



# ALGORITMOS

PROF<sup>º</sup>: SIMONE DOMINICO

ORDENAÇÃO DE VETOR

---

## ORDENAÇÃO DE VETORES

- Ordenação de vetores:
  - entrada: vetor com os  $n$  elementos a serem ordenados
  - saída: mesmo vetor com  $n$  elementos na ordem especificada

## ORDENAÇÃO DE VETORES

- Algoritmos:
  1. Ordenação por Inserção
  2. Ordenação por Seleção
  3. BubbleSort
  4. ShellSort
  5. MergeSort
  6. Ordenação por Partição (QuickSort)
  7. BucketSort

## ORDENAÇÃO DE VETORES – INSERÇÃO

- A ideia da ordenação por inserção é dividir os elementos em duas subestruturas, uma com os elementos já ordenados e outra com elementos ainda por ordenar.



## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
3	1	9	5	2	8

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
3	1	9	5	2	8

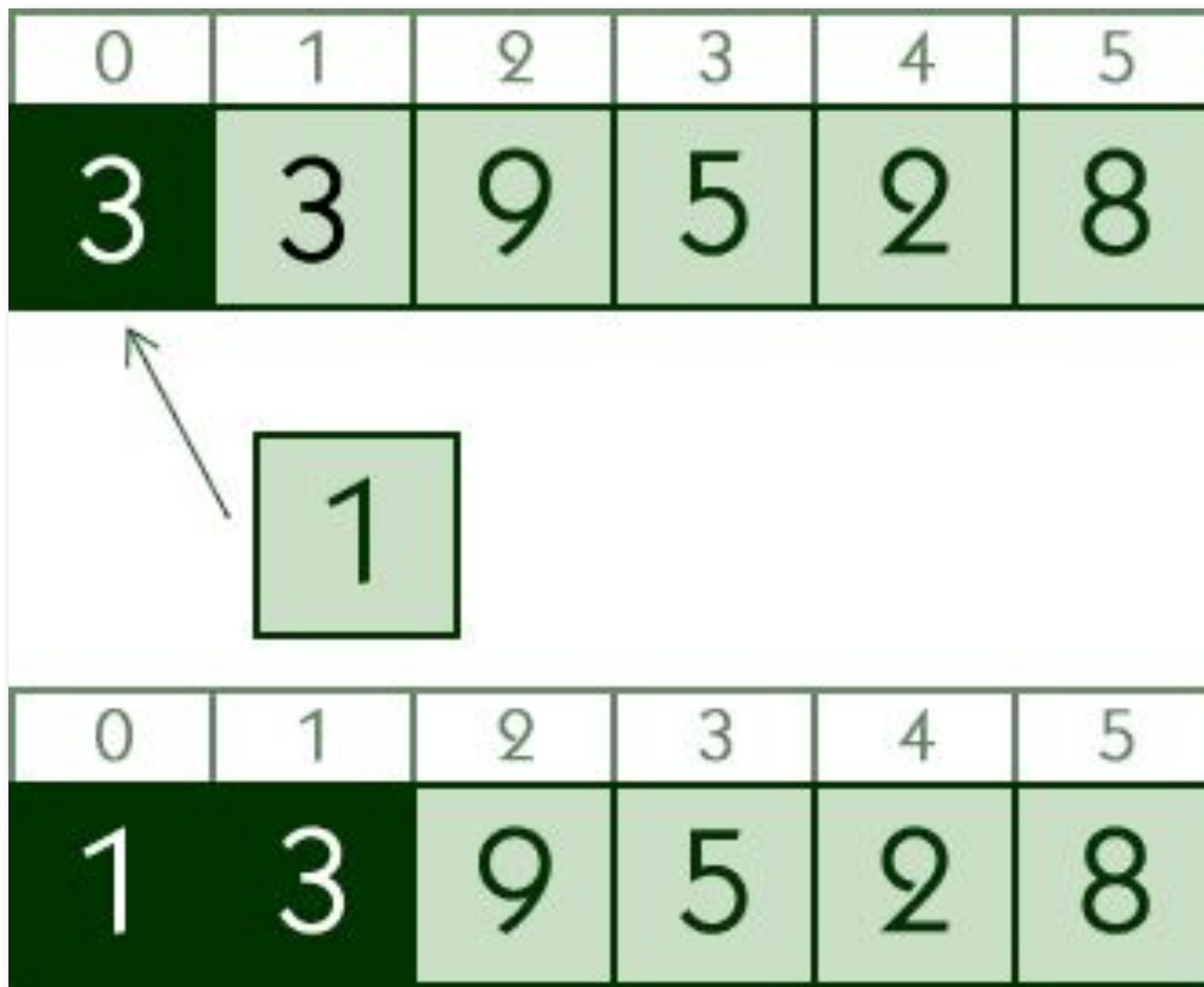
É maior?

0	1	2	3	4	5
3	1	9	5	2	8

0	1	2	3	4	5
3	3	9	5	2	8

- Inicialmente, temos:  
 $i = 1$   
 $j = 0$   
 $\text{atual} = 1.$
- Queremos inserir elemento na parte ordenado à esquerda.
- Como  $3 > 1$ , então deslocamos o 3 para direita e decrementamos  $j$ :
- $j = 0 - 1 = -1$

## ORDENAÇÃO DE VETORES – INSERÇÃO



- Como atingimos o início do vetor, então inserimos 1 em  $j + 1 = 0$

## ORDENAÇÃO DE VETORES – INSERÇÃO


0	1	2	3	4	5
1	3	9	5	2	8

É maior?

- Agora:  
     $i = 2$   
     $j = 1$   
    atual = 9.
- Como  $3 < 9$ ,  
então o  
elemento já  
está na  
posição  
correta




## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	3	9	5	2	8
					
É maior?					
0	1	2	3	4	5
1	3	9	9	2	8

- Na terceira iteração  
 $i = 3$   
 $j = 2$   
atual = 5.
- Como  $9 > 5$ , então deslocamos o 9 para direita e
- decrementamos  $j$
- $j = 2 - 1 = 1$

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	3	9	9	2	8

0	1	2	3	4	5
1	3	5	9	2	8


- Como  $3 < 5$ ,  
então  
inserimos  
5 em  $j + 1 = 2$

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	3	5	9	2	8
É maior?					
0	1	2	3	4	5
1	3	5	9	9	8


- Na quarta iteração:  
     $i = 4$ ,  
     $j = 3$   
    atual = 2.
- Como  $9 > 2$ ,  
então deslocamos o 9 para direita
- decrementamos  $j$ :  $j = 3 - 1 = 2$

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	3	5	9	9	8
<p>É maior que o 2?</p> 					
0	1	2	3	4	5
1	3	5	5	9	8

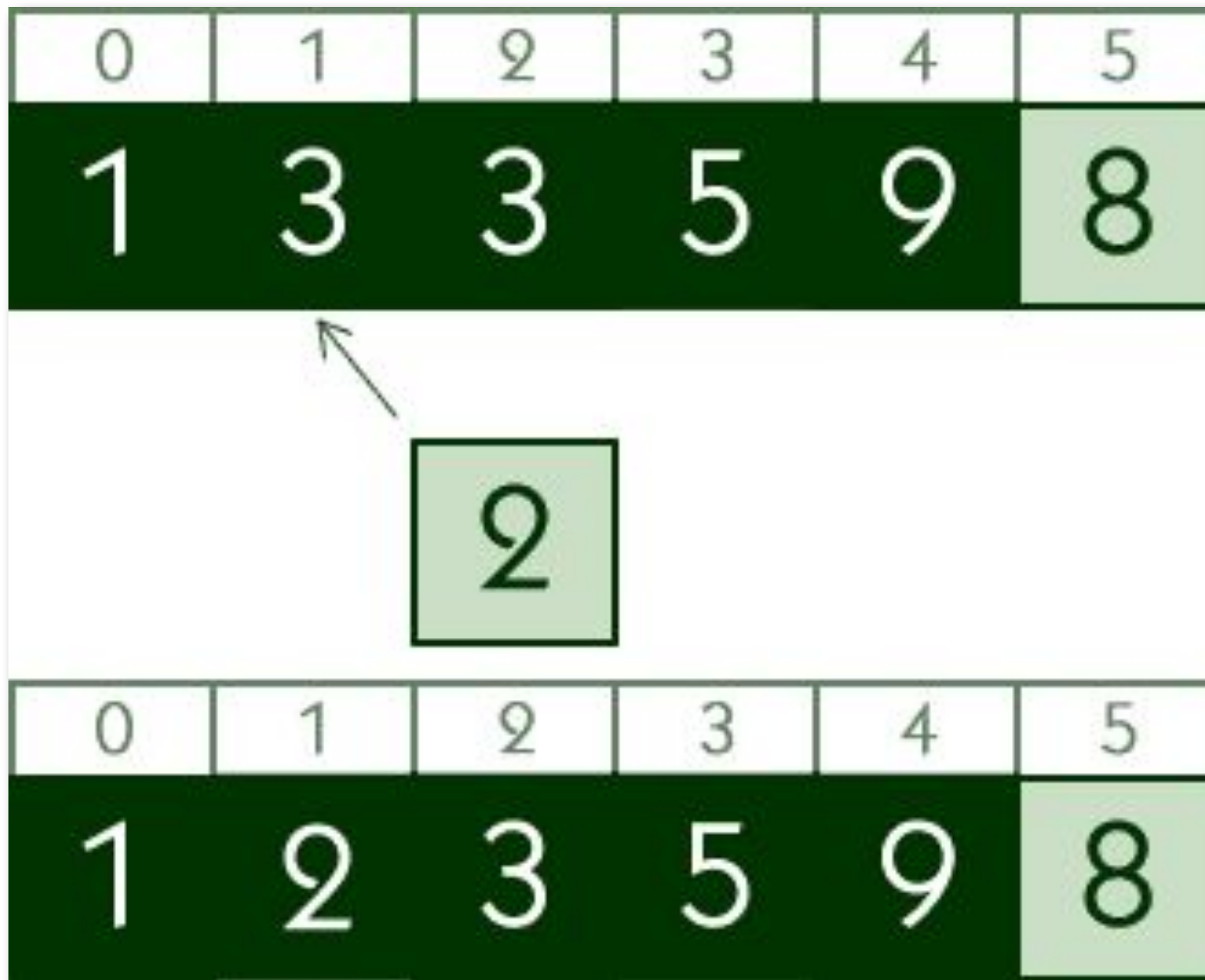
- Como  $5 > 2$ , então deslocamos o 5 para direita
- decrementamos  $j$ :  $j = 2 - 1 = 1$

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	3	5	5	9	8
 É maior que o 2?					
0	1	2	3	4	5
1	3	3	5	9	8

- Como  $3 > 2$ , então deslocamos o 3 para direita e decrementamos os  $j$ :  $j = 1 - 1 = 0$

## ORDENAÇÃO DE VETORES – INSERÇÃO



- Como  $1 < 2$ , então inserimos 2 em
- $j + 1 = 1$



## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	2	3	5	9	8
É maior?					
0	1	2	3	4	5
1	2	3	5	9	9

• Na quinta e última iteração:

$i = 5$

$j = 4$

atual = 8.

- Como  $9 > 8$ , então deslocamos o 9 para direita e decrementamos
- $j: j = 4 - 1 = 3$

## ORDENAÇÃO DE VETORES – INSERÇÃO

0	1	2	3	4	5
1	2	3	5	9	9

8

- Como  $5 < 8$ , então inserimos 8 em
- $j + 1 = 4$

0	1	2	3	4	5
1	2	3	5	8	9



## ALGORITMO EM C

```
6 void insertionSort(int n, int *A){
7     int i, atual, j;
8     for(i=1;i < n;i++){
9         atual = A[i];
10        j = i-1;
11        while ((j>=0)&& (A[j]>atual))
12        {
13            A[j+1] = A[j];
14            j = j - 1;
15        }
16        A[j+1] = atual;
17    }
18 }
```

## ALGORITMO EM C

```
int main(){
    int vet[size]={29,3,65,7,8}, i;
    insertionSort(size, vet);
    for(i=0;i<size;i++){
        printf("[ %d ] ", vet[i]);
    }
    return 0;
}
```

## ORDENAÇÃO DE VETORES – SELEÇÃO

- A ideia da ordenação por seleção é procurar o menor elemento do vetor (ou maior) e movimentá-lo para a primeira (última) posição do vetor.
- Repetir para os  $n$  elementos do vetor.



# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
3	1	9	5	2	8

- Na primeira iteração, procuramos o menor valor entre os índices 0 e 5 (índice da última posição). O menor valor é o 1, que está no índice 1. Então, trocamos os valores dos índices 0 e 1

# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
1	3	9	5	2	8

- Na segunda iteração, procuramos o menor valor entre os índices 1 e 5. O menor valor é o 2, que está no índice 4. Então, trocamos os valores dos índices 1 e 4

# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
1	2	9	5	3	8

- Na terceira iteração, procuramos o menor valor entre os índices 2 e 5. O menor valor é o 3, que está no índice 4. Então, trocamos os valores dos índices 2 e 4:

# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
1	2	3	5	9	8

- Na quarta iteração, procuramos o menor valor entre os índices 3 e 5. O menor valor é o 3, que está no índice 3. Então, não trocamos.



# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
1	2	3	5	9	8

- Na quinta iteração, procuramos o menor valor entre os índices 4 e 5. O menor valor é o 8, que está no índice 5. Então, trocamos os valores dos índices 4 e 5



# ORDENAÇÃO DE VETORES – SELEÇÃO

0	1	2	3	4	5
1	2	3	5	8	9

- Agora, nos resta apenas um subvetor com um único elemento [9], que obviamente já está ordenado.

## ALGORITMO EM C

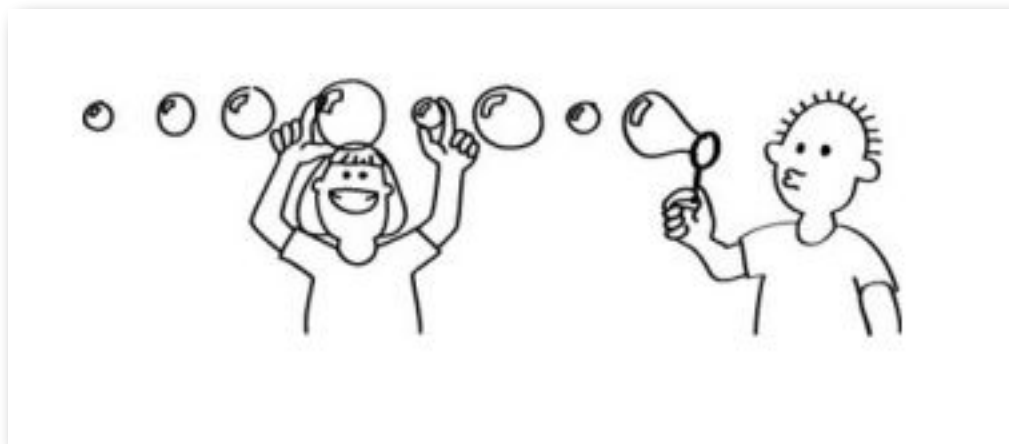
```
19 void selectionSort(int vet[], int tam){
20     int i, j, min;
21     for(i=0; i<tam; i++){
22         min = i;
23         for(j = i+1; j<tam; j++){//Acha posicao do menor elemento a
24             if(vet[min] > vet[j])
25                 min = j;
26         }
27         troca(&vet[i], &vet[min]);
28     }
29 }
```

## ALGORITMO EM C

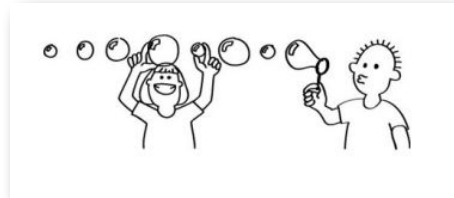
```
30 void troca(int *a, int *b){
31     int aux;
32     aux = *a;
33     *a = *b;
34     *b = aux;
35 }
36 int main(){
37     int vet[size]={29,3,65,7,8}, i;
38     selectionSort(vet, size);
39     for(i=0;i<size;i++){
40         printf("[ %d ] ", vet[i]);
41     }
42     return 0;
```

## ORDENAÇÃO DE VETORES – BUBBLESORT

- A idéia da ordenação por bolhas é flutuar o maior elemento para o fim.
- Repita n vezes a flutuação.

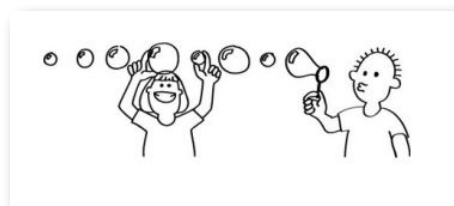


## ORDENAÇÃO DE VETORES – BUBBLESORT

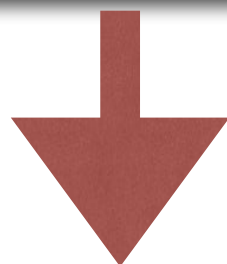


- A ideia da ordenação por bolhas é flutuar o maior elemento para o fim.
- Repita n vezes a flutuação.
- A ideia básica do algoritmo é realizar a troca de elementos em posições adjacentes (um ao lado do outro), de modo que os elementos com maior valor sejam posicionados no final do vetor (array) e os com menor valor sejam posicionados no início do vetor.

## ORDENAÇÃO DE VETORES – BUBBLESORT



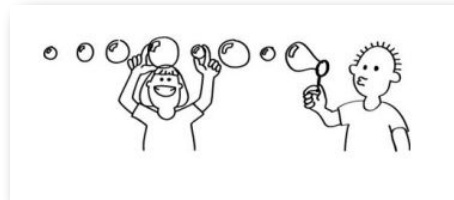
0	1	2	3	4	5
3	1	9	5	2	8



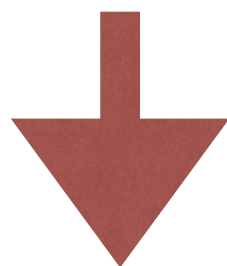
0	1	2	3	4	5
1	3	9	5	2	8

Vamos comparar o elemento no índice 0 com o elemento imediatamente posterior, isto é, vamos comparar os valores 3 e 1. Como 1 é menor que 3, então precisamos trocá-los de posição.

## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	9	5	2	8

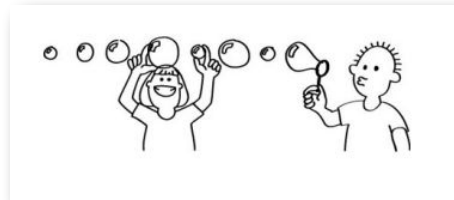


0	1	2	3	4	5
1	3	9	5	2	8

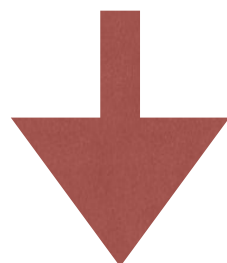
Continuando, compararemos o elemento do índice 1 com o elemento do índice 2. Como 9 é maior que 3, então não trocamos suas posições.



## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	9	5	2	8

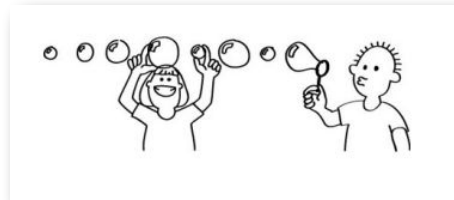


0	1	2	3	4	5
1	3	5	9	2	8

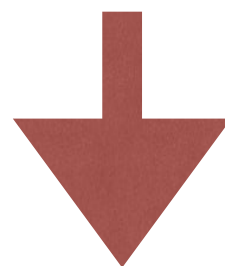
A comparação agora é entre 9 e 5, que resultará em outra troca



## ORDENAÇÃO DE VETORES – BUBBLESORT



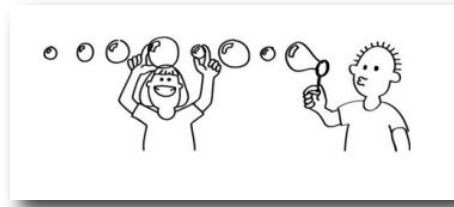
0	1	2	3	4	5
1	3	5	9	2	8



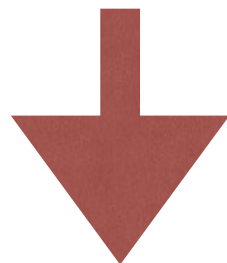
0	1	2	3	4	5
1	3	5	2	9	8

A comparação agora é entre 9 e 2, que resultará em outra troca

## ORDENAÇÃO DE VETORES – BUBBLESORT



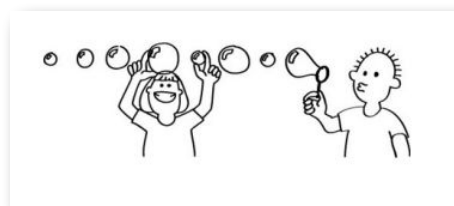
0	1	2	3	4	5
1	3	5	2	9	8



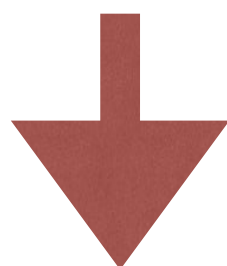
0	1	2	3	4	5
1	3	5	2	8	9

A comparação agora é entre 9 e 8, que resultará em outra troca.

## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	5	2	9	8



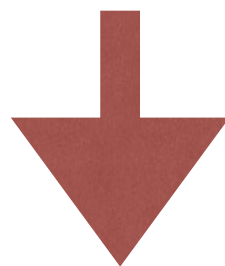
0	1	2	3	4	5
1	3	5	2	8	9

A comparação agora é entre 9 e 8, que resultará em outra troca. Observe que 9 era o maior elemento e, após percorrer o vetor, ele foi posicionado na última posição.

## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	5	2	8	9

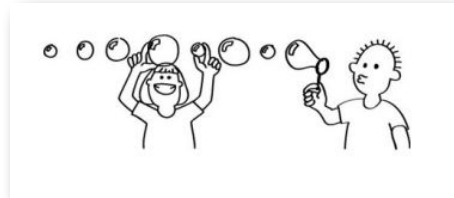


0	1	2	3	4	5
1	3	5	2	8	9

Vamos repetir o procedimento. Comparamos 1 com 3 (repare que voltamos ao primeiro índice), porém não há troca a ser realizada, pois  $1 < 3$ .

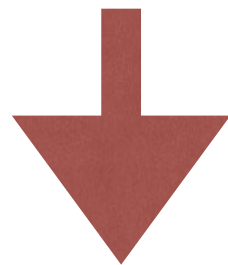
Comparamos 3 com 5 e, como  $3 < 5$ , não trocamos os valores de posição

## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	5	2	8	9

Comparamos 5 com 2 e, como  $5 > 2$ , trocamos os valores de posição.

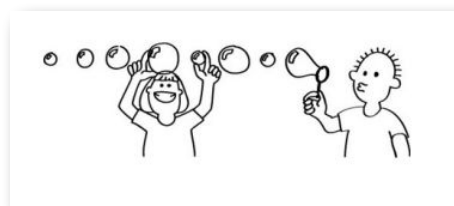


0	1	2	3	4	5
1	3	2	5	8	9

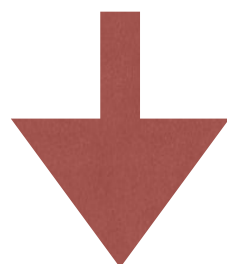
Comparamos 5 com 8 e, como  $5 < 8$ , não trocamos os valores de posição.  
O mesmo com 8 e 9.



## ORDENAÇÃO DE VETORES – BUBBLESORT



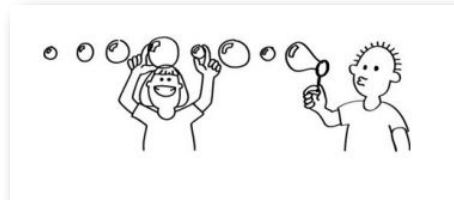
0	1	2	3	4	5
1	3	5	2	8	9



0	1	2	3	4	5
1	3	2	5	8	9

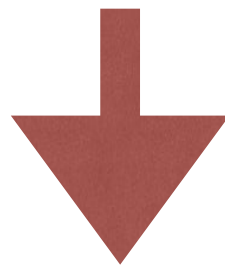
Vamos repetir o procedimento. Comparamos 1 com 3 (repare que voltamos ao primeiro índice), porém não há troca a ser realizada, pois  $1 < 3$ .

## ORDENAÇÃO DE VETORES – BUBBLESORT



0	1	2	3	4	5
1	3	2	5	8	9

Comparamos 3 com 2, trocamos de posição, pois  $3 > 2$ .



0	1	2	3	4	5
1	2	3	5	8	9

## ALGORITMO EM C

```
37 void bubble_sort (int vetor[], int n) {
38     int k, j, aux;
39
40     for (k = 0; k < n - 1; k++) {
41         for (j = 0; j < n - k - 1; j++) {
42             if (vetor[j] > vetor[j + 1]) {
43                 aux = vetor[j];
44                 vetor[j] = vetor[j + 1];
45                 vetor[j + 1] = aux;
46             }
47         }
48     }
49 }
```



## ORDENAÇÃO DE VETORES – SHELLSORT

- É uma extensão do algoritmo de ordenação por inserção.
- Problema com o algoritmo de ordenação por inserção:
- Troca itens adjacentes para determinar o ponto de inserção.
- São efetuadas  $n - 1$  comparações e movimentações quando o menor item está na posição mais à direita no vetor.
- O método de Shell contorna este problema permitindo trocas de registros distantes um do outro.

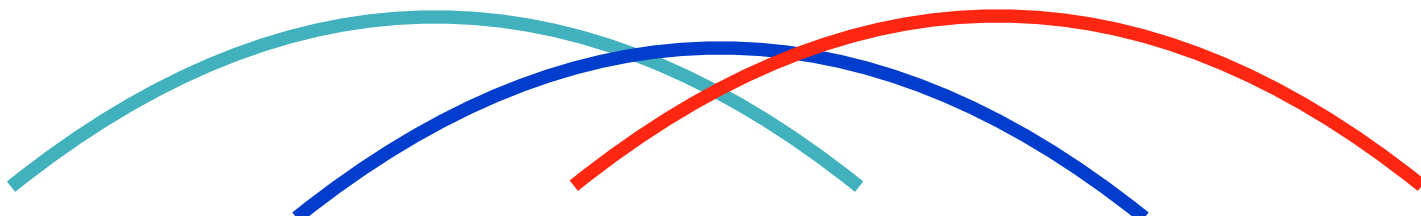
## ORDENAÇÃO DE VETORES – SHELLSORT

- Faz trocas a uma certa distância (que diminuiu a cada iteração)
- Compara elementos separados por "h" posições e os ordena, vai diminuindo a distância de comparações até  $h=1$

## ORDENAÇÃO DE VETORES – SHELLSORT

- Faz trocas a uma certa distância (que diminuiu a cada iteração)
- Compara elementos separados por "h" posições e os ordena, vai diminuindo a distância de comparações até  $h=1$

# ORDENAÇÃO DE VETORES – SHELLSORT

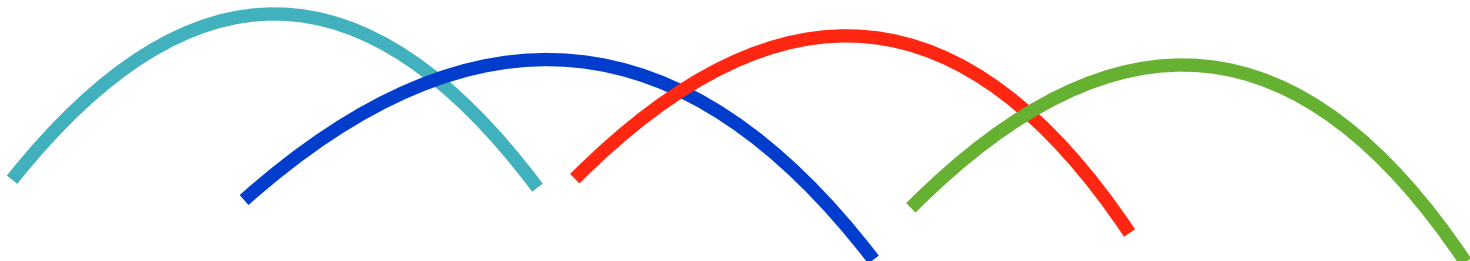


0	1	2	3	4	5
3	1	9	5	2	8

$$h = n/2$$
$$h = 3$$

0	1	2	3	4	5
3	1	8	5	2	9

# ORDENAÇÃO DE VETORES – SHELLSORT



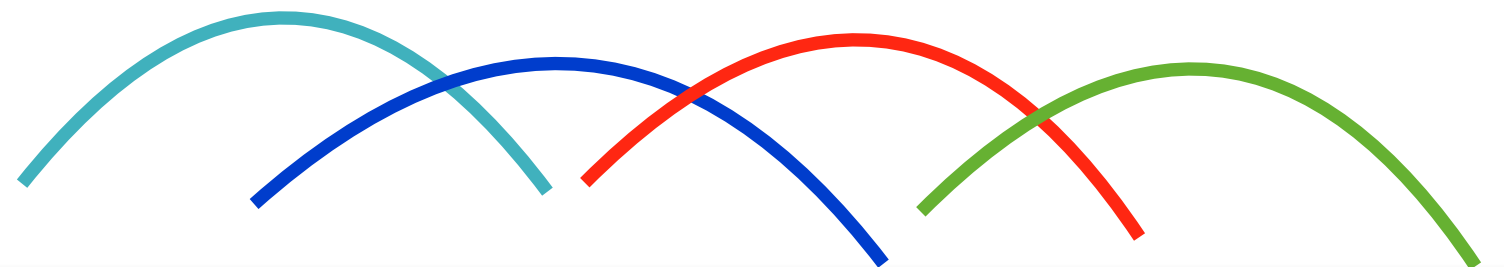
0	1	2	3	4	5
3	1	8	5	2	9

$$h = n/2$$

$$h = 2$$

0	1	2	3	4	5
3	1	2	5	8	9

## ORDENAÇÃO DE VETORES – SHELLSORT



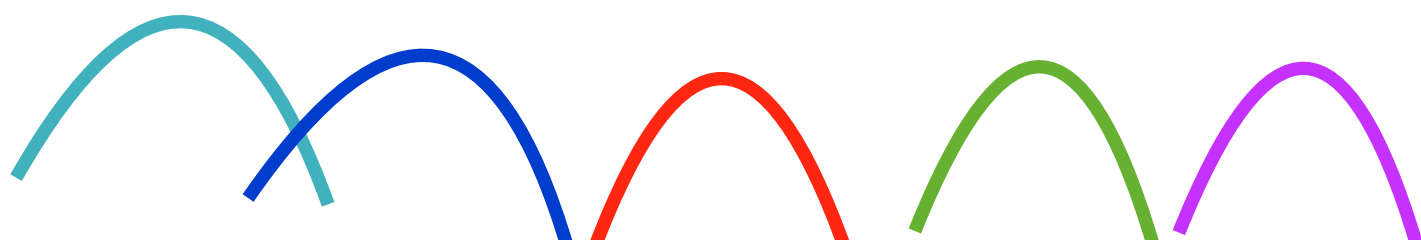
0	1	2	3	4	5
3	1	2	5	8	9

$$h = n/2$$

$$h = 2$$

0	1	2	3	4	5
2	1	3	5	8	9

## ORDENAÇÃO DE VETORES – SHELLSORT



0	1	2	3	4	5
2	1	3	5	8	9

$$h = n/2$$

$$h = 1$$

0	1	2	3	4	5
1	2	3	5	8	9



## ALGORITMO EM C

```
51 void shell_sort(int *vetor, int tamanho)
52 {
53     int i = (tamanho - 1) / 2;
54     int chave, k, aux;
55     while (i != 0)
56     {
57         do
58         {
59             chave = 1;
60             for (k = 0; k < tamanho - i; ++k)
61             {
62                 if (vetor[k] > vetor[k + i])
63                 {
64                     aux = vetor[k];
65                     vetor[k] = vetor[k + i];
66                     vetor[k + i] = aux;
67                     chave = 0;
68                 }
69             }
70         } while (chave == 0);
71         i = i / 2;
72     }
73 }
```

## EXERCÍCIOS

1. Elabore um programa que leia  $n$  letras e ordene-os pelo tamanho utilizando o algoritmo da seleção. No final, o algoritmo deve mostrar todos os nomes ordenados.
2. Crie um programa que dado um vetor de caractere, coloque as letras em ordem crescente pelo algoritmo shell-sort.
3. Crie um programa que dado um vetor de caractere, coloque as letras em ordem crescente pelo algoritmo da bolha.