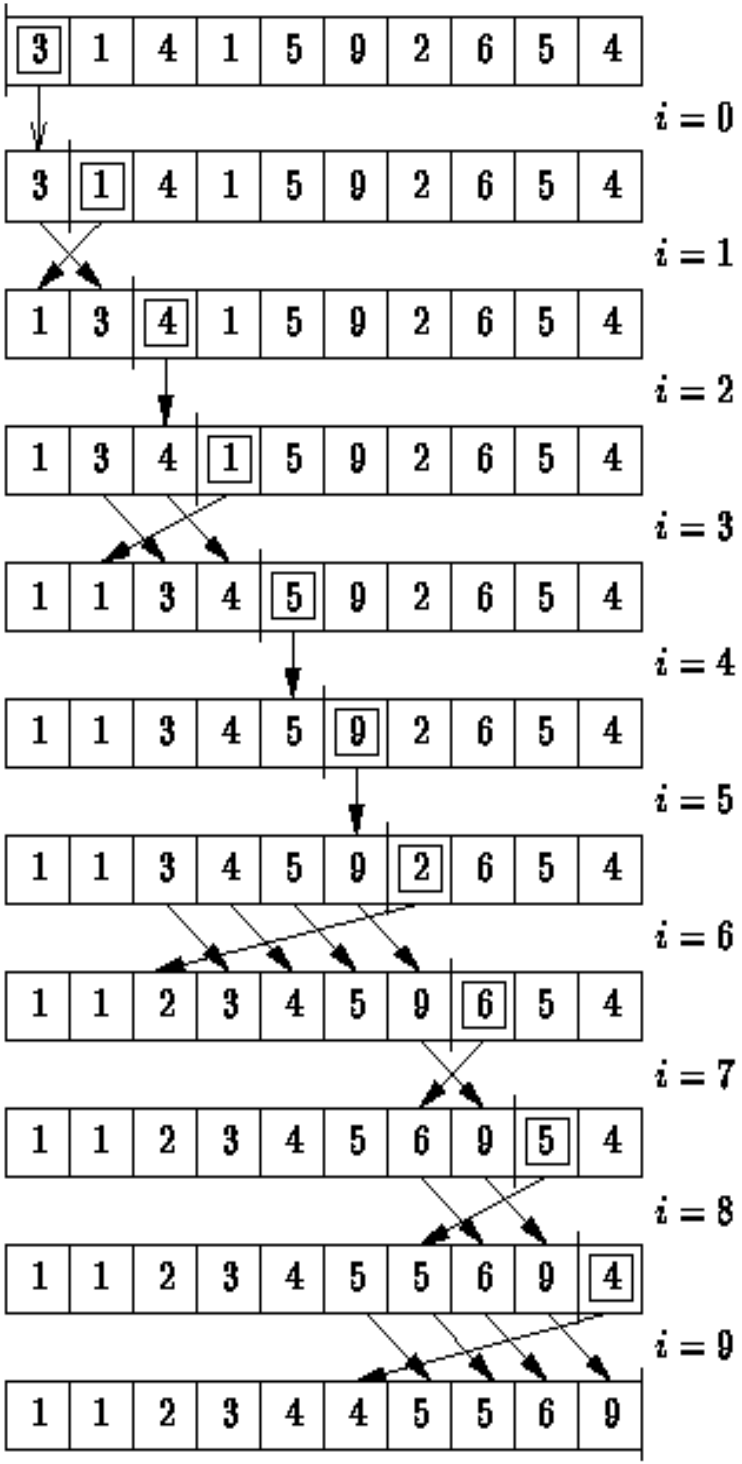
InsertionSort

- A principal característica deste algoritmo consiste em ordenar o vetor utilizando um subvetor ordenado em seu inicio.

- **A cada novo passo, acrescentamos a este subvetor mais um elemento até atingirmos o último elemento de um arranjo.**




3 1 4 1 5 9 2 6 5 4
i = 0
3 1 4 1 5 9 2 6 5 4
i = 1
1 3 4 1 5 9 2 6 5 4
i = 2
1 3 4 1 5 9 2 6 5 4
i = 3
1 1 3 4 5 9 2 6 5 4
i = 4
1 1 3 4 5 9 2 6 5 4
i = 5
1 1 3 4 5 9 2 6 5 4
i = 6
1 1 2 3 4 5 9 6 5 4
i = 7
1 1 2 3 4 5 6 9 5 4
i = 8
1 1 2 3 4 5 5 6 9 4
i = 9
1 1 2 3 4 4 5 5 6 9



Crie uma função iterativa que permita ordenar de forma ascendente uma lista dada como entrada.

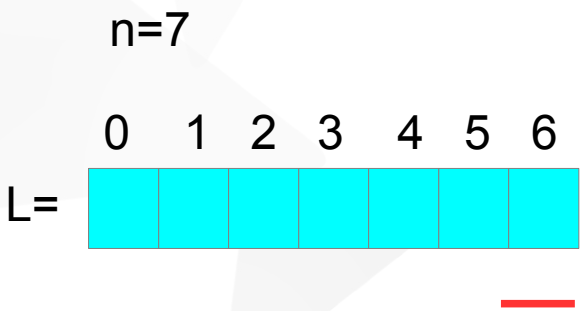
void InsertionSort (int[] v, int n)

InsertionSort

```
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i++) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux ; j--)
                v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

P2: Versão recursiva

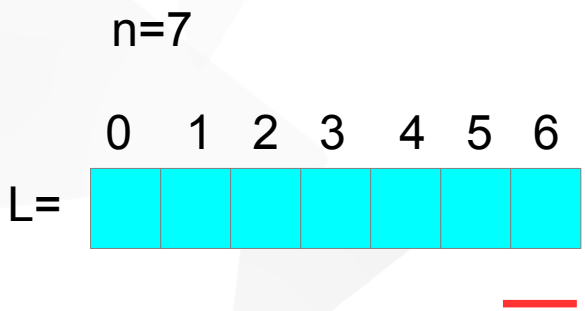
InsertionSort



	Comparações
i=1	1
i=2	2
i=3	3
…	
i=n-1	n-1

Tempo = (n–1)(n)/2
Tempo = $n^2/2 – n/2$

InsertionSort



	Comparações
i=1	1
i=2	2
i=3	3
…	
i=n-1	n-1

Tempo = (n–1)(n)/2
Tempo = $n^2/2 – n/2$

Este algoritmo é o mais apropriado quando a lista estiver semi-ordenada.

BubbleSort:
Ordenação pelo método da bolha
Ordenação por troca dois-a-dois

Bubble sort

- O algoritmo de ordenação baseado em **troca**, consiste em intercalar pares de elementos que não estão em ordem até que não exista mais pares.

- **O principio do bolha é a troca de valores entre posições consecutivas fazendo com que os valores mais altos “borbulhem” para o final do vetor.**

Animação: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Bubble sort

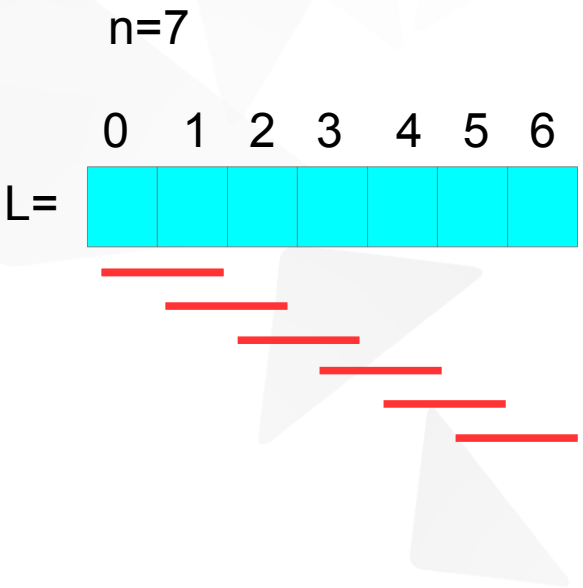
Crie uma função iterativa que permita ordenar de forma ascendente uma lista dada como entrada.

void BubbleSort (int[] v, int n)

Bubble sort

```
void BubbleSort (int v[], int n) {
    int i, j, aux;

    for (i=n-1; i>=1; i--) {
        for (j=0; j<i; j++) {
            if (v[j]>v[j+1]) {
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
        }
    }
}
```

Bubble sort



	Comparações
j=n-1	n-1
j=n-2	n-2
j=n-3	n-3
…	
j=1	1

Tempo = (n–1)(n)/2
Tempo = $n^2/2 – n/2$

Lista 05:

Sorting

- **Intro to Tutorial Challenges**
- **Insertion Sort - Part 1**
- **Insertion Sort - Part 2**
- **Running Time of Algorithms**

Será utilizado um programa de deteção de plágio em todas as submissões!
Plágio → reprovação

Data: **29**/Março (domingo) até às 23h50.

Envio: Através do Tidia.

Arquivos:

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA_nomeDoProblema.pdf

Exemplo: 10123456_solveMeFirst**.c**
10123456_solveMeFirst**.pdf**