Exercícios Práticos - Bubble Sort

Exercício 1: Implementação Básica

Objetivo: Implementar o algoritmo Bubble Sort do zero.

Instruções:
1. Crie um programa em C++ que implemente o Bubble Sort
2. Teste com o array: {7, 3, 9, 1, 5, 2, 8}
3. Imprima o array antes e depois da ordenação

Código Base:

```cpp
#include <iostream>
using namespace std;

// TODO: Implementar a função bubbleSort
void bubbleSort(int arr[], int n) {
    // Seu código aqui
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {7, 3, 9, 1, 5, 2, 8};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Array original: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "Array ordenado: ";
    printArray(arr, n);

    return 0;
}
```

Resultado Esperado:

```
Array original: 7 3 9 1 5 2 8
Array ordenado: 1 2 3 5 7 8 9
```

Exercício 2: Versão Otimizada

Objetivo: Implementar a versão otimizada do Bubble Sort que para quando não há mais trocas.

Instruções:
1. Modifique o algoritmo para usar uma flag `swapped`
2. Teste com um array já ordenado: {1, 2, 3, 4, 5}
3. Conte quantas passadas foram necessárias

Dicas:
- Use uma variável booleana para detectar se houve trocas
- Se não houve trocas em uma passada, o array está ordenado

Teste Adicional:
Compare o número de passadas necessárias para:
- Array ordenado: {1, 2, 3, 4, 5}
- Array desordenado: {5, 4, 3, 2, 1}

Exercício 3: Contador de Operações

Objetivo: Analisar experimentalmente a complexidade do algoritmo.

Instruções:
1. Modifique o Bubble Sort para contar:
   - Número de comparações
   - Número de trocas
2. Teste com arrays de diferentes tamanhos
3. Compare com a complexidade teórica

Arrays para Teste:

```
 // Teste 1: Array pequeno
int arr1[] = {3, 1, 4, 1, 5};

// Teste 2: Array médio
int arr2[] = {9, 7, 5, 11, 12, 2, 14, 3, 10, 6};
```

```
// Teste 3: Pior caso (ordem inversa)
int arr3[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

// Teste 4: Melhor caso (já ordenado)
int arr4[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Análise Esperada:
Para um array de tamanho n:
- Comparações (pior caso): n(n-1)/2
- Comparações (melhor caso): n-1 (versão otimizada)

Exercício 4: Ordenação Decrescente

Objetivo: Adaptar o algoritmo para ordenação decrescente.

Instruções:
1. Modifique o Bubble Sort para ordenar em ordem decrescente
2. Teste com o array: {1, 5, 3, 9, 2, 7, 4}

Resultado Esperado:

```
Array original: 1 5 3 9 2 7 4
Array decrescente: 9 7 5 4 3 2 1
```

Pergunta: Qual é a única linha que precisa ser modificada?

Exercício 5: Visualização do Processo

Objetivo: Criar uma versão que mostra cada passo da ordenação.

Instruções:
1. Implemente uma versão que imprime:
   - O estado do array após cada troca
   - Quais elementos estão sendo comparados
   - O resultado de cada passada

Exemplo de Saída Esperada:

```
Passada 1:
 Comparando 5 e 2: Troca! -> [2, 5, 8, 1, 9]
 Comparando 5 e 8: Não troca
```

```
  Comparando 8 e 1: Troca! -> [2, 5, 1, 8, 9]
  Comparando 8 e 9: Não troca
  Final da passada 1: [2, 5, 1, 8, 9]

Passada 2:
  ...
```

Exercício 6: Análise Experimental

Objetivo: Medir o tempo de execução para diferentes tamanhos de array.

Instruções:

1. Use a biblioteca `<chrono>` para medir tempo
2. Teste com arrays de tamanhos: 100, 500, 1000, 2000
3. Gere arrays aleatórios para cada teste
4. Calcule o tempo médio de 5 execuções

Código Base para Medição:

```cpp
 #include <chrono>
#include <random>

// Função para gerar array aleatório
void generateRandomArray(int arr[], int n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, 1000);

    for (int i = 0; i < n; i++) {
        arr[i] = dis(gen);
    }
}

// Função para medir tempo
double measureTime(int arr[], int n) {
    auto start = chrono::high_resolution_clock::now();
    bubbleSort(arr, n);
    auto end = chrono::high_resolution_clock::now();

    auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
    return duration.count() / 1000.0; // retorna em milissegundos
}
```
