Troca Vizinhos

⇒**Implementação.** Escreva um método que receba um array de inteiros e troca os elementos vizinhos. Se o tamanho da sequência for ímpar, troque os vizinhos e mantenha o último elemento em sua posição.

```
public void troca_vizinhos(int[] v);
```

Exemplo

```
1 13 3 4 5
13 1 4 3 5
```

Restrições
O algoritmo deve ser in-place. Ou seja, não é permitido utilizar qualquer outro array auxiliar para trocar os elementos.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Palíndromo

⇒**Implementação.** Uma palavra ou frase é um palíndromo caso possa ser lida tanto da esquerda para a direita, como da direita para a esquerda, sem alterar seu sentido. Escreva um método que receba um array de char, representando uma palavra, e retorna um *boolean*. O método deve retornar *true*, caso a palavra seja um palíndromo, e *false*, caso o contrário.

public boolean eh_palindromo(char[] palavra);

Exemplo

```
['a','n','a']
true

['c','a','s','a']
false
```

Restrições
O algoritmo deve ser in-place. Ou seja, não é permitido utilizar qualquer outro array auxiliar para trocar os elementos.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Soma Dois

@author maria.dantas@ccc.ufcg.edu.br

⇒**Implementação.** Escreva um método que receba um array de inteiros e um inteiro, *target*. O método deve retornar um array de duas posições contendo um par de inteiros distintos cuja soma seja igual ao valor *target*, caso não exista um par, retorna array vazio.

public int[] two_sum(int[] v, int target);

Exemplo

Input

```
1 13 0 4 5
```

```
5
```

Output

```
[1, 4]
```

Restrições

1.1 Não é permitido usar nenhuma outra estrutura de dados auxiliar.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

É Primo?

⇒**Implementação.** Escreva um método que verifica se um número é primo ou não.

public boolean eh_primo(int n);

Exemplo

```
3
true

15
false
```

Restrições
Não é permitido usar nenhuma biblioteca de Java.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Tem repetido

⇒**Implementação.** Escreva um método que receba um array de inteiros não ordenado e verifica existe algum elemento repetido presente no array. Retorne true ou false.

public boolean tem_repetido(int[] v);

Exemplo

```
1 13 3 4 5 1
true
```

```
1 13 3 4 5 2
false
```

Restrições
O algoritmo deve ser in-place. Ou seja, não é permitido utilizar qualquer outro array auxiliar.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Vetor Circular

⇒**Implementação.** Escreva um método que receba um array inteiros e um inteiro N e retorne uma String que representa a impressão do array de forma circular. Ou seja, quando o último elemento for visitado ele deve voltar para o primeiro e continuar até chegar a N elementos impressos.

```
public String vetorCircular(int[] array, int quantidadeElementos);
```

Exemplo

```
4 5 6 1
6

4 5 6 1 4 5

3 4 2
8
3 4 2 3 4 2 3 4
```

Restrições
O algoritmo deve ser in-place. Ou seja, não é permitido utilizar qualquer outro array auxiliar para trocar os elementos.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Verifica Divisíveis

⇒**Implementação.** Escreva um método que receba um array de inteiros e verifica se há pelo menos um par x,y de elementos na sequência tal que y seja divisível por x.

public boolean verifica_divisiveis(int[] v);

Exemplos

```
// como 8 é divisível por 4 a saída é true

5 13 3 4 8
true

// como não há par x,y tal que y seja divisível por x a saída é false

5 13 3 4 7
false
```

Restrições
Não é permitido utilizar array auxiliar, nem bibliotecas de Java.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.

Bubble sort

⇒**Implementação.** Implemente o Bubble Sort. Bubble Sort é um algoritmo que itera em um array comparando elementos adjacentes e os troca (aos pares) se a ordem estiver errada. A varredura sob a lista é repetida até que todos os elementos estejam ordenados corretamente. O método deverá retornar o array ordenado de forma crescente.

```
public int[] bubble_sort(int[] lista);
```

Exemplos

```
5 13 3 4 8
3 4 5 8 13

1 2 3 4 5
1 2 3 4 5
```

Restrições
O algoritmo deve ser in-place. Ou seja, não é permitido utilizar qualquer outro array auxiliar.

⇒**Tempo de execução.** Use o método analítico para determinar a função que define o tempo de execução do algoritmo.

⇒**Ordem de crescimento.** Use as diretrizes que aprendemos em sala de aula para simplificar a função e determinar a que classe de funções pertence o algoritmo.