


# Método de Ordenação BubbleSort



Professor Marcelo Módolo  
Universidade Metodista de São Paulo

## Ordenação

- Ordenar um conjunto de dados é colocar esses dados em ordem crescente ou decrescente
- A ordenação de um conjunto de dados facilita a busca e diminui a complexidade das operações de busca nesse conjunto
- Cada item do conjunto de dados é chamado **registro**
- Cada registro contém uma **chave**, que é o valor a ser ordenado

# Ordenação

**Ordenar é colocar em ordem algo desordenado**

77	42	35	12	101	5
----	----	----	----	-----	---



5	12	35	42	77	101
---	----	----	----	----	-----

## Ordenação Interna e Externa

- **Interna:** os registros que serão classificados estão na memória principal
- **Externa:** os registros que serão classificados estão em algum dispositivo de armazenamento auxiliar, como, disco rígido, CD, etc.
- A ordenação externa somente é utilizada quando não é possível armazenar o conjunto de dados na memória
- **IMPORTANTE:** todos métodos de ordenação que serão estudados nesse módulo têm Ordenação Interna

## Busca x Ordenação

- A primeira decisão sobre complexidade é se ela é necessária para o número de registros que será buscado
- Na maioria das situações é mais eficiente buscar poucos elementos em um conjunto de dados desordenado que ordenar esse conjunto para depois realizar a busca de um único elemento
- Quando muitas buscas sucessivas são necessárias, geralmente ordenar o conjunto de dados antes de realizar essas buscas é mais eficiente

## Complexidade dos Métodos de Ordenação

- Caso a ordenação seja necessária, a decisão será sobre qual método de ordenação utilizar
- Não existe método de ordenação mais eficiente para todos os conjuntos de dados
- Alguns fatores devem ser considerados para definição do método mais eficiente:
  - Função de complexidade do algoritmo ( $f(n)$ )
  - Espaço de memória utilizado na ordenação
  - Tamanho do conjunto de dados
  - Ordenação inicial do conjunto de dados

## Função de Complexidade dos Algoritmos de Ordenação ( $f(n)$ )



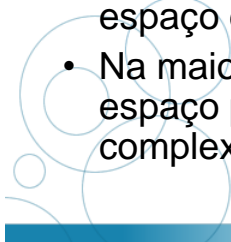
- Para encontrar a função de complexidade ( $f(n)$ ) de um método vamos considerar somente o número de iterações do algoritmo, abstraindo questões como atribuição de valores, comparações entre valores, etc.
- Geralmente a complexidade dos algoritmos de ordenação variam de  $O(n \log n)$  até  $O(n^2)$



## Espaço de memória utilizado na ordenação



- O espaço de memória necessário é medido em função das estruturas de dados auxiliares que o método cria durante a ordenação
- Alguns métodos precisam criar vetores temporários para a ordenação enquanto outros utilizam somente algumas variáveis
- Geralmente algoritmos que ocupam menos espaço demandam mais processamento
- Na maioria dos casos as considerações de espaço pesam menos que as demais para a complexidade do método



## Tamanho do conjunto de dados

- O tamanho do conjunto de dados pode influenciar diretamente na complexidade do método
- Alguns métodos podem ser muito eficientes para pequenos e médios conjuntos de dados, mas ineficientes para conjuntos grandes
- Outros métodos podem comportar-se de maneira inversa, sendo mais eficientes para grandes conjuntos de dados

## Ordenação inicial do conjunto de dados

- A complexidade de um método pode variar muito para um conjunto de dados que já esteja praticamente ordenado ou em ordem totalmente inversa
- Um método mais eficiente para conjunto de dados praticamente ordenados deve ser utilizado para reordenar um conjunto de dados que são inseridos poucos dados a cada vez
- Um método mais eficiente para um conjunto totalmente desordenado deve ser utilizado para ordenar novos conjuntos de dados

## Escolha do método mais eficiente

- A primeira medida de complexidade de um método pode ser calculada pela função de complexidade do algoritmo ( $f(n)$ )
- A partir dessa medida podem ser selecionados alguns métodos de acordo com classificação da notação 'O'
- No entanto, para decisão de qual método utilizar é necessário testar na prática os métodos selecionados em conjuntos de dados diferentes quanto a:
  - Tamanho
  - Classificação inicial

## Métodos de Ordenação

- Ordenação por troca
  - *BubbleSort* (método da bolha)
  - *QuickSort* (método da troca e partição)
- Ordenação por inserção
  - *InsertionSort* (método da inserção direta)
  - *BinaryInsertionSort* (método da inserção direta binária)
- Ordenação por seleção
  - *SelectionSort* (método da seleção direta)
  - *HeapSort* (método da seleção em árvore)
- Outros métodos
  - *MergeSort* (método da intercalação)
  - *BucketSort* (método da distribuição de chave)

# Métodos de Ordenação Simples

- São três
  - *BubbleSort*
  - *InsertionSort*
  - *SelectionSort*
- Características
  - fácil implementação
  - alta complexidade
  - comparações ocorrem sempre entre posições adjacentes do vetor

## Método de Ordenação Bubble Sort (Método da Bolha)

## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.

0	1	2	3	4	5
77	42	35	12	101	5

## Borbulhando o maior elemento

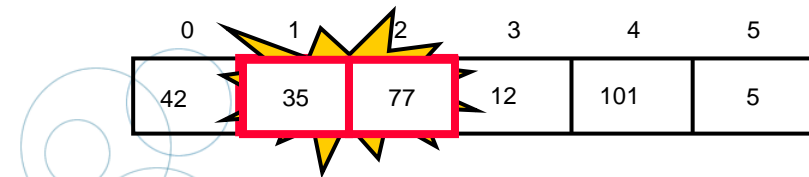
- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.

0	1	2	3	4	5
42	77	35	12	101	5



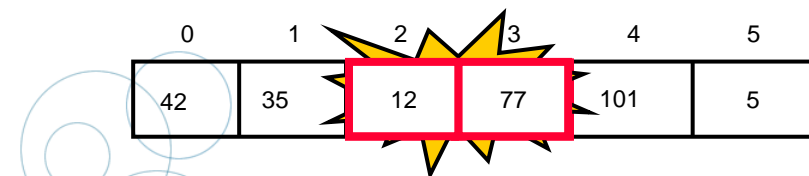
## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.



## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.



## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.

0	1	2	3	4	5
42	35	12	77	101	5

Não trocar

## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.

0	1	2	3	4	5
42	35	12	77	5	101

## Borbulhando o maior elemento

- Atravesse toda a coleção de elementos
  - Mova-se do início para o final
  - "Borbulhe" o maior valor para o final usando comparação de pares subsequentes e troca.

0	1	2	3	4	5
42	35	12	77	5	101

Maior Valor colocado no lugar

## Borbulhando o maior elemento

```
for (int j = 0; j < vetor.length-1; j++) {
    if (vetor[j] > vetor[j+1]) {
        int aux = vetor[j];
        vetor[j] = vetor[j+1];
        vetor[j+1] = aux;
    }
}
```

## Itens de Interesse

- Note que o maior valor está no último lugar
- Os outros valores estão fora de seu lugar
- Então precisamos repetir todo o processo

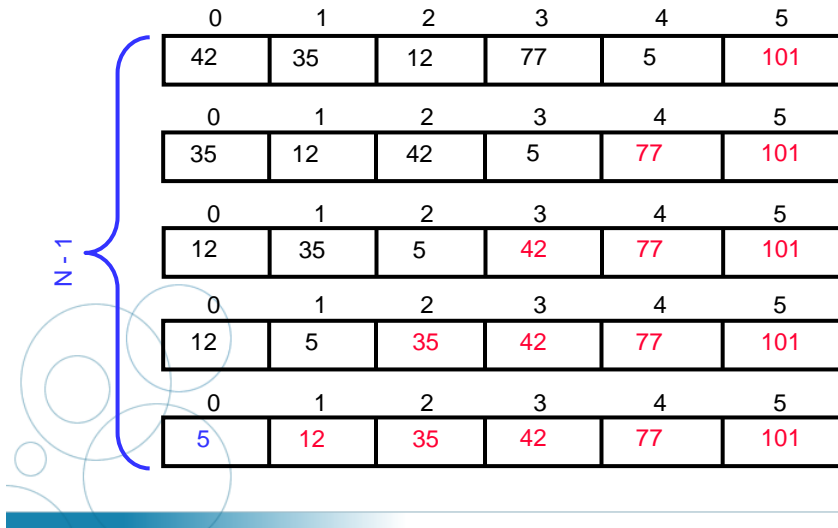
0	1	2	3	4	5
42	35	12	77	5	101

Maior Valor colocado no lugar

## Quantas vezes é necessário repetir o método?

- Se nós temos  $N$  elementos...
- E se cada vez que nós repetimos o processo colocamos o maior na última posição...
- Então repetiremos o método  $N - 1$  vezes.
- Assim garantiremos que os  $N$  elementos estarão em seu lugar.

## “Borbulhando” Todos os Elementos

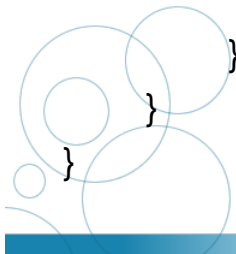


## Borbulhando todos os elementos

```

for (int i=0; i<vetor.length-1; i++) {
    for (int j = 0; j < vetor.length-1; j++) {
        if (vetor[j] > vetor[j+1]) {
            int aux = vetor[j];
            vetor[j] = vetor[j+1];
            vetor[j+1] = aux;
        }
    }
}

```



## Reduzindo o número de Comparações

0	1	2	3	4	5
77	42	35	12	101	5
0	1	2	3	4	5
42	35	12	77	5	101
0	1	2	3	4	5
35	12	42	5	77	101
0	1	2	3	4	5
12	35	5	42	77	101
0	1	2	3	4	5
12	5	35	42	77	101

## Reduzindo o número de Comparações

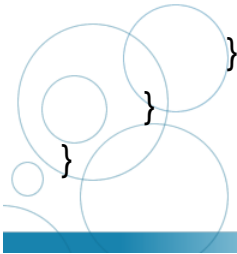
- Na N-ésima “bolha”, precisaremos fazer no máximo N comparações.
- Por exemplo:
  - Esta é a quarta “bolha”
  - Máximo é 6
  - Resta então 2 comparações a serem feitas

0	1	2	3	4	5
12	35	5	42	77	101

Diagram illustrating the final state of the array after the 4th pass (N=4). The elements 12, 35, and 5 are grouped by a bracket, indicating they are the elements being compared in the current pass. The elements 42, 77, and 101 are already in their final sorted positions.

## Reduzindo o número de comparações

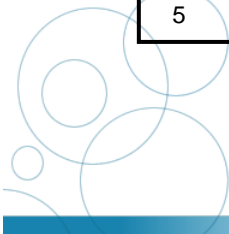
```
for (int i = 0; i < vetor.length-1; i++) {
    for (int j = 0; j < (vetor.length-1) -i; j++) {
        if (vetor[j] > vetor[j+1]) {
            int aux = vetor[j];
            vetor[j] = vetor[j+1];
            vetor[j+1] = aux;
        }
    }
}
```



## E se a coleção estiver arrumada?

- E se a coleção estiver parcialmente arrumada?
- Como detectar o estágio de todos os elementos no lugar e parar o processo?

0	1	2	3	4	5
5	12	35	42	77	101



## Usando uma sentinela

- Neste caso podemos colocar em uso uma variável booleana que detecta se houve ou não trocas.
- Se não houver trocas significa que a coleção estará arrumada!
- Esta sentinela deve ser desarmada a cada uso do processo.

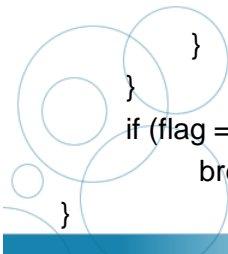


## Reduzindo o número de comparações

```

for (int i = 0; i < vetor.length-1; i++) {
    boolean flag = false;
    for (int j = 0; j < (vetor.length-1) -i; j++) {
        if (vetor[j] > vetor[j+1]) {
            int aux = vetor[j];
            vetor[j] = vetor[j+1];
            vetor[j+1] = aux;
            flag = true;
        }
    }
    if (flag == false)
        break;
}

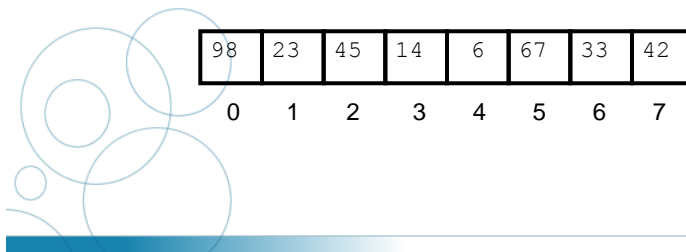
```





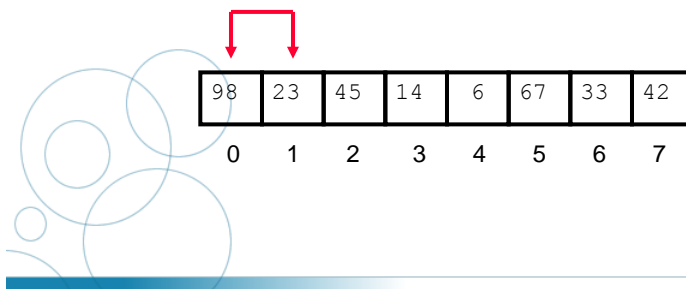
## Um Exemplo Animado

N	8	flag	true
bolhas	7		
índice			

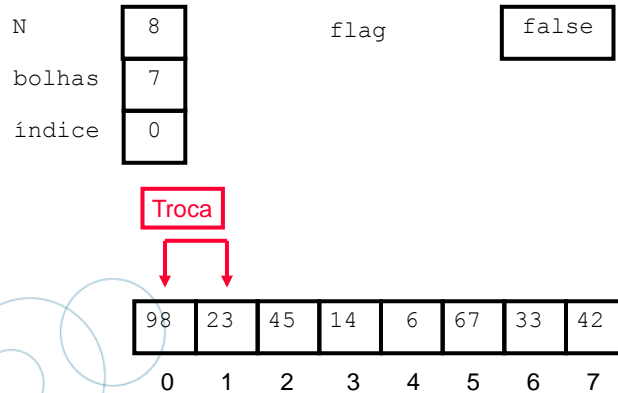


## Um Exemplo Animado

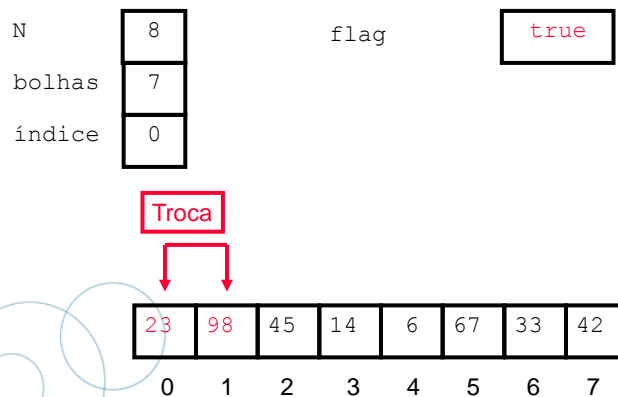
N	8	flag	false
bolhas	7		
índice	0		



## Um Exemplo Animado

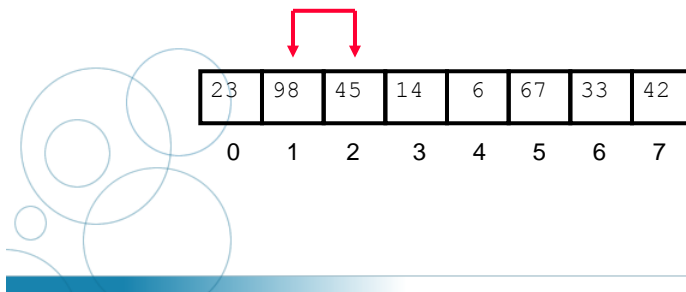


## Um Exemplo Animado



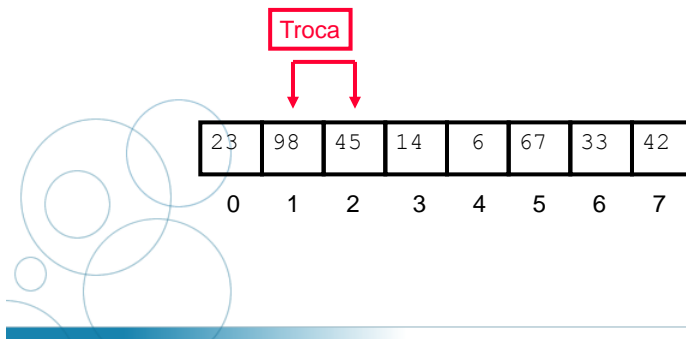
## Um Exemplo Animado

N	8	flag	true
bolhas	7		
índice	1		

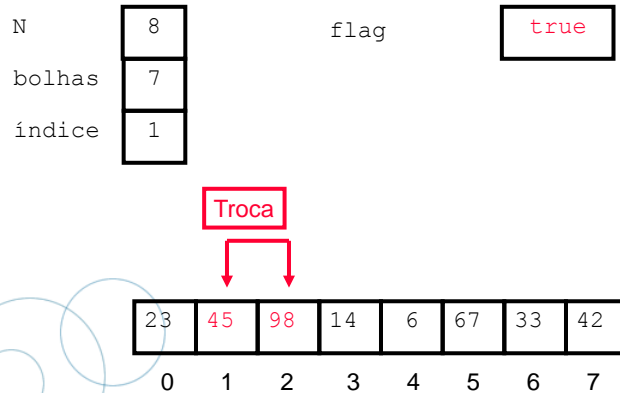


## Um Exemplo Animado

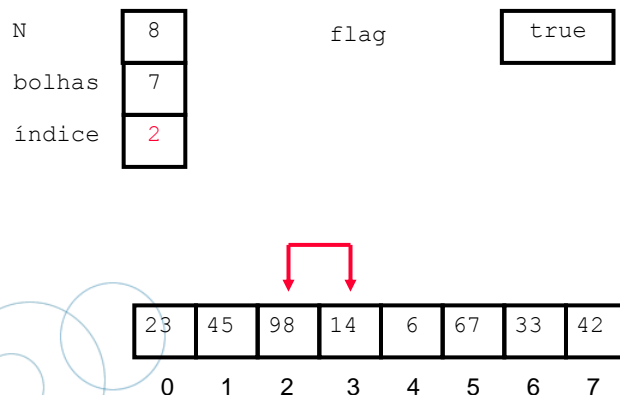
N	8	flag	true
bolhas	7		
índice	1		



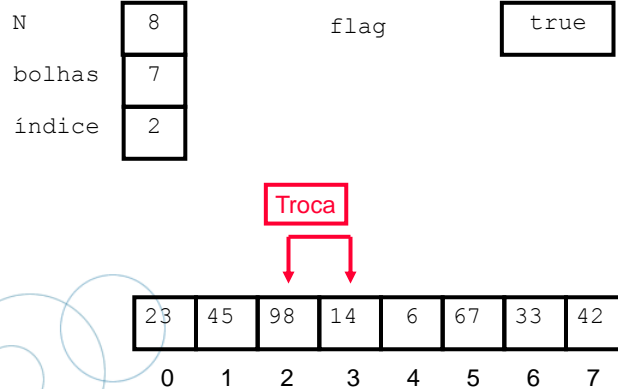
## Um Exemplo Animado



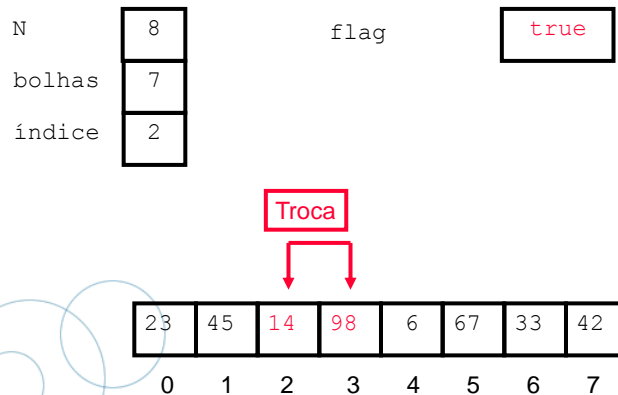
## Um Exemplo Animado



## Um Exemplo Animado

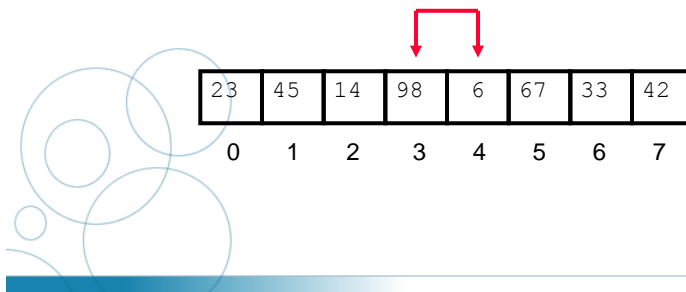


## Um Exemplo Animado



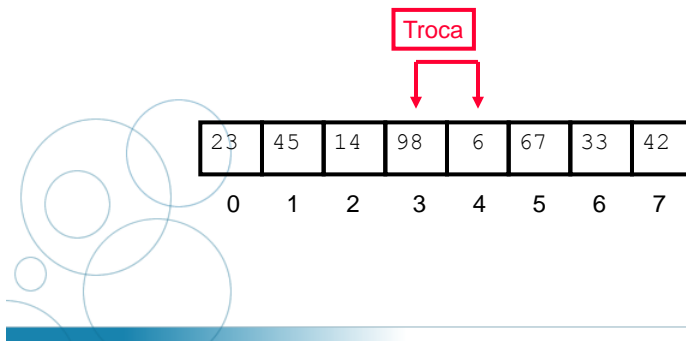
## Um Exemplo Animado

N	8	flag	true
bolhas	7		
índice	3		

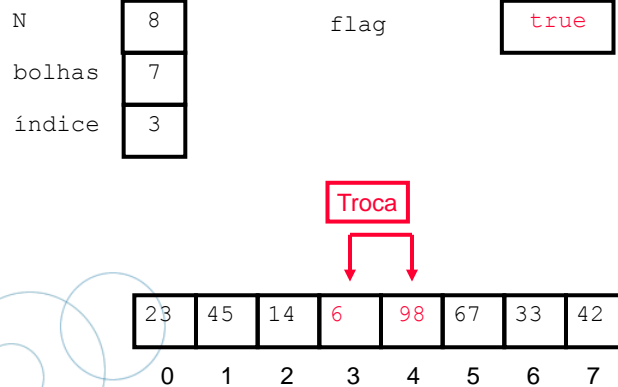


## Um Exemplo Animado

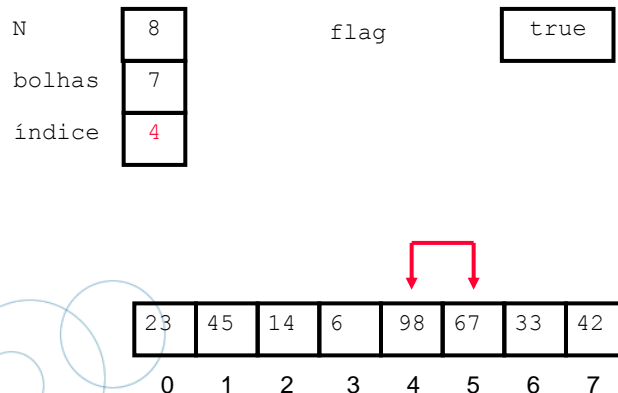
N	8	flag	true
bolhas	7		
índice	3		



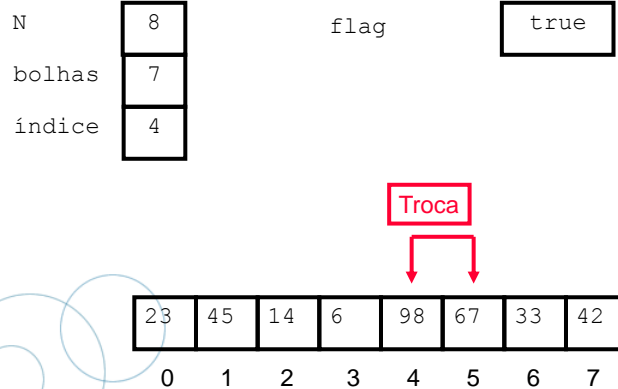
## Um Exemplo Animado



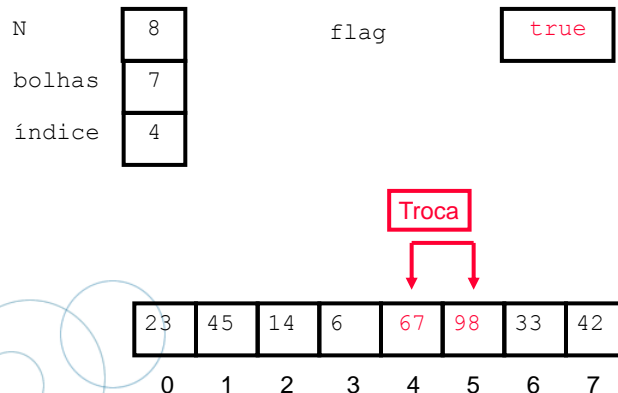
## Um Exemplo Animado



## Um Exemplo Animado



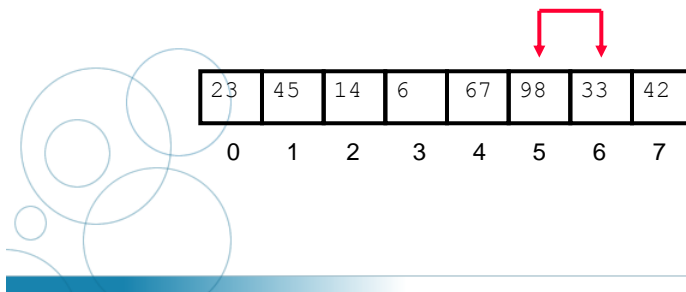
## Um Exemplo Animado





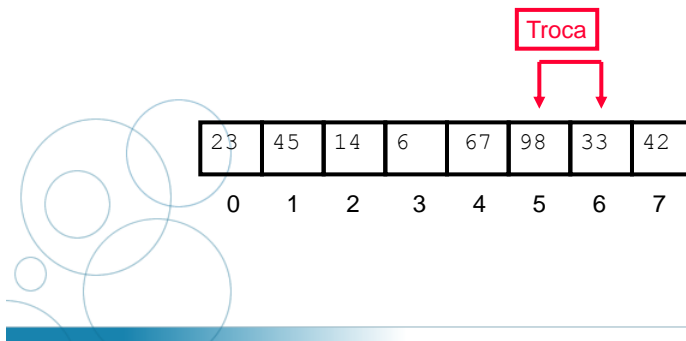
## Um Exemplo Animado

N	8	flag	true
bolhas	7		
índice	5		

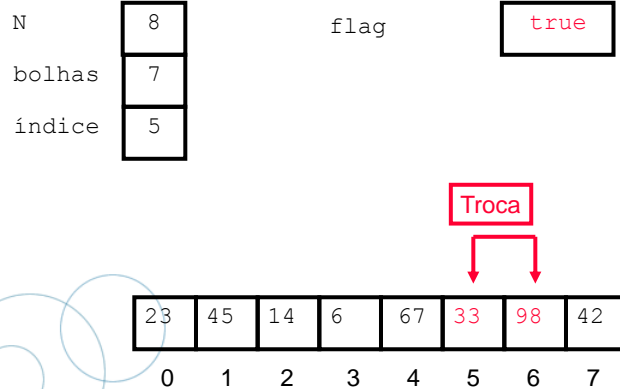


## Um Exemplo Animado

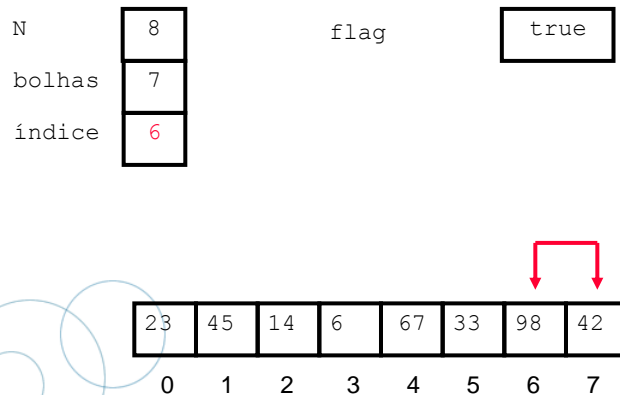
N	8	flag	true
bolhas	7		
índice	5		



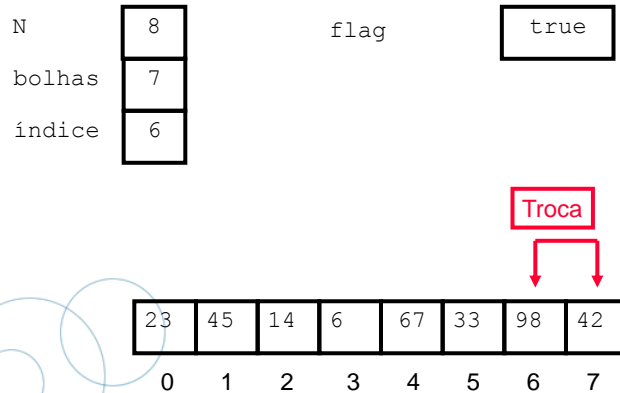
## Um Exemplo Animado



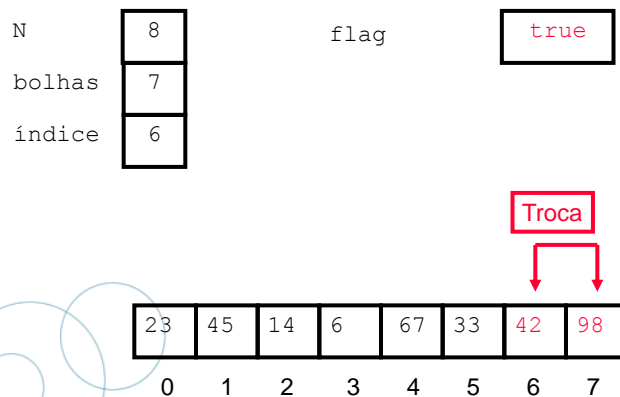
## Um Exemplo Animado



## Um Exemplo Animado



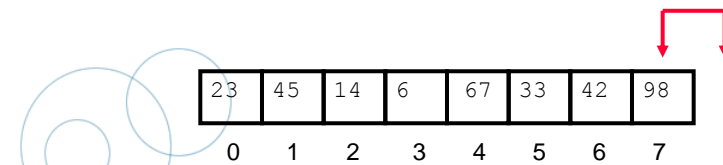
## Um Exemplo Animado



## Após a primeira bolha

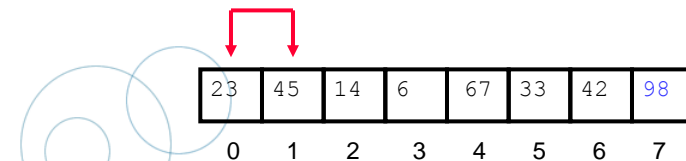
N	8	flag	true
bolhas	6		
índice	7		

Terminada a primeira "bolha"

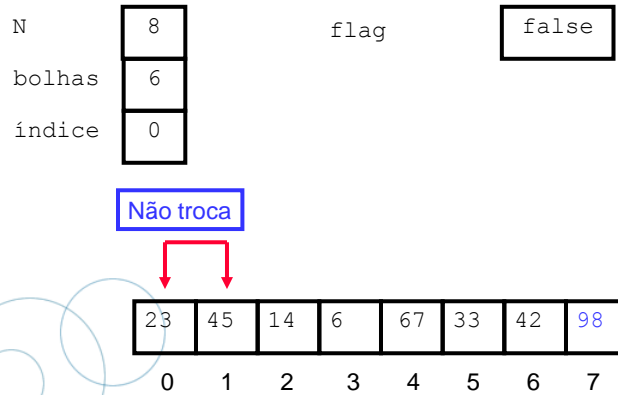


## A segunda bolha

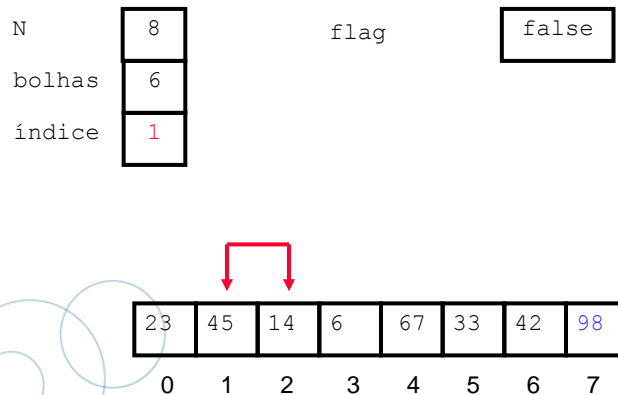
N	8	flag	false
bolhas	6		
índice	0		



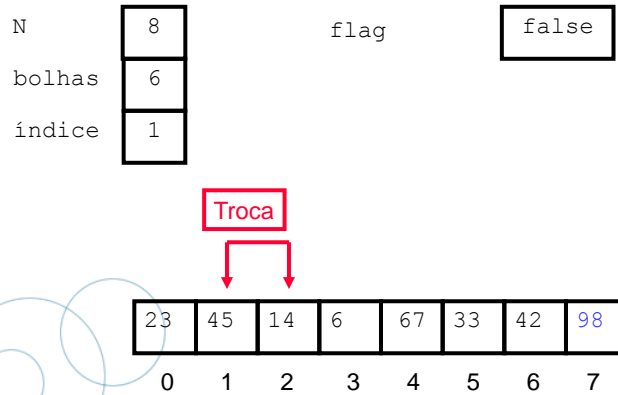
## A segunda bolha



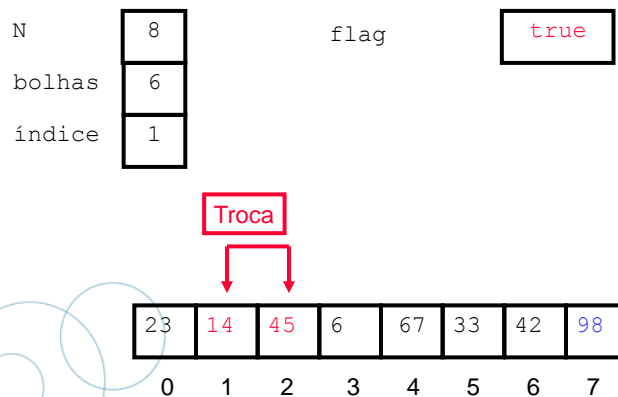
## A segunda bolha



## A segunda bolha

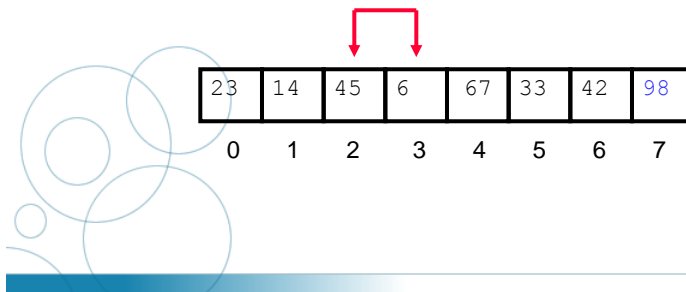


## A segunda bolha



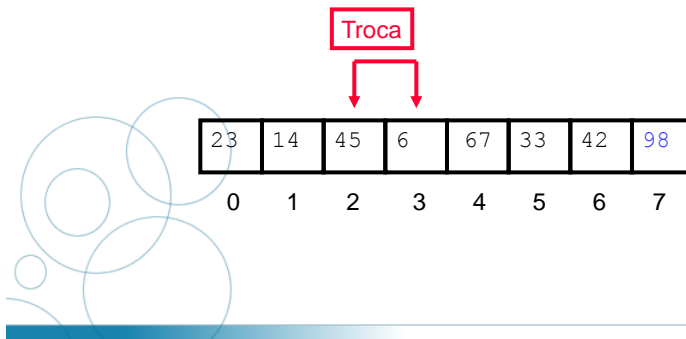
## A segunda bolha

N	8	flag	true
bolhas	6		
índice	2		

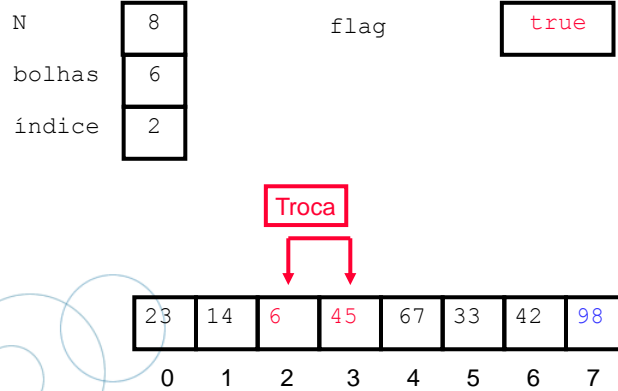


## A segunda bolha

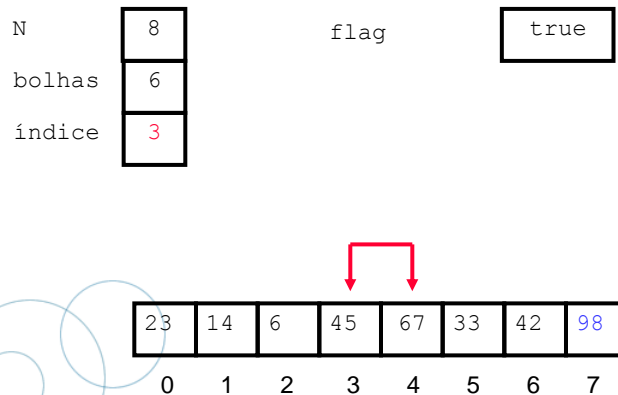
N	8	flag	true
bolhas	6		
índice	2		



## A segunda bolha



## A segunda bolha





## A segunda bolha

N	8	flag	true
bolhas	6		
índice	3		

Não troca

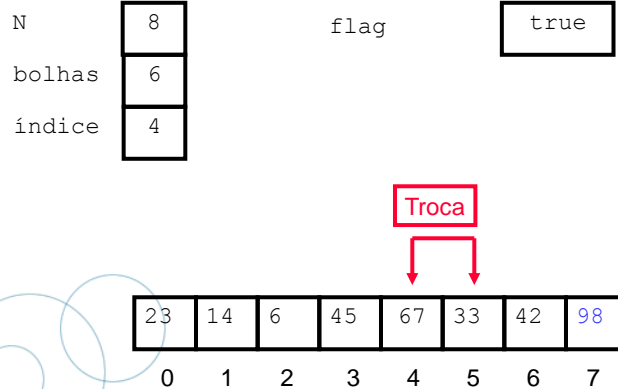
23	14	6	45	67	33	42	98
0	1	2	3	4	5	6	7

## A segunda bolha

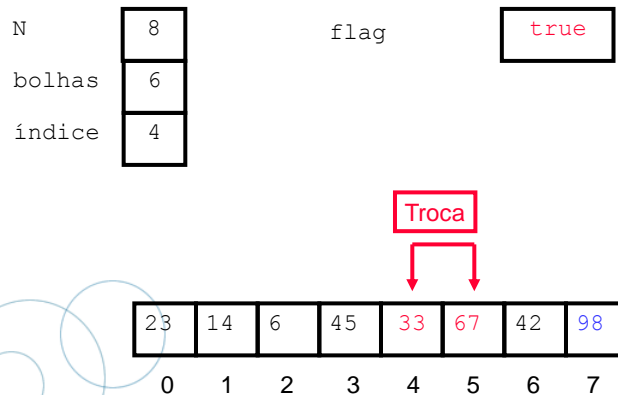
N	8	flag	true
bolhas	6		
índice	4		

23	14	6	45	67	33	42	98
0	1	2	3	4	5	6	7

## A segunda bolha

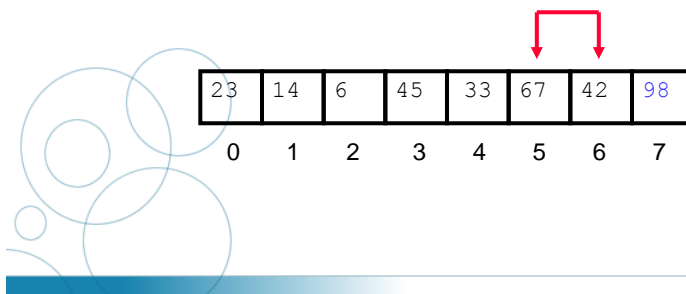


## A segunda bolha



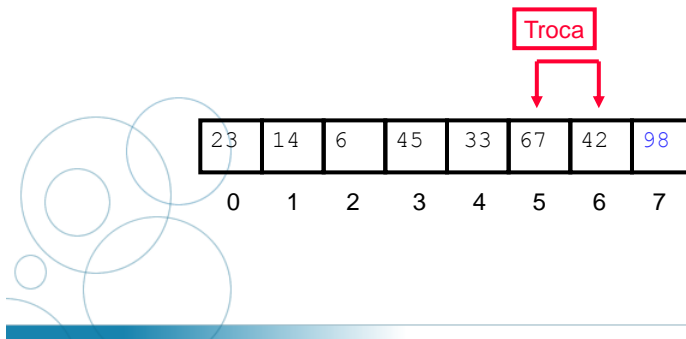
## A segunda bolha

N	8	flag	true
bolhas	6		
índice	5		

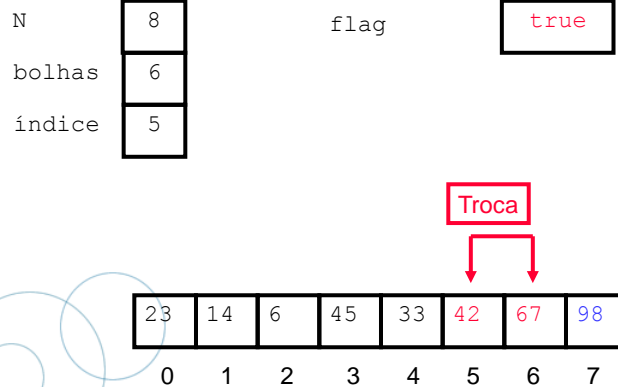


## A segunda bolha

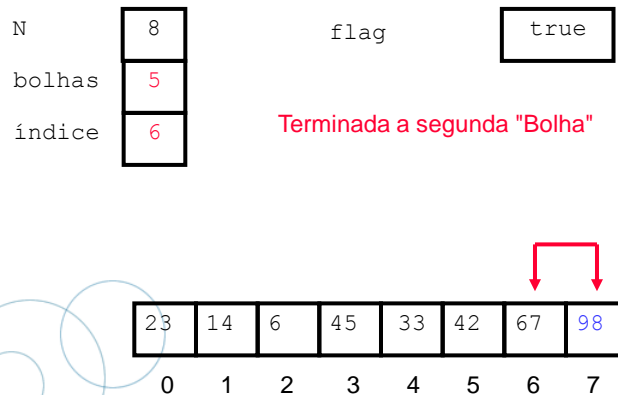
N	8	flag	true
bolhas	6		
índice	5		



## A segunda bolha

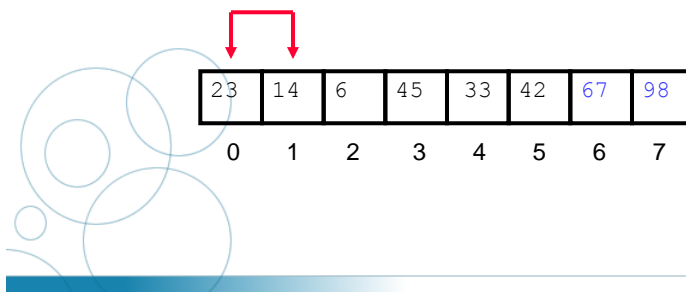


## Após a segunda bolha



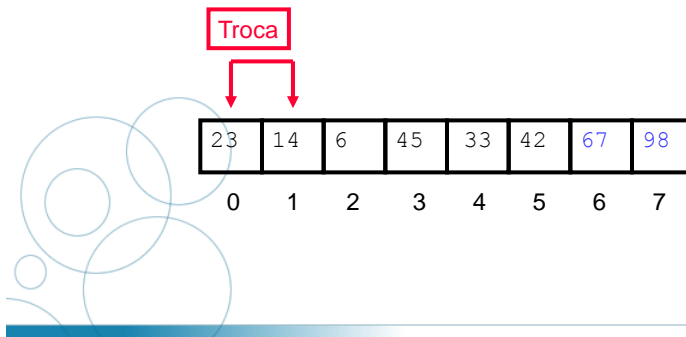
## A terceira bolha

N	8	flag	false
bolhas	5		
índice	0		

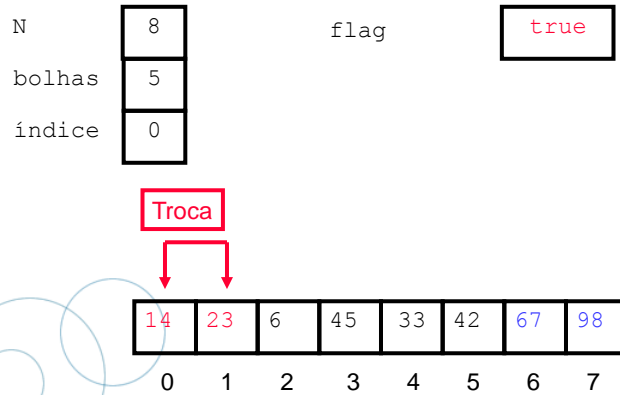


## A terceira bolha

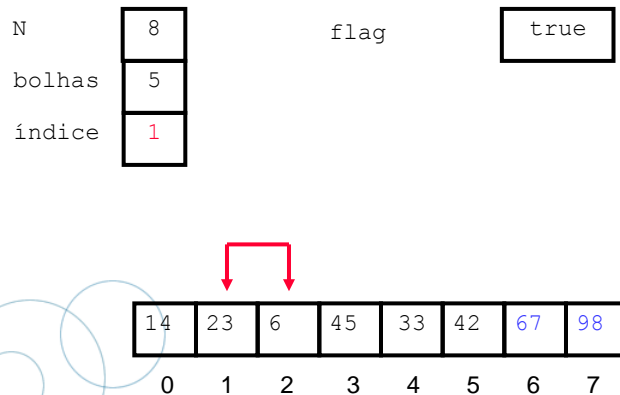
N	8	flag	false
bolhas	5		
índice	0		



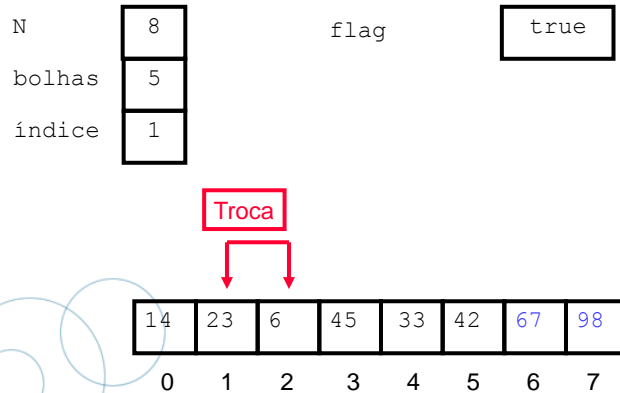
## A terceira bolha



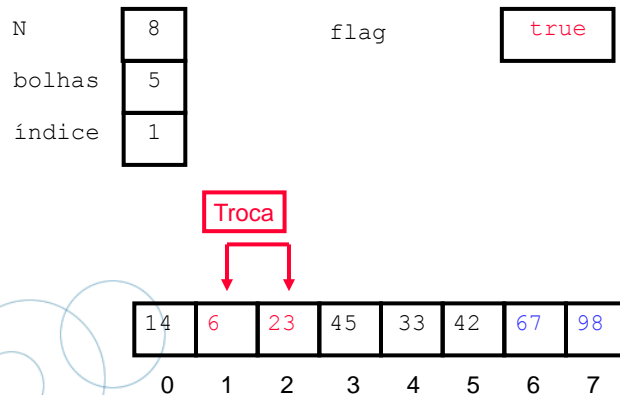
## A terceira bolha



## A terceira bolha

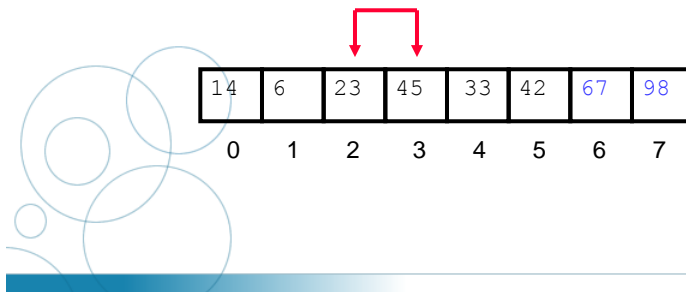


## A terceira bolha



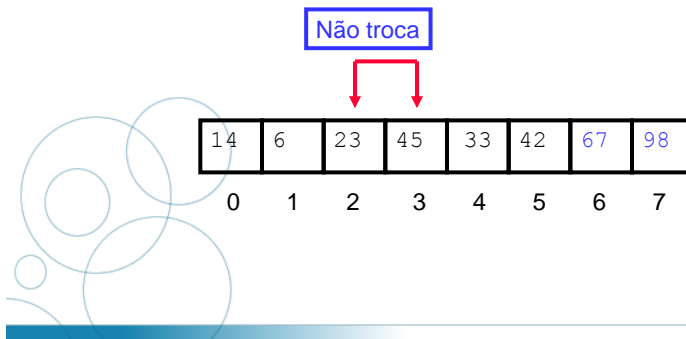
## A terceira bolha

N	8	flag	true
bolhas	5		
índice	2		



## A terceira bolha

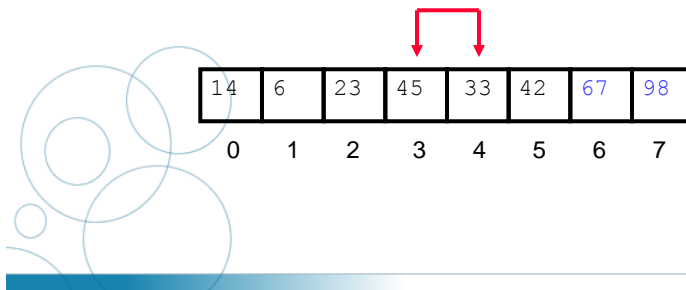
N	8	flag	true
bolhas	5		
índice	2		





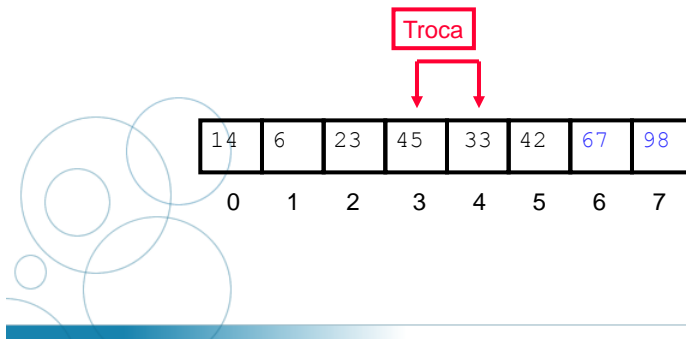
## A terceira bolha

N	8	flag	true
bolhas	5		
índice	3		



## A terceira bolha

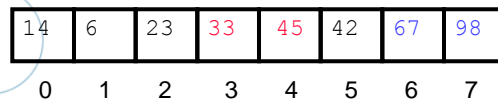
N	8	flag	true
bolhas	5		
índice	3		



## A terceira bolha

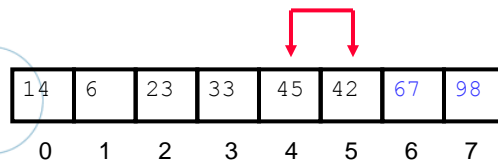
N	8	flag	true
bolhas	5		
índice	3		

Troca



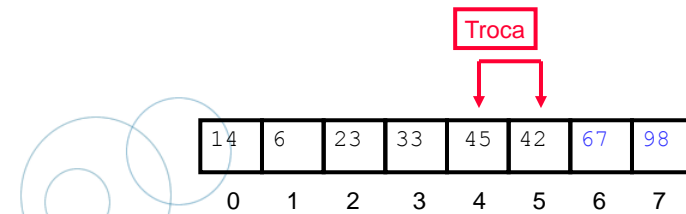
## A terceira bolha

N	8	flag	true
bolhas	5		
índice	4		



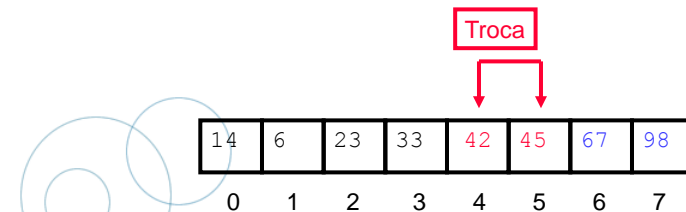
## A terceira bolha

N	8	flag	true
bolhas	5		
índice	4		



## A terceira bolha

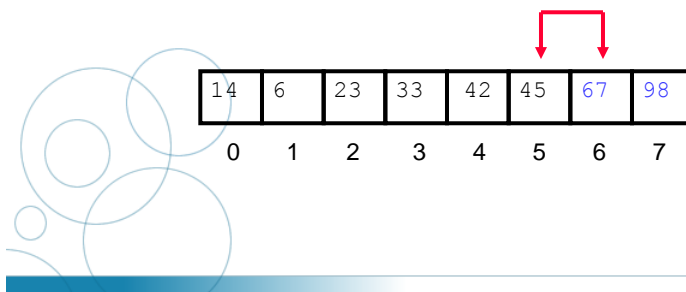
N	8	flag	true
bolhas	5		
índice	4		



## Após a terceira bolha

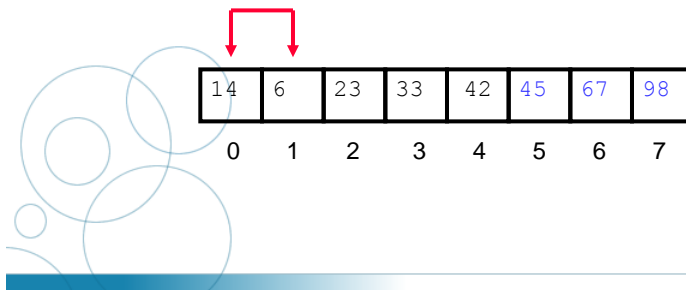
N	8	flag	true
bolhas	4		
índice	5		

Terminada a terceira "Bolha"

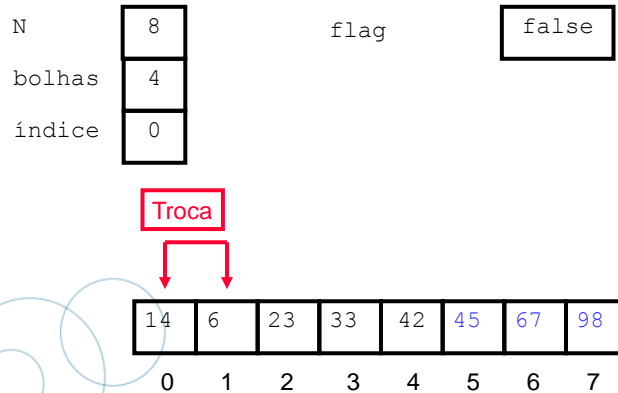


## A quarta bolha

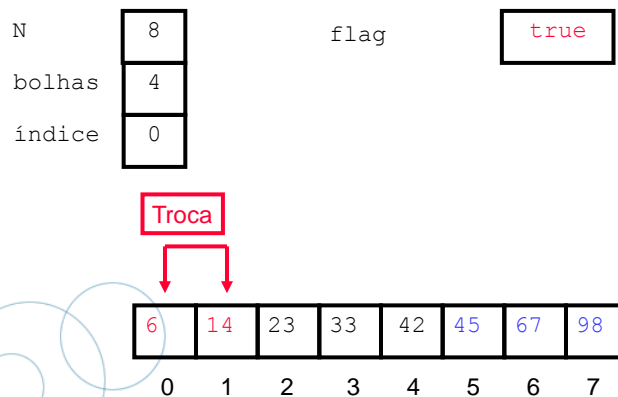
N	8	flag	false
bolhas	4		
índice	0		



## A quarta bolha

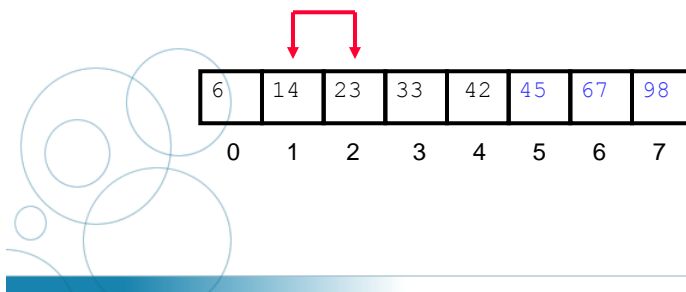


## A quarta bolha



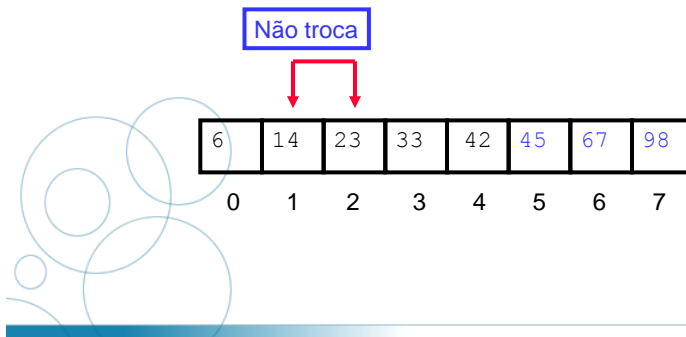
## A quarta bolha

N	8	flag	true
bolhas	4		
índice	1		



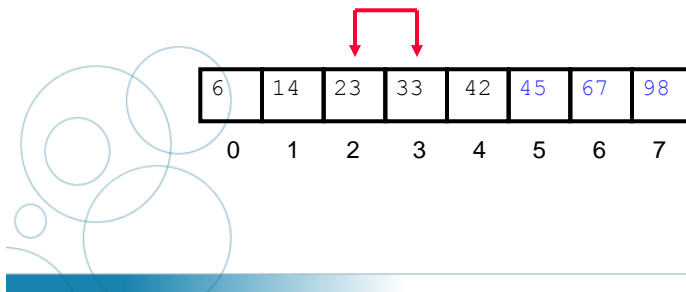
## A quarta bolha

N	8	flag	true
bolhas	4		
índice	1		



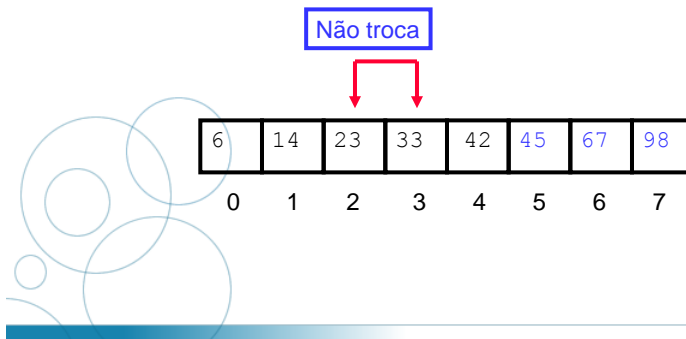
## A quarta bolha

N	8	flag	true
bolhas	4		
índice	2		



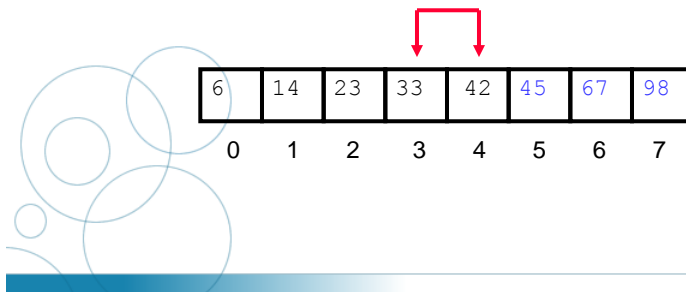
## A quarta bolha

N	8	flag	true
bolhas	4		
índice	2		



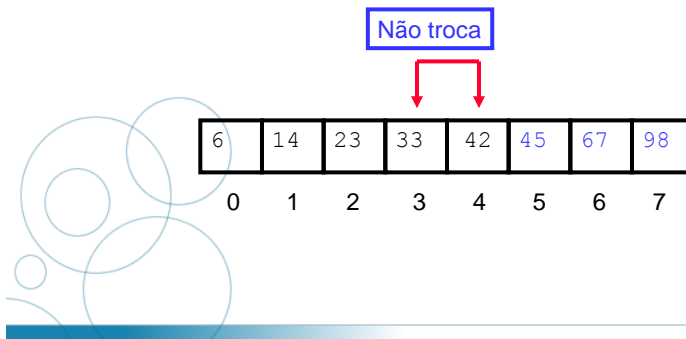
## A quarta bolha

N	8	flag	true
bolhas	4		
índice	3		



## A quarta bolha

N	8	flag	true
bolhas	4		
índice	3		

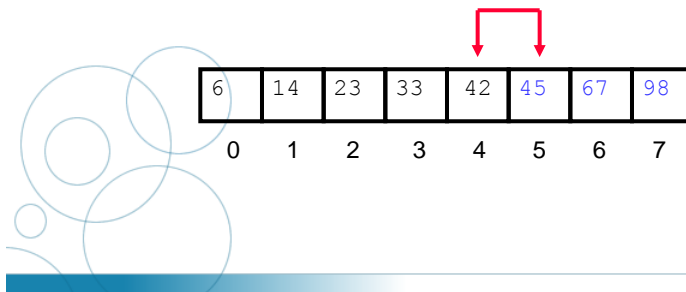




## Após a quarta bolha

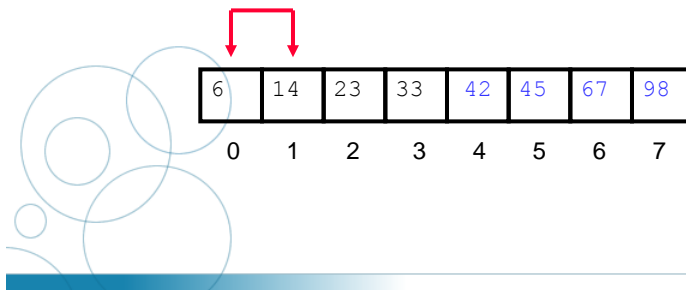
N	8	flag	true
bolhas	3		
índice	4		

Terminada a quarta "Bolha"

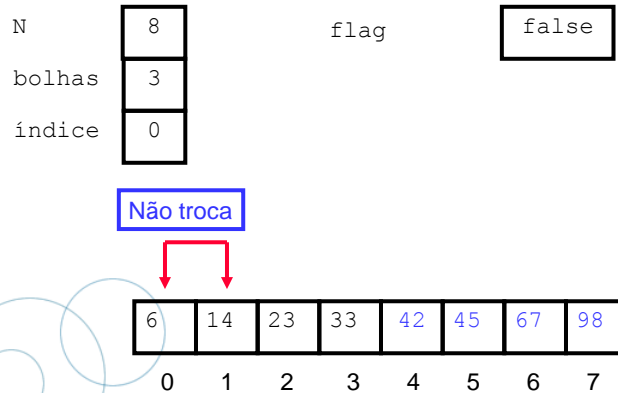


## A quinta bolha

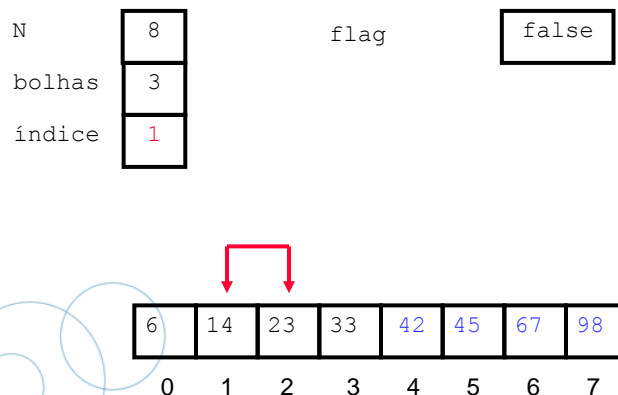
N	8	flag	false
bolhas	3		
índice	0		



## A quinta bolha



## A quinta bolha



## A quinta bolha

N	8	flag	false
bolhas	3		
índice	1		

Não troca

6	14	23	33	42	45	67	98
0	1	2	3	4	5	6	7

## A quinta bolha

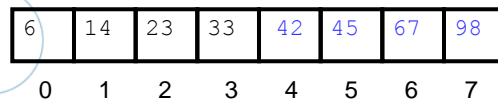
N	8	flag	false
bolhas	3		
índice	2		

6	14	23	33	42	45	67	98
0	1	2	3	4	5	6	7

## A quinta bolha

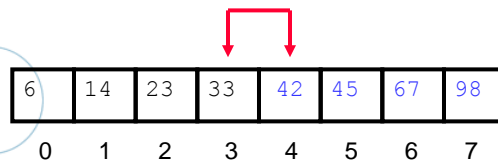
N	8	flag	false
bolhas	3		
índice	2		

Não troca



## Após a quinta bolha

N	8	flag	false
bolhas	2		
índice	3	Terminada a quinta "Bolha"	



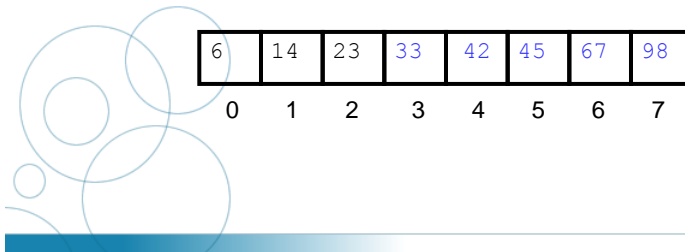
# Terminado

N	8
bolhas	2
índice	3

flag false

Nenhuma troca foi realizada

Podemos pular os dois próximos estágios.



# Resumo

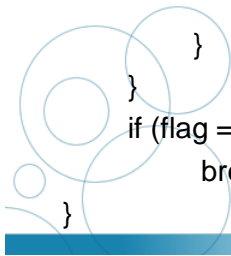
- O algoritmo do “Método da Bolha” irá mover o maior valor para sua correta posição (para a direita)
- Repetir “Bubble Sort” até que todos os elementos estejam em seus lugares:
  - Máximo número de estágios é N-1.
  - Pode ser antecipado o final se não ocorrer Trocas
  - O número de elementos a serem comparados deve ser decrementado a cada vez que o processo ocorre.

## Método BubbleSort

```

for (int i = 0; i < vetor.length-1; i++) {
    boolean flag = false;
    for (int j = 0; j < (vetor.length-1) -i; j++) {
        if (vetor[j] > vetor[j+1]) {
            int aux = vetor[j];
            vetor[j] = vetor[j+1];
            vetor[j+1] = aux;
            flag = true;
        }
    }
    if (flag == false)
        break;
}

```

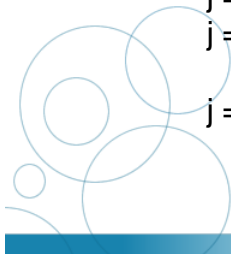


## Função de Complexidade

- Quanto tempo o algoritmo consome para fazer a ordenação?
- O tempo é proporcional ao número de execuções da comparação "`vetor[j] > vetor[j+1]`". Calculemos esse número.
- No pior caso, para cada valor de  $j$ , a variável  $i$  assume os valores  $j-1, \dots, 0$ .

$j = 1$	$\rightarrow$	$i = 0$	$\rightarrow$	1
$j = 2$	$\rightarrow$	$i = 1, 0$	$\rightarrow$	2
$j = 3$	$\rightarrow$	$i = 2, 1, 0$	$\rightarrow$	3
$\dots$				
$j = n-1$	$\rightarrow$	$i = n-2, \dots, 0$	$\rightarrow$	$n-1$

**Pior Caso**  $\rightarrow \frac{n*(n-1)}{2}$



# Função de Complexidade

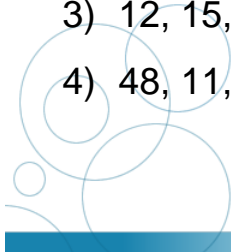
- $n*(n-1)/2 = O(n^2)$  ← Pior Caso



## Exercícios

Simule o BubbleSort, mostrando como é feita passo a passo a ordenação dos vetores a seguir:

- 1) 21, 45, 37, 51, 23, 98, 10, 33
- 2) 88, 15, 27, 55, 44, 38
- 3) 12, 15, 81, 75, 37, 47, 25, 34
- 4) 48, 11, 88, 33, 57, 12, 18, 87, 54, 8



## Referências Bibliográficas

- TENEMBAUM, Aaron M. *Estrutura de dados usando C*. São Paulo: Makron Books, 1995.
- MORAES, Carlos Roberto. *Estrutura de Dados e Algoritmos: uma abordagem didática*. 2. ed. São Paulo: Futura, 2003.

