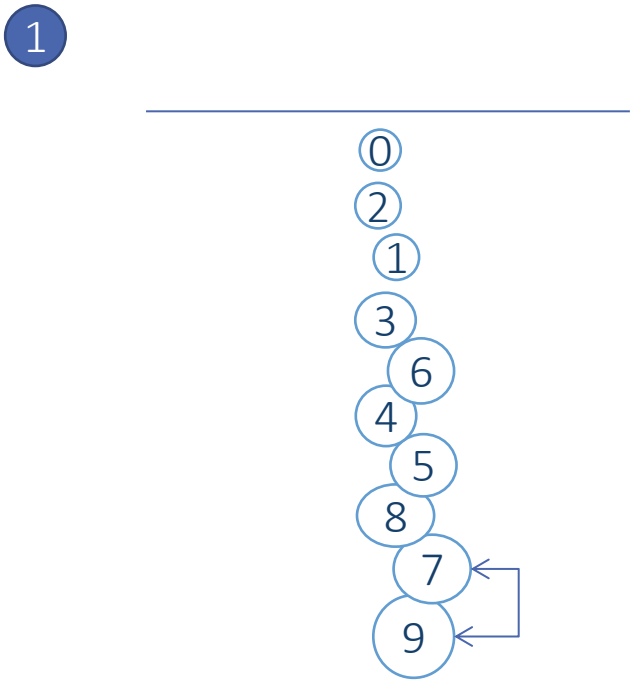
Aula 10
Ordenação
Bubble Sort

Algoritmos e Estruturas de Dados

Bubble Sort

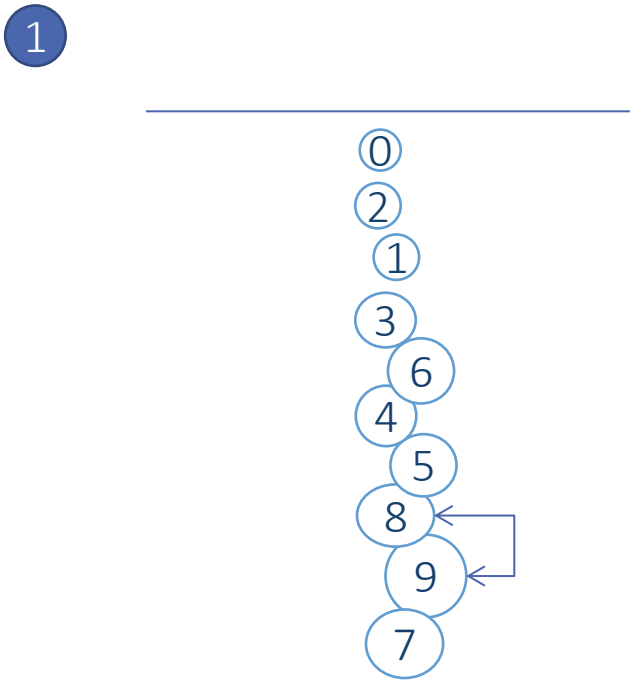
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



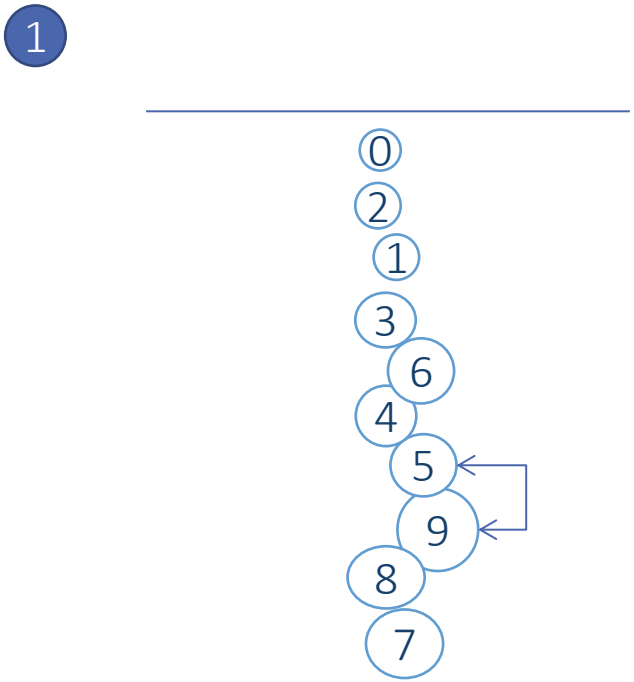
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



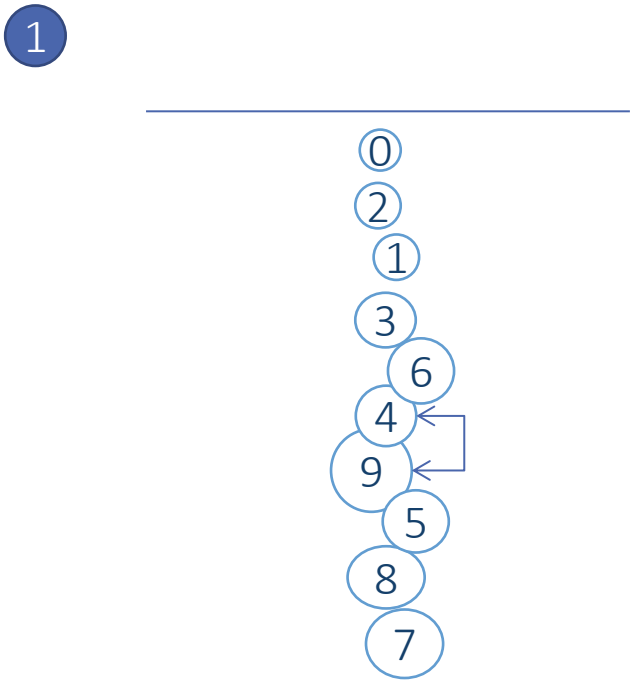
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



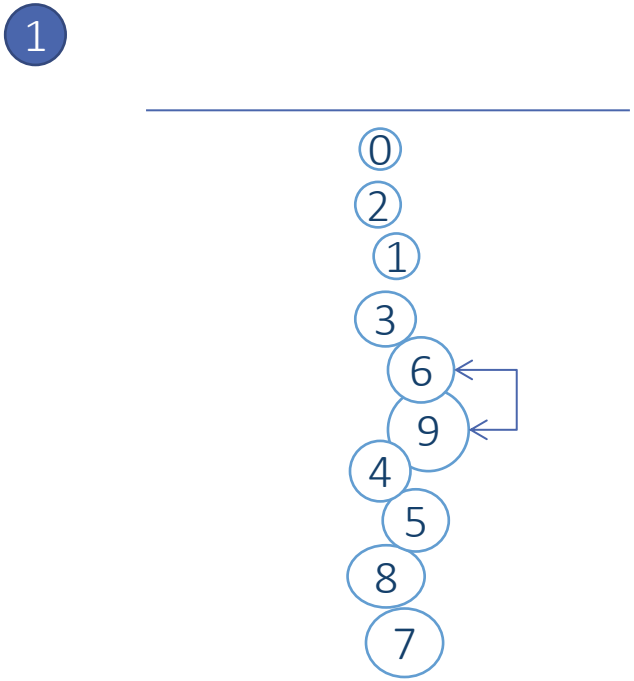
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



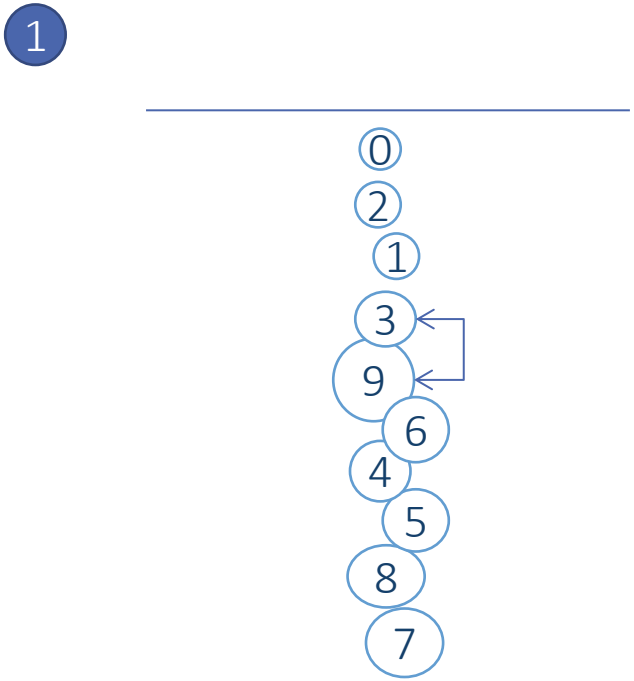
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



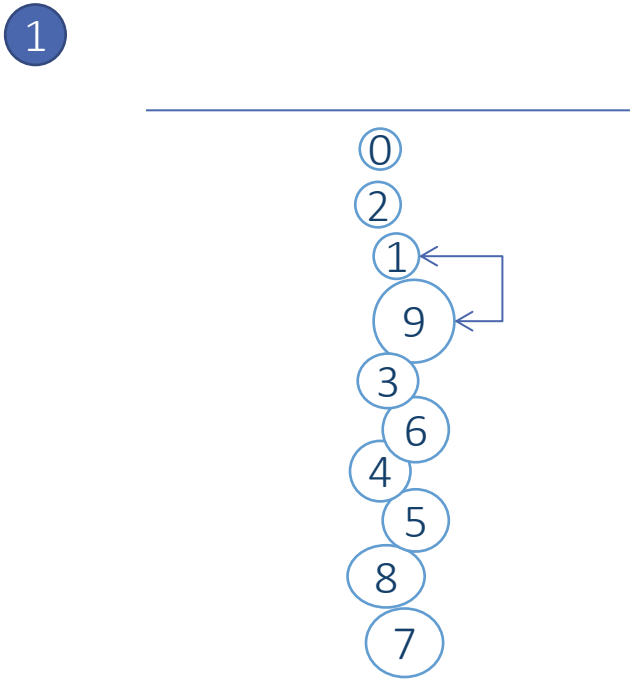
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



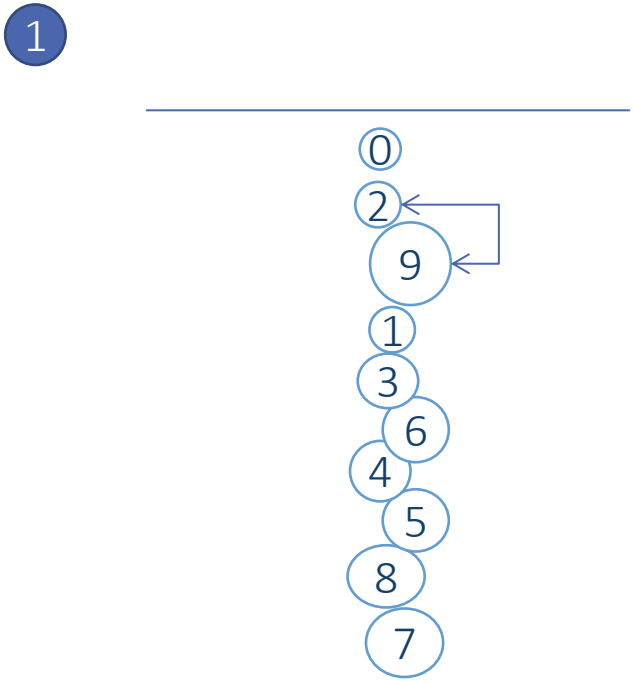
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



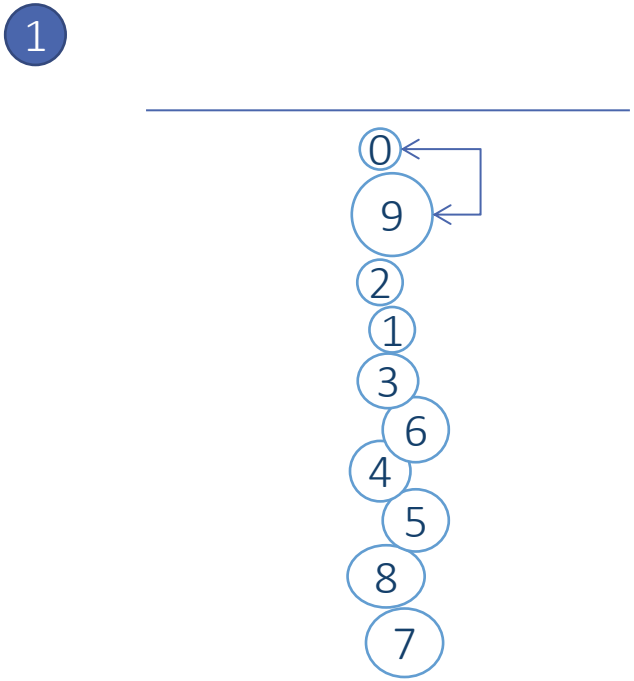
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



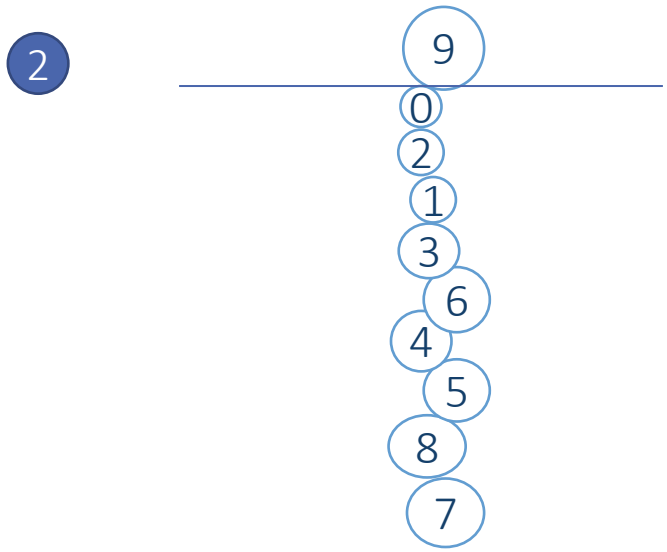
Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- Em cada iteração
  - Fazer flutuar para o topo o maior elemento
  - Comparando apenas pares de elementos



Bubble Sort

- Inspirado na forma como as bolhas de água num tanque procuram o seu nível
- Percorrer o array várias vezes
- No fim da iteração
  - O maior elemento está no topo
  - Diminuir o tamanho
  - Repetir do início

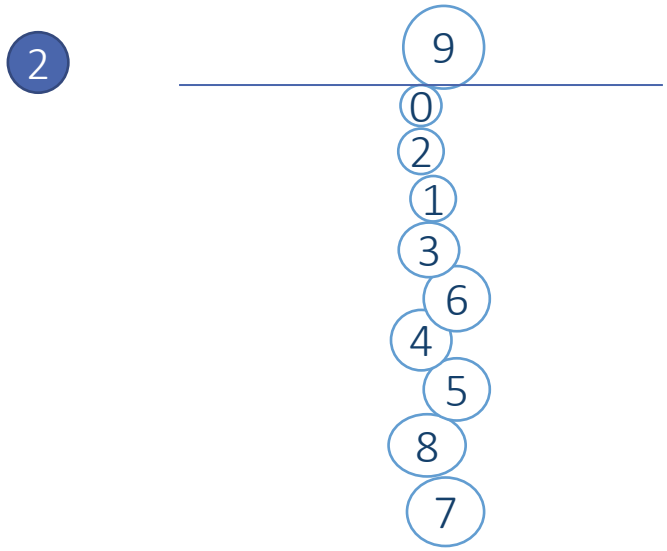


Early exit:
Se durante uma iteração não existir nenhuma troca

Podemos acabar, está tudo ordenado

Bubble Sort

```
public static void sort(Comparable[] a)
{
    int n = a.length;
    boolean swapped = false;
    for(int i = n; i > 0; i--)
    {
        swapped = false;
        for(int j = 1; j < i ; j++)
        {
            if(less(a[j],a[j-1]))
            {
                exchange(a, j,j-1);
                swapped = true;
            }
        }
        if(!swapped) return;
    }
}
```



Complexidade Temporal

- *No pior caso (temos de efetuar todas as iterações)*
  - *2.º for* $\overbrace{(n-1)(n-2) + \ldots + 1 + 0}^{n}$

$$= \frac{n}{2}(n - 1 + 0)$$

$$= \frac{n^2 - n}{2}$$

$$\sim \frac{n^2}{2}$$

```
public static void sort(Comparable[] a)
{
    int n = a.length;
    boolean swapped = false;
    for(int i = n; i > 0; i--)
    {
        swapped = false;
        for(int j = 1; j < i ; j++)
        {
            if(less(a[j],a[j-1]))
            {
                exchange(a, j,j-1);
                swapped = true;
            }
        }
        if(!swapped) return;
    }
}
```

Complexidade Temporal

- *No melhor caso*
  - *Já está ordenado*
  - *Sai depois da primeira iteração*
  - *2.º for*

$$(n-1)+0+\ldots+0+0$$

$$= n - 1$$

$$\sim n$$

```
public static void sort(Comparable[] a)
{
    int n = a.length;
    boolean swapped = false;
    for(int i = n; i > 0; i--)
    {
        swapped = false;
        for(int j = 1; j < i ; j++)
        {
            if(less(a[j],a[j-1]))
            {
                exchange(a, j,j-1);
                swapped = true;
            }
        }
        if(!swapped) return;
    }
}
```

Complexidade Temporal

- *Array aleatório*
  - *less falha 50% das vezes*
  - *Mas a probabilidade de não haver troca numa iteração é mt baixa*
  - *2.º for*

$$\overbrace{(n-1)(n-2) + \ldots + 1 + 0}^{n} = \frac{n}{2}(n-1+0) = \frac{n^2-n}{2} \sim \frac{n^2}{2}$$

```
public static void sort(Comparable[] a)
{
    int n = a.length;
    boolean swapped = false;
    for(int i = n; i > 0; i--)
    {
        swapped = false;
        for(int j = 1; j < i ; j++)
        {
            if(less(a[j],a[j-1]))
            {
                exchange(a, j,j-1);
                swapped = true;
            }
        }
        if(!swapped) return;
    }
}
```

Complexidade Temporal

```
public static void sort(Comparable[] a)
{
    int n = a.length;
    boolean swapped = false;
    for(int i = n; i > 0; i--)
    {
        swapped = false;
        for(int j = 1; j < i ; j++)
        {
            if(less(a[j],a[j-1]))
            {
                exchange(a, j,j-1);
                swapped = true;
            }
        }
        if(!swapped) return;
    }
}
```

Bubble Sort	Best case	Worst Case	Aleatório	O
less/compare	~n	$\sim n^2/2$	$\sim n^2/2$	$O(n^2)$
exchange	*0*	$\sim n^2/2$	$\sim n^2/4$	

Observação:
Conseguimos poupar nos exchanges, mas não conseguimos poupar nos compares