

**Aula 17:**  
**- Métodos simples de ordenação**

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

3Q-2017

# Algoritmos para ordenar elementos

- Baseado em comparações:
  - Bogo sort
  - Selection sort
  - Insertion sort
  - Bubble sort
  - Mergesort
  - Quicksort
  - Heapsort
- Ordenação em tempo linear:
  - Radix sort
- Ordenação de primeiros elementos (seleção parcial):
  - Partial Quicksort



**(1)**

**Bogo sort**

**Miracle sort**

**Monkey sort**

**Stupid sort**

```
void imprimir(int v[], int n) {
    int i;
    for (i=0; i<n; i=i+1) {
        printf("%d ", v[i] );
    }
    printf("\n");
}
```

```
void bogoSort(int v[], int n) {
    while (crescenteRec(v,n)==0) {
        embaralhar(v,n);
    }
}
```

```
int main()
{
    int v[] = {100,4,999,555,222,3,0,-1};
    int n=sizeof(v)/sizeof(v[0]);

    imprimir(v, n);
    bogoSort(v, n);
    imprimir(v, n);
}
```

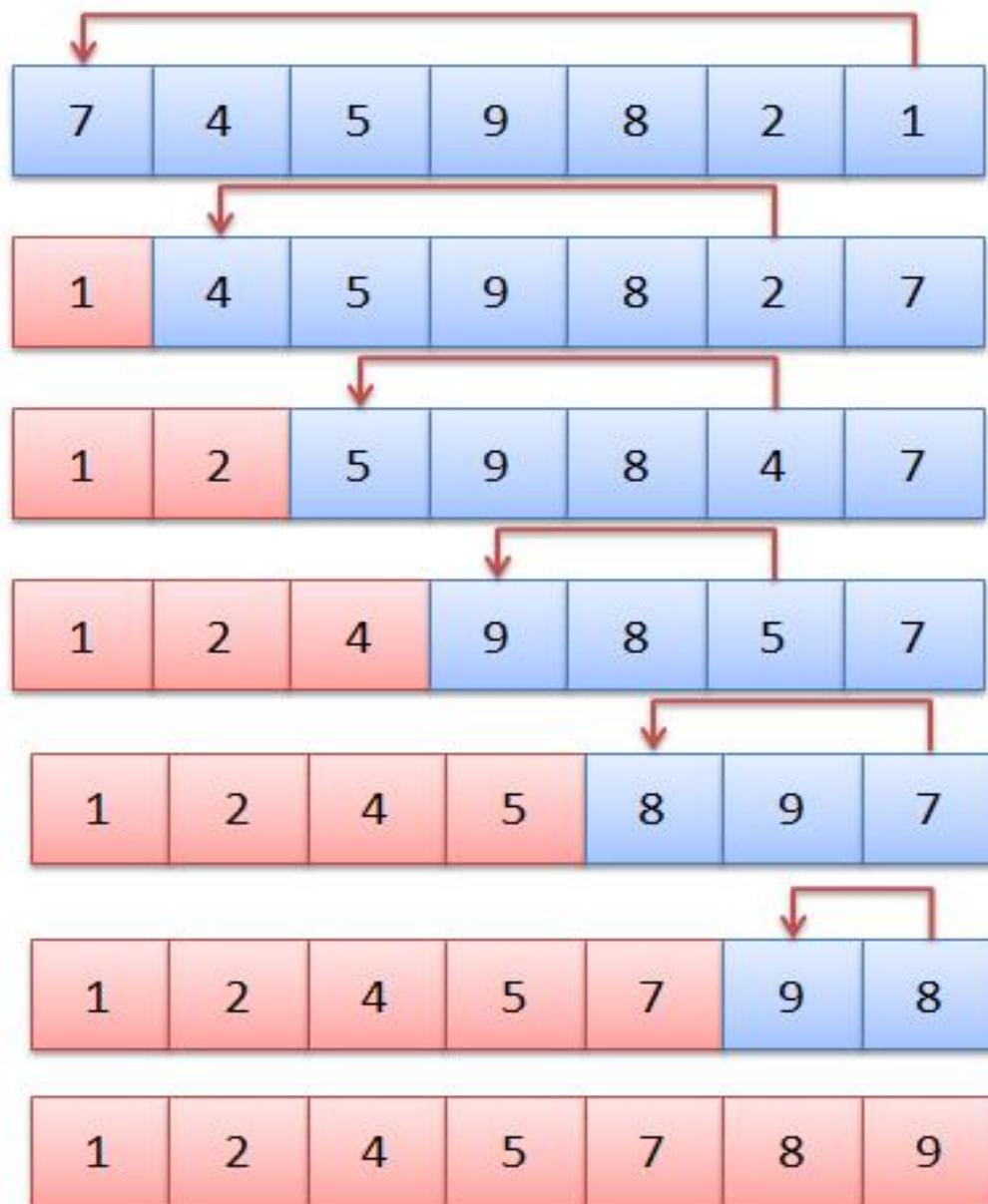
Número de comparações  $T(n)$ :

- No melhor caso:  $T(n) = n-1$
- No pior caso:  $T(n) = ?$



**(2)**  
**Selection Sort:**  
**Algoritmo de ordenação por seleção**

# Selection Sort



# Selection Sort

```
void SelectionSort (int v[], int n) {  
    int i, j, iMin, aux;  
  
    for (i=0; i<n-1; i=i+1) {  
        iMin = i;  
  
        for (j=i+1; j<n; j=j+1) {  
            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?

# Selection Sort

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

# Selection Sort

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

9 8 7 6 5 4 3 2 1

1 8 7 6 5 4 3 2 9

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
...	
n-1 iteração	1

# Selection Sort

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
...	
n-1 iteração	1

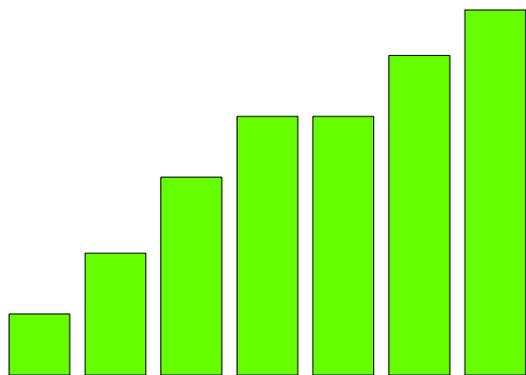
$$\text{Tempo} = (n-1)(n)/2$$

$$\text{Tempo} = n^2/2 - n/2$$

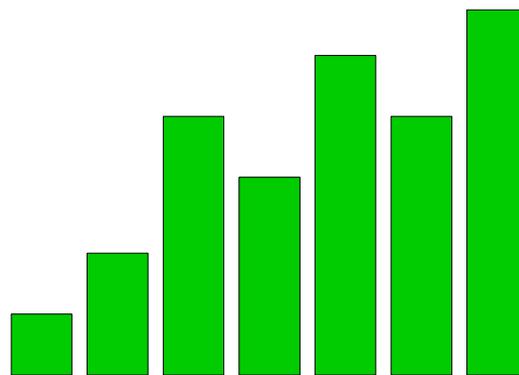
Se o vetor tiver **n=10,000** elementos, o número de comparações (tempo) será proporcional a: **49,995,000**.

Se o computador fizer **1000** comparações por segundo, o tempo gasto será de: 49995 segundos = 832.25 min = **13.88 horas**

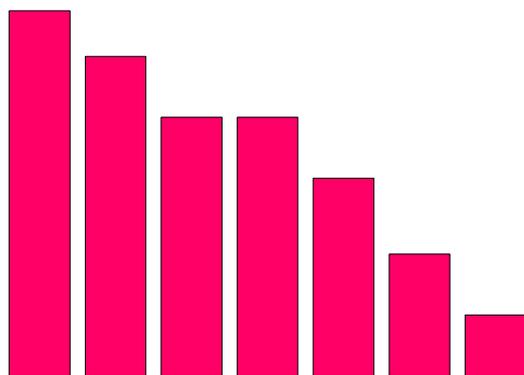
# Selection Sort: Eficiente para quais sequências?



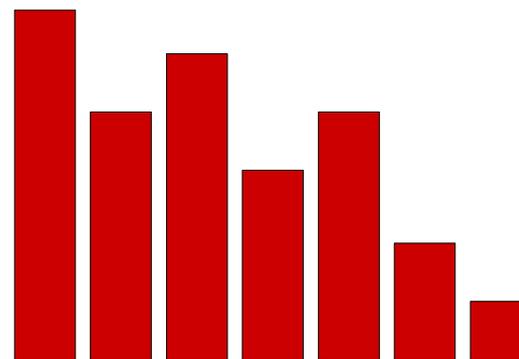
Vetor Crescente



Vetor Parcialmente Crescente

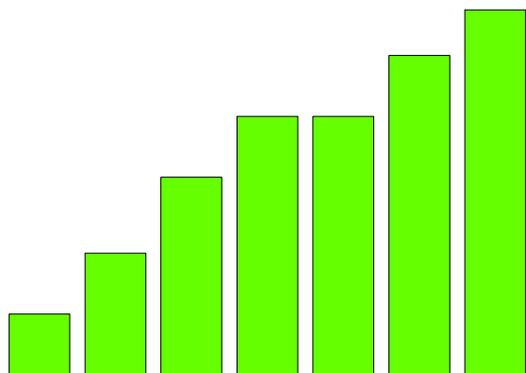


Vetor Decrescente

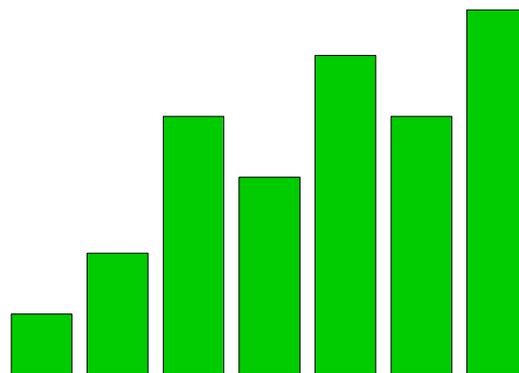


Vetor Parcialmente Decrescente

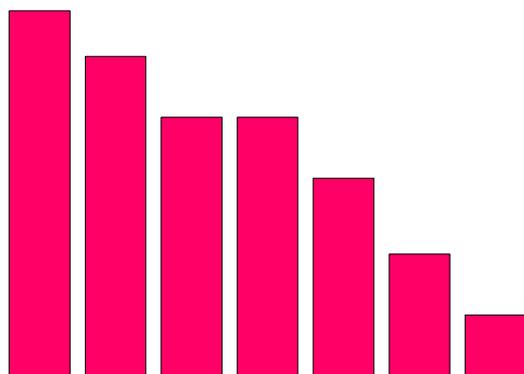
# Selection Sort: Eficiente para quais sequências?



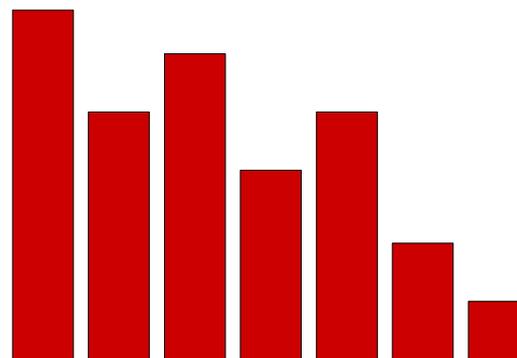
Vetor Crescente



Vetor Parcialmente Crescente



Vetor Decrescente

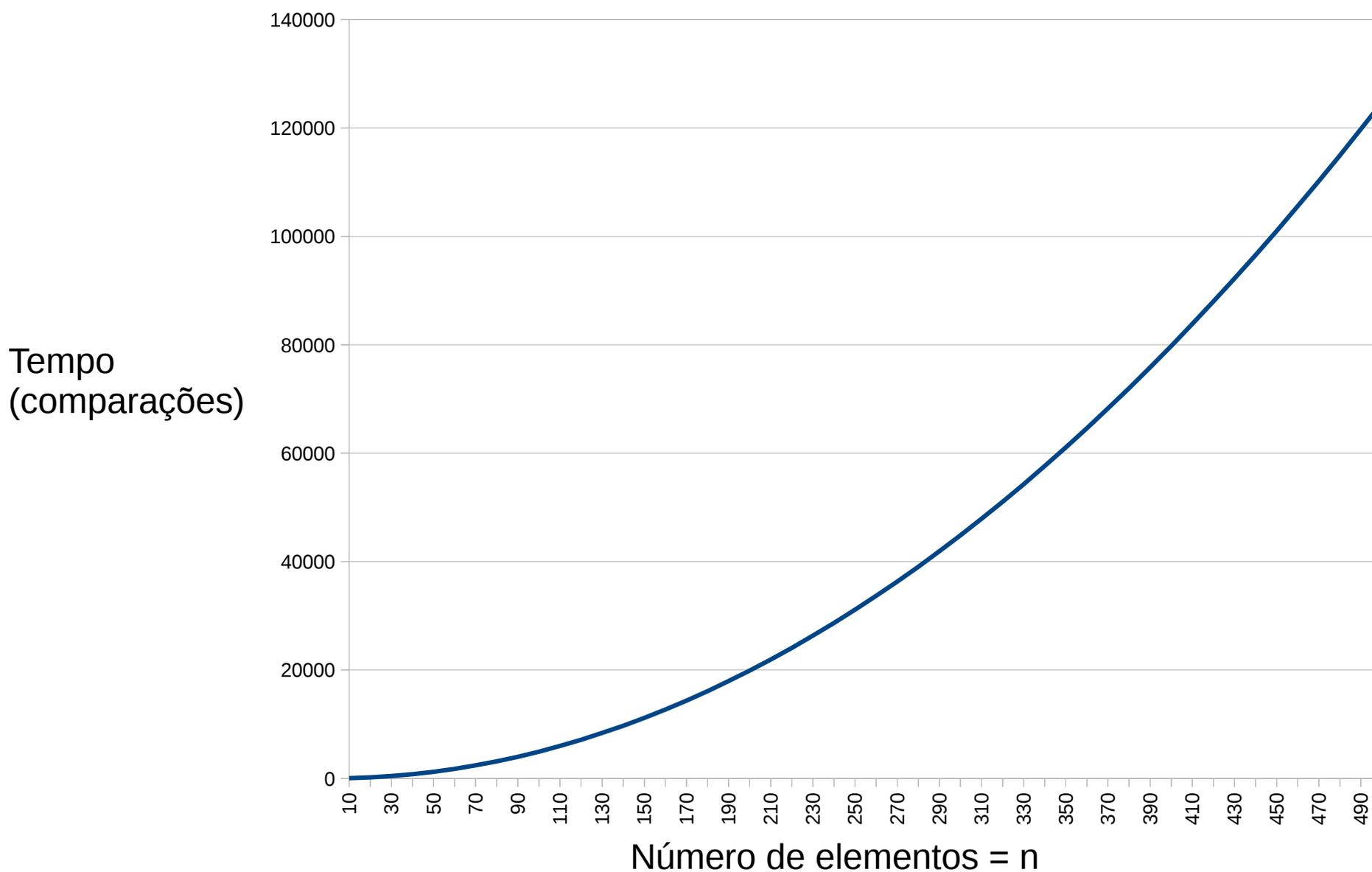


Vetor Parcialmente Decrescente

Para qualquer sequência o algoritmo custa  $n^2/2 - n/2 \rightarrow O(n^2)$

# Selection Sort

Número de comparações, em qualquer caso =  $O(n^2)$



# Versão recursiva do Selection Sort

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

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

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

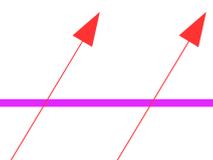
        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

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

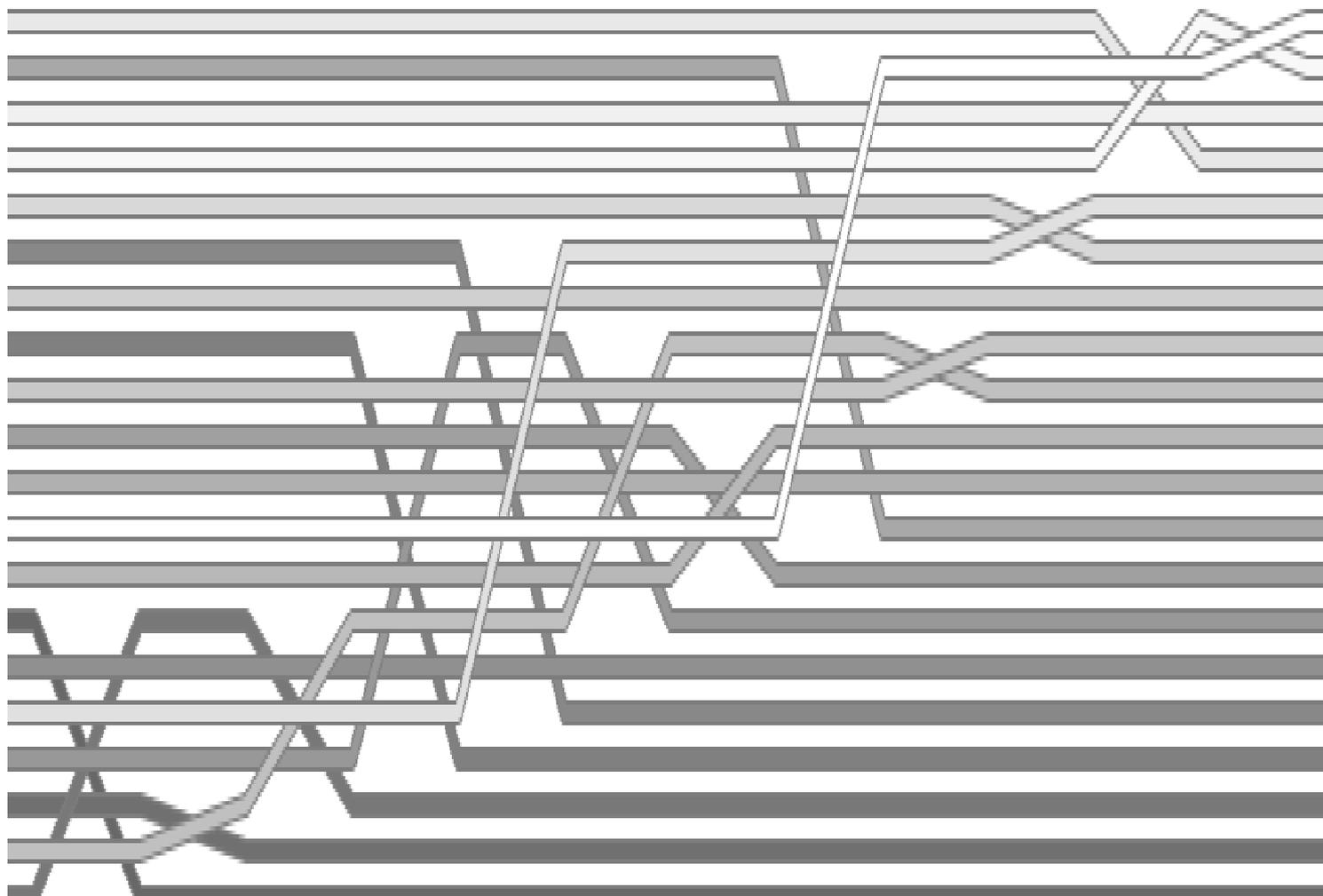
    if (n==1) {
        return;
    }
    else {
        iMin = 0;
        for (j=1; j<n; j=j+1) {
            if (v[iMin]>v[j])
                iMin = j;
        }

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

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



# Selection Sort (<http://sortvis.org>)

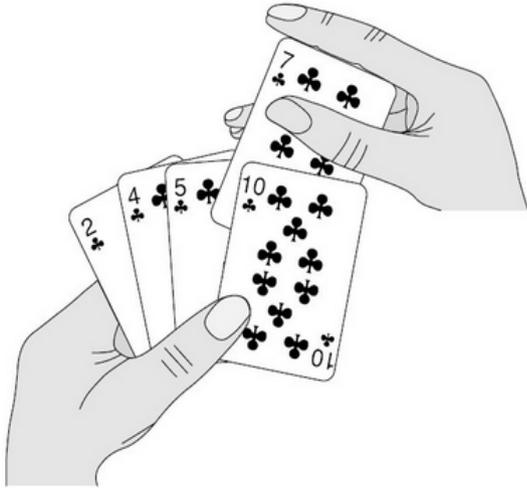


Seleciona o maior elemento em cada iteração



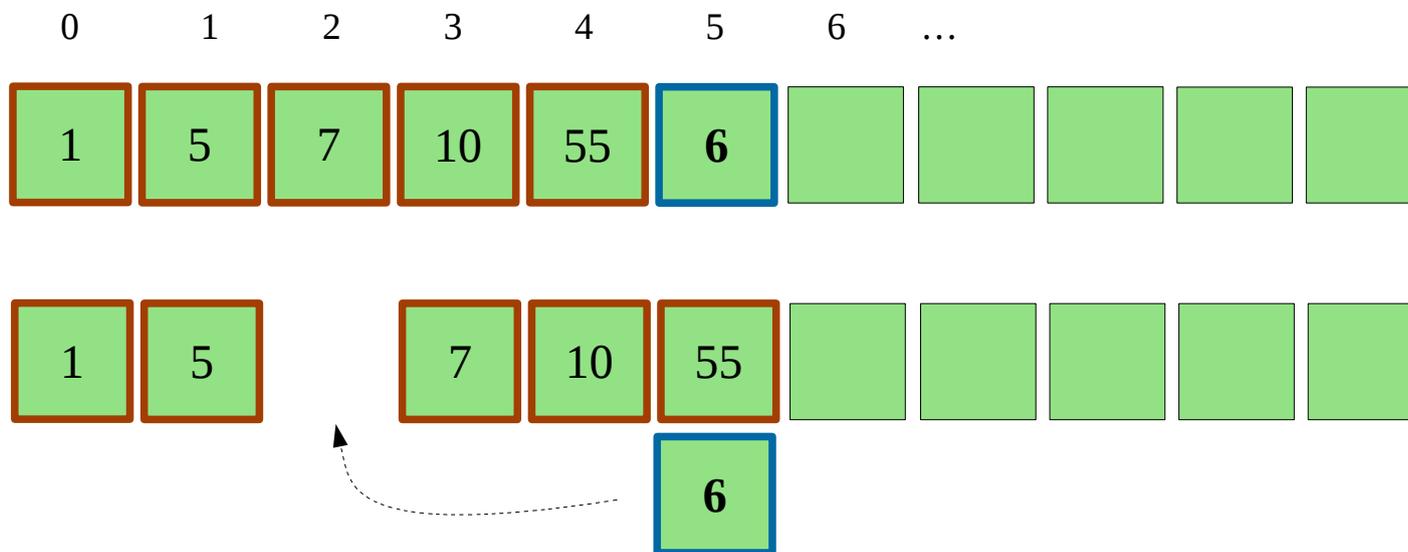
**(3)**  
**Insertion Sort:**  
**Algoritmo de ordenação por inserção**

# Insertion Sort



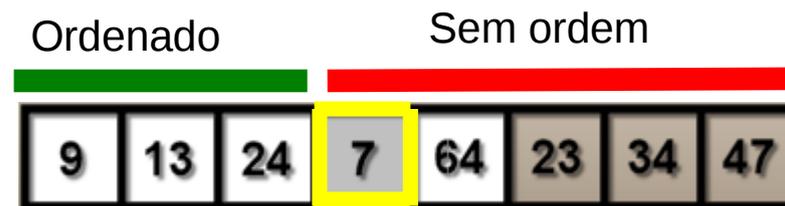
*Método preferido dos jogadores de cartas*

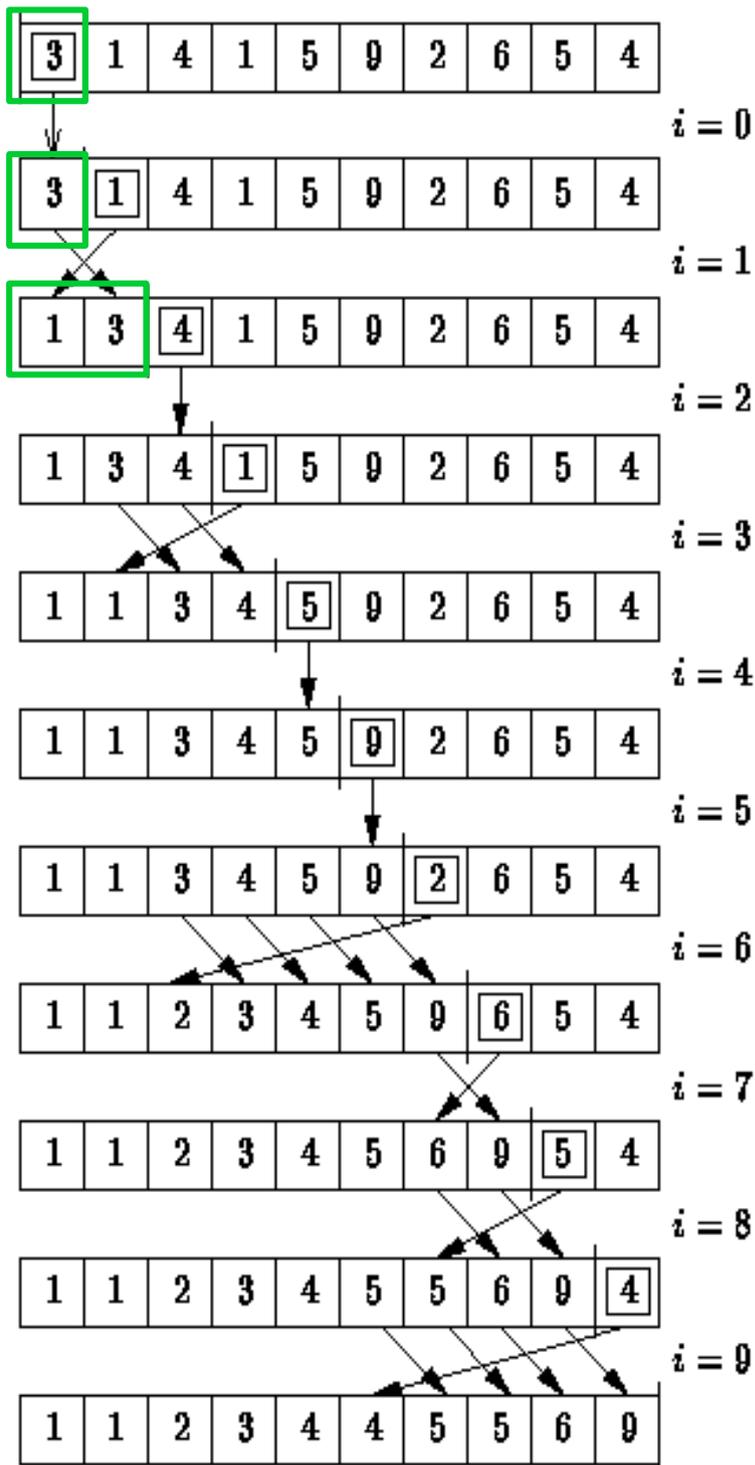
*Em cada passo, a partir do  $i=1$ , o  $i$ -ésimo elemento da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.*



# Insertion Sort

- A principal característica deste algoritmo consiste em ordenar o vetor utilizando um subvetor ordenado em seu início.
- **A cada novo passo, acrescentamos a este subvetor mais um elemento até atingirmos o último elemento de um arranjo.**





```
void InsertionSort (int[] v, int n)
```

# Insertion Sort

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

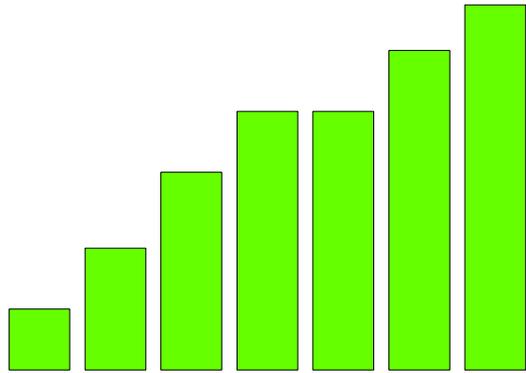
    for (i=1; i<n; i=i+1) {
        aux = v[i];

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

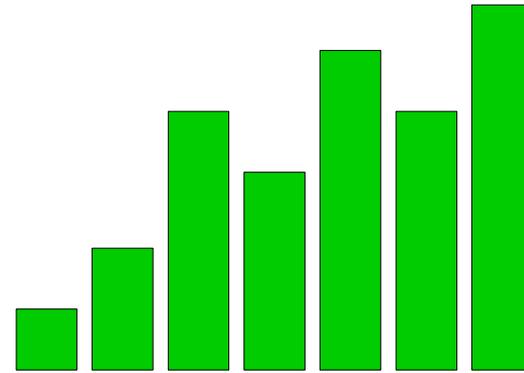
        v[j+1] = aux;
    }
}
```

Pense, no recesso, em uma versão recursiva?

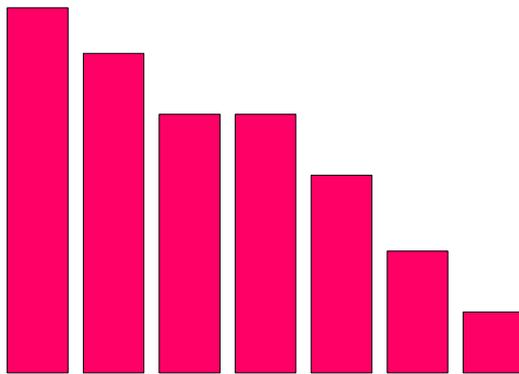
# Insertion Sort: Eficiente para quais sequências?



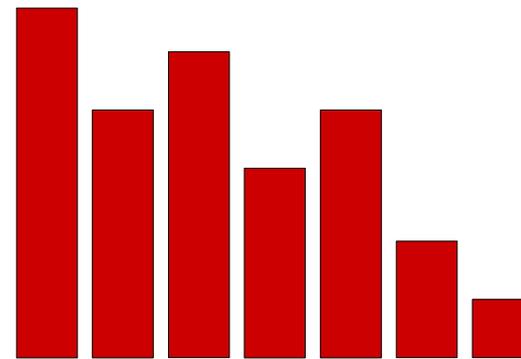
Vetor Crescente



Vetor Parcialmente Crescente

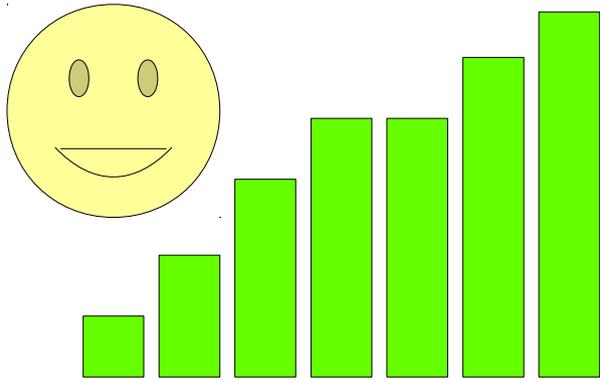


Vetor Decrescente



Vetor Parcialmente Decrescente

# Insertion Sort: Eficiente para quais sequências?



Vetor Crescente



Vetor Parcialmente Crescente



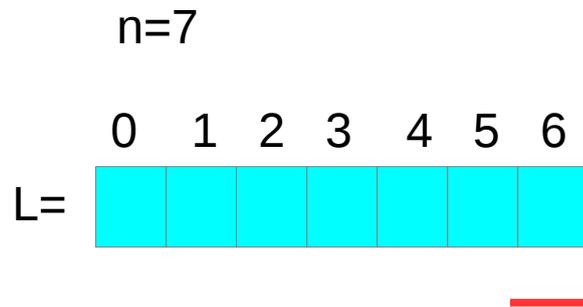
Vetor Decrescente



Vetor Parcialmente Decrescente

Este algoritmo é o mais apropriado quando os elementos do vetor estiverem ordenados ou parcialmente ordenados.

# Insertion Sort



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

$$\text{Tempo} = (n-1)(n)/2$$

$$\text{Tempo} = n^2/2 - n/2$$

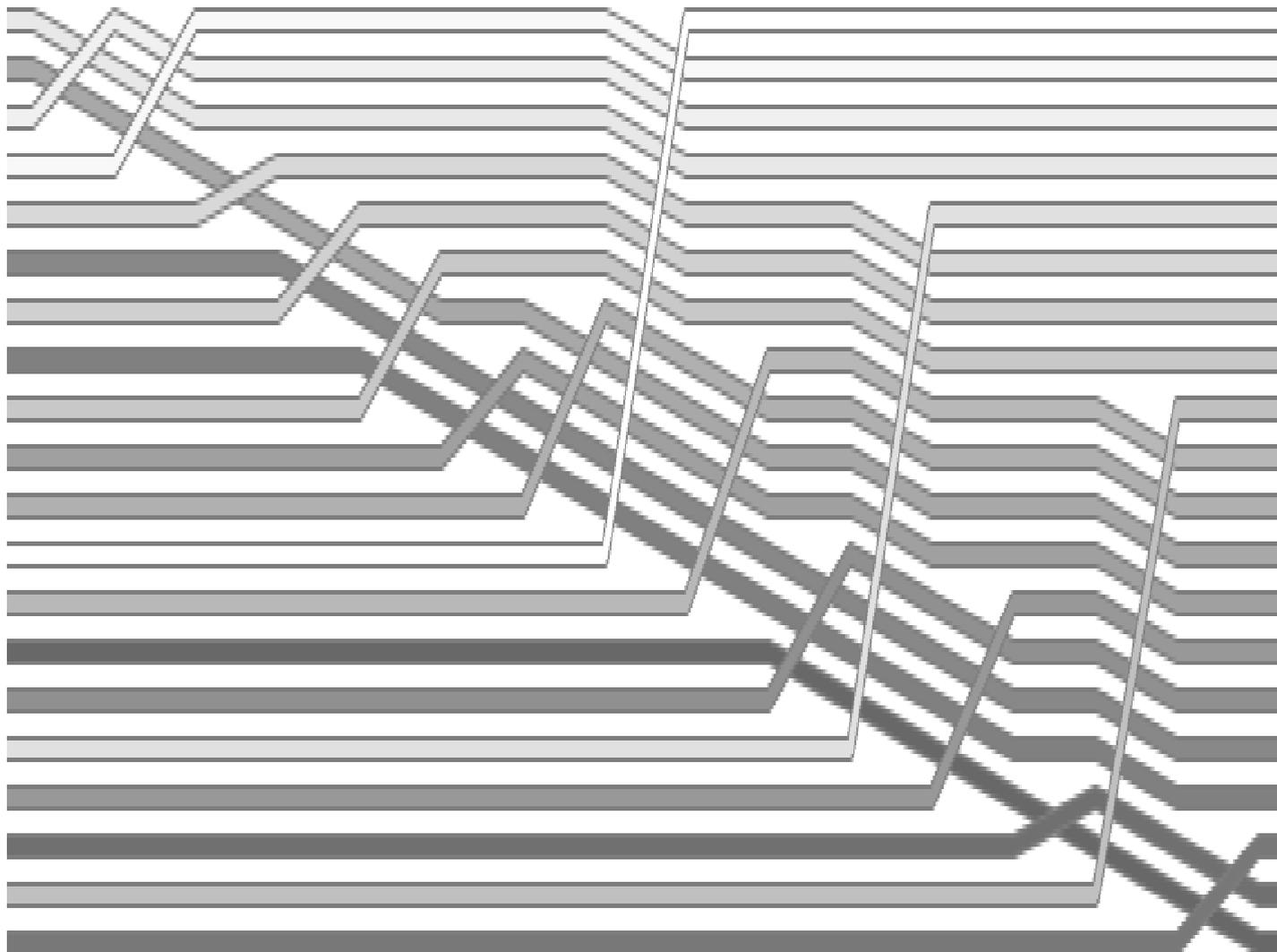
$$\text{Tempo} = O(n^2)$$

Número de comparações  $T(n)$ :

- No melhor caso:  $T(n) = n-1$

- No pior caso:  $T(n) = n^2/2 - n/2$

# Insertion Sort (<http://sortvis.org>)





**(4)**

**Bubble Sort:**

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

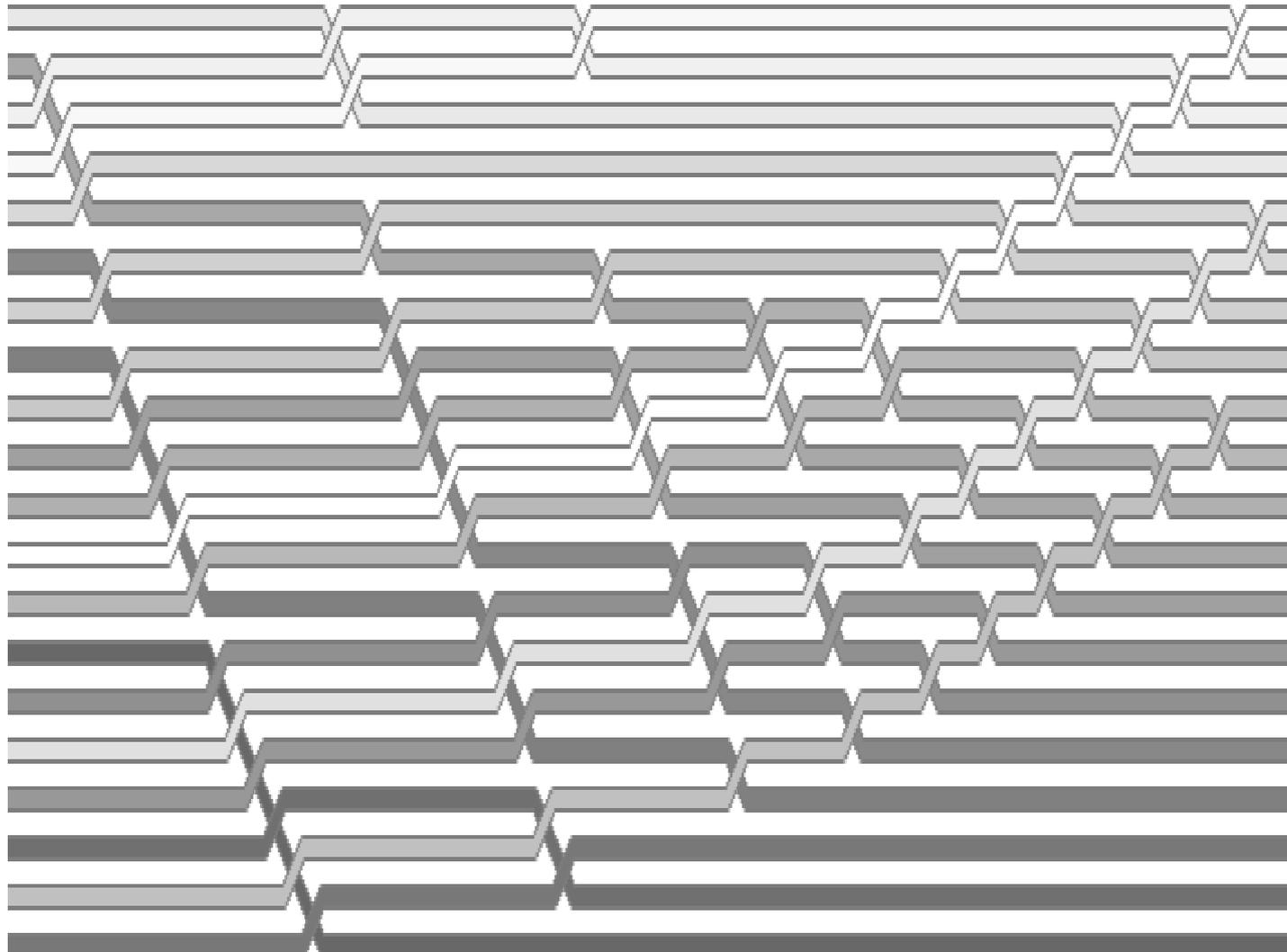


# Bubble Sort

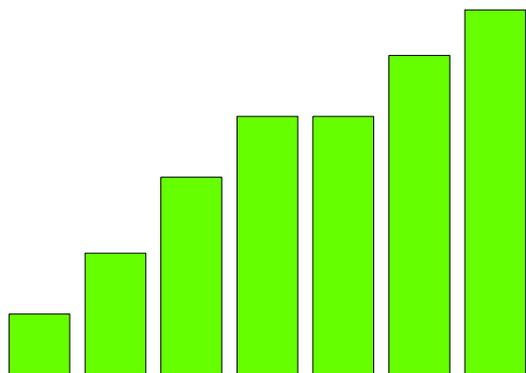
```
void BubbleSort1 (int v[], int n) {
    int k, i, aux;

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

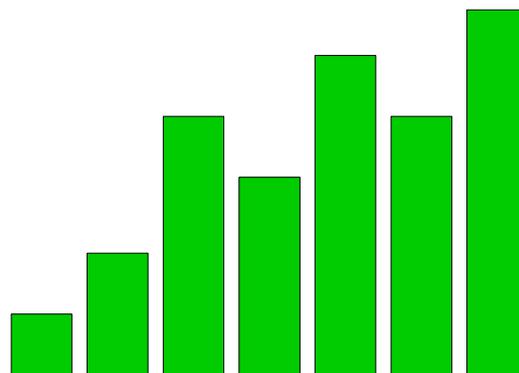
# Bubble Sort (<http://sortvis.org>)



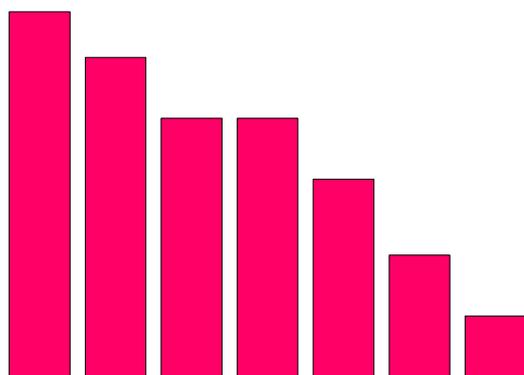
# Bubble Sort: Eficiente para quais sequências?



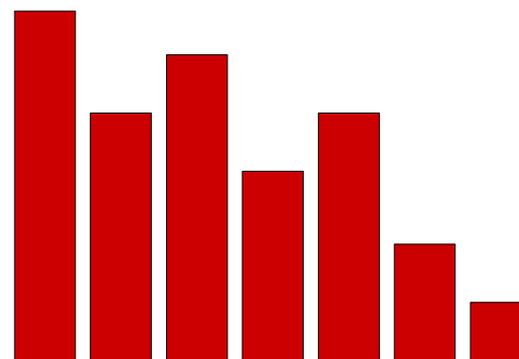
Vetor Crescente



Vetor Parcialmente Crescente



Vetor Decrescente

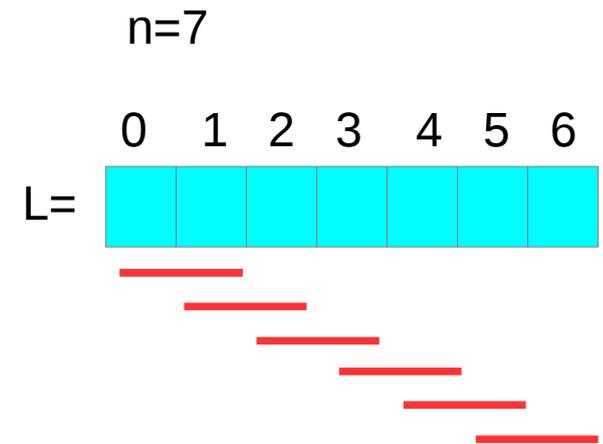


Vetor Parcialmente Decrescente

# Bubble Sort

```
void BubbleSort1 (int v[], int n) {
    int k, i, aux;

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



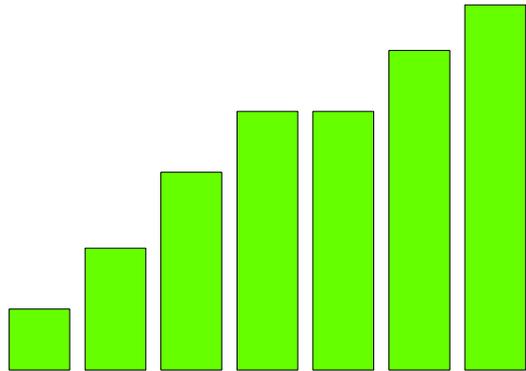
Comparações

k=n-1	n-1
k=n-2	n-2
k=n-3	n-3
...	
k=1	1

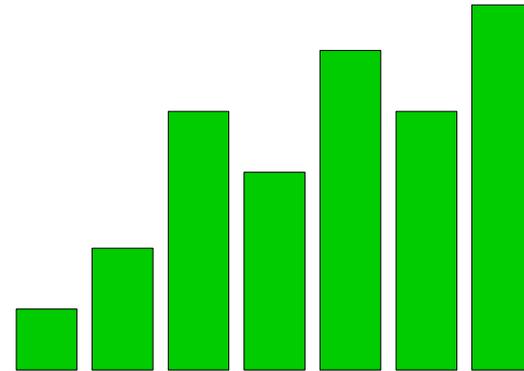
$$\text{Tempo} = (n-1)(n)/2$$

$$\text{Tempo} = n^2/2 - n/2$$

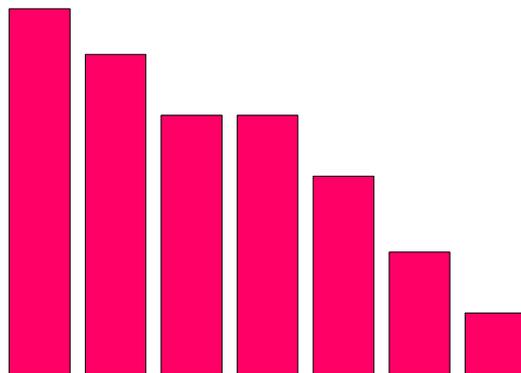
# Bubble Sort: Eficiente para quais sequências?



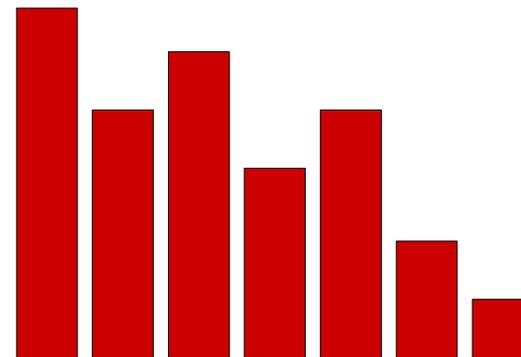
Vetor Crescente



Vetor Parcialmente Crescente



Vetor Decrescente



Vetor Parcialmente Decrescente

Para qualquer sequência o algoritmo custa  $n^2/2 - n/2 \rightarrow O(n^2)$



# Teste de avaliação

# Questão 1

- Uma versão alternativa do algoritmo de Bubble Sort

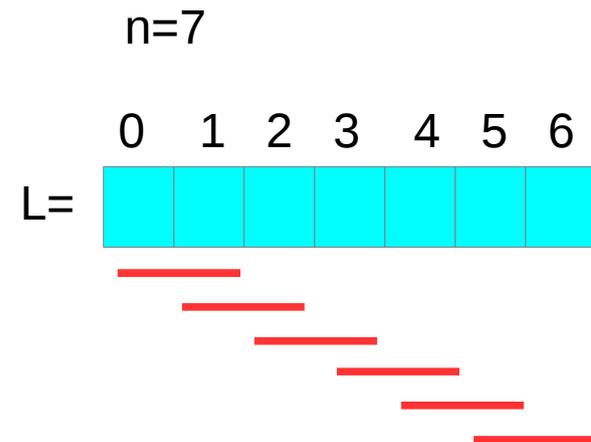
```
void BubbleSort2 (int v[], int n) {
    int i, aux, hasChanged;

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

# Questão 1

- Uma versão alternativa do algoritmo de Bubble Sort

```
void BubbleSort2 (int v[], int n) {  
    int i, aux, hasChanged;  
  
    do {  
        hasChanged = 0;  
        for (i=0; i<n-1; i=i+1) {  
            if (v[i]>v[i+1]) {  
                aux = v[i];  
                v[i] = v[i+1];  
                v[i+1] = aux;  
                hasChanged = 1;  
            }  
        }  
    }while(hasChanged==1);  
}
```



# Questão 1

```
void BubbleSort1 (int v[], int n) {
    int k, i, aux;

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

Número de comparações  $T(n)$ :

- No melhor caso:  $T(n) = n^2/2 - n/2$
- No pior caso:  $T(n) = n^2/2 - n/2$

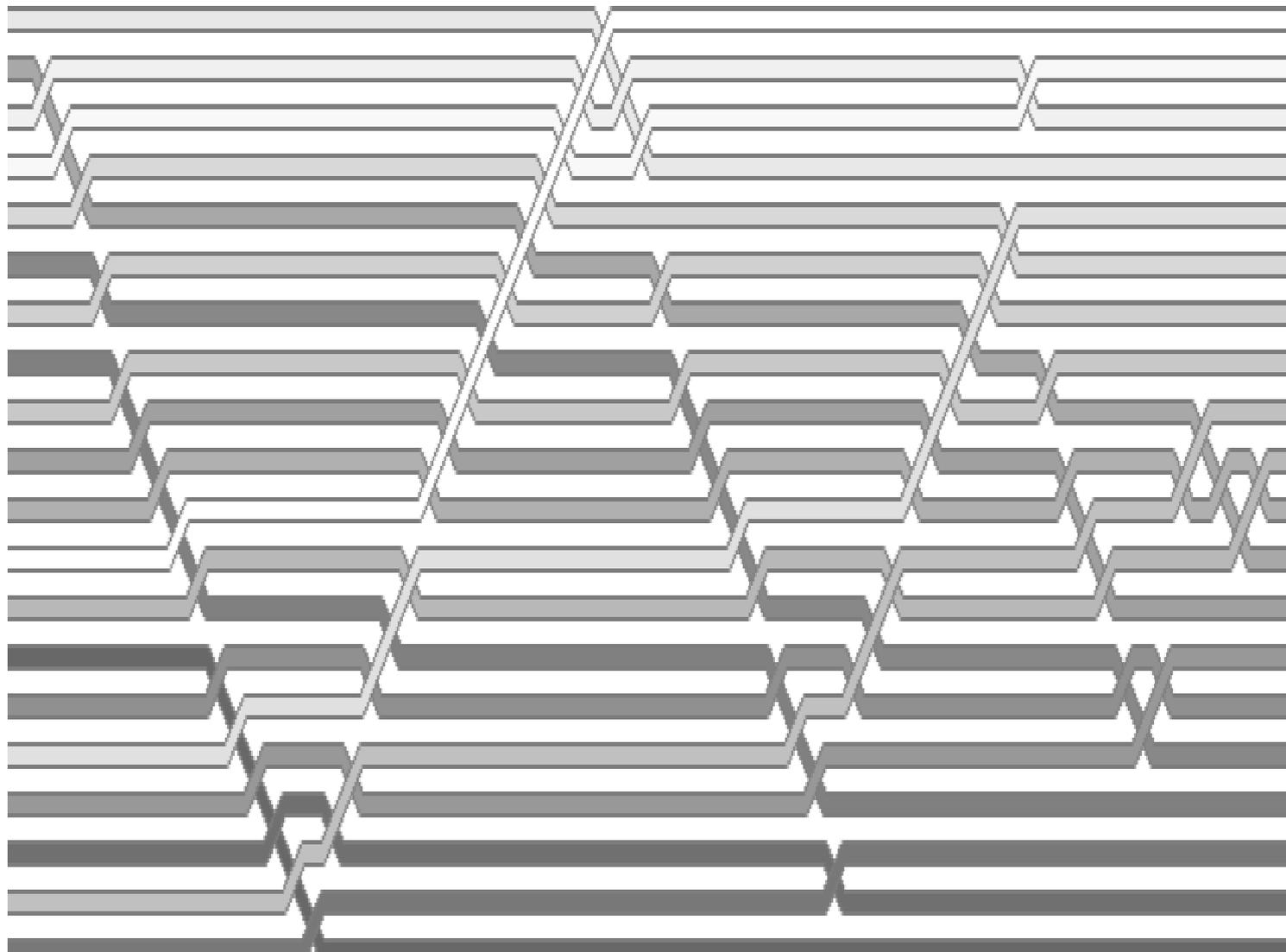
```
void BubbleSort2 (int v[], int n) {
    int i, aux, hasChanged;

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

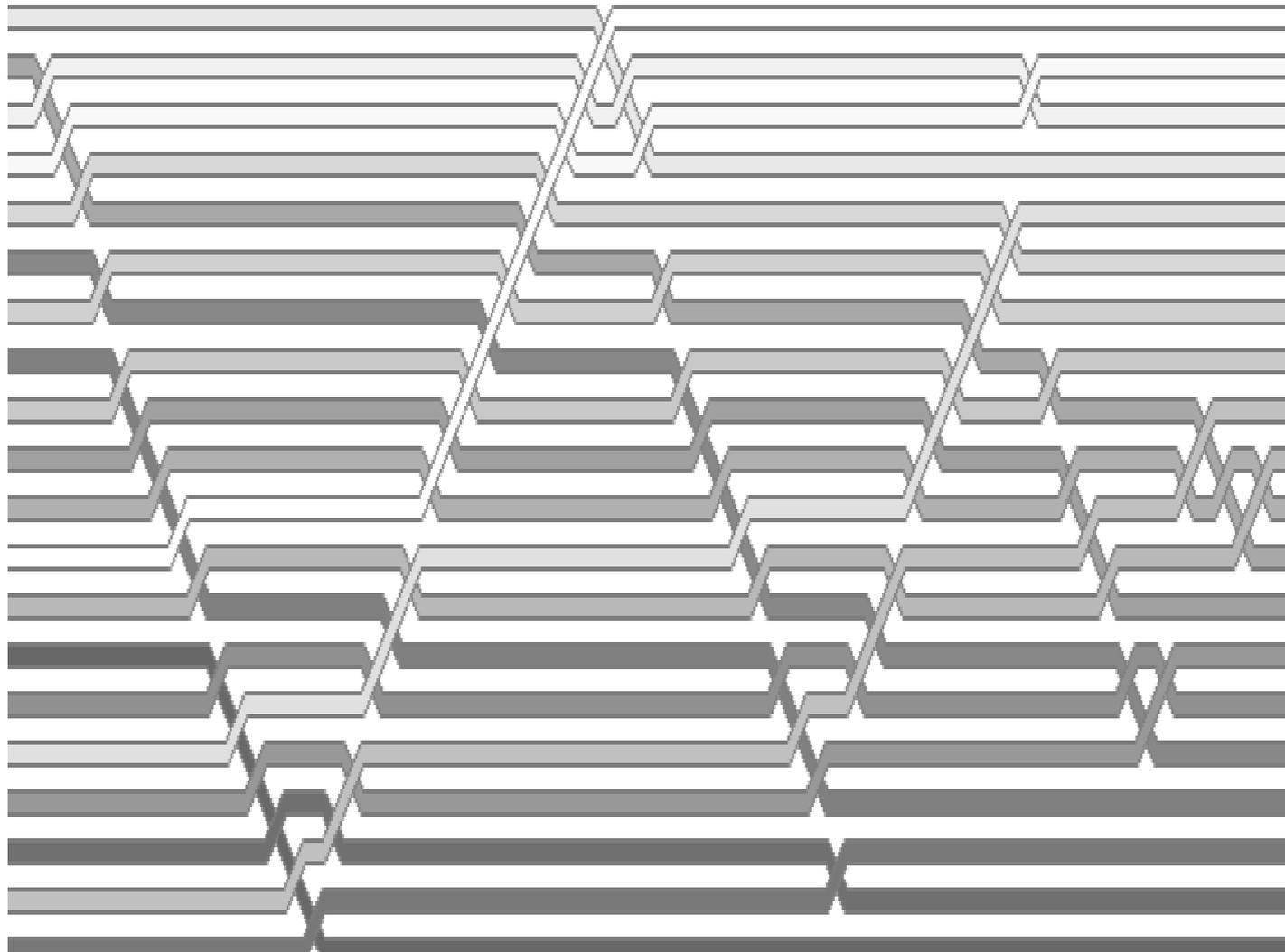
Número de comparações  $T(n)$ :

- No melhor caso:  $T(n) = n-1$
- No pior caso:  $T(n) = n(n-1)$

# Questão 2



# Questão 2: Cocktail Sort



# Questão 2: Cocktail Sort

```
void cocktailSort(int v[], int n) {
    int i, t;
    int trocou = 1; // 1:True, 0:False
    int inicio = 0;
    int fim = n-1;

    while (trocou==1) {
        trocou = 0;
        for (i=inicio; i<fim; i++) {
            if (v[i] > v[i+1]) {
                t = v[i];
                v[i] = v[i+1];
                v[i+1] = t;
                trocou = 1;
            }
        }
        fim--;

        if (trocou==0)
            break;

        trocou = 0;
        for (i=fim-1; i>=inicio; i--) {
            if (v[i] > v[i+1]) {
                t = v[i];
                v[i] = v[i+1];
                v[i+1] = t;
                trocou = 1;
            }
        }
        inicio++;
    }
}
```

# Questão 3

- Tabela com número de comparações necessárias para ordenar uma sequência de  $n$  elementos.

Algoritmo	Melhor caso	Pior caso
Selection sort	$n^2/2 - n/2$	$n^2/2 - n/2$
Insertion sort	$n-1$	$n^2/2 - n/2$
Bubble sort	$n^2/2 - n/2$	$n^2/2 - n/2$
Bubble2 sort	$n-1$	$n^2 - n$
Cocktail sort	$n-1$	???