Zadanie 1: Zrozumienie idei Merge Sort

Cel: wiedzieć *co* implementujesz, zanim zaczniesz pisać kod.

- Merge Sort to algorytm **dziel i zwyciężaj**
- Działa w 3 krokach:
    1. Podziel tablicę na dwie połowy
    2. Posortuj każdą połowę (rekurencyjnie)
    3. Scal (merge) dwie posortowane połowy

Warunek stopu: tablica o długości `0` lub `1` jest już posortowana

Zadanie 2: Przygotowanie struktury programu

Cel: mieć miejsce, gdzie wszystko się połączy.

- Utwórz:
    - klasę (np. `MergeSort`)
    - metodę `Main`, żeby móc przetestować algorytm
- Przygotuj przykładową tablicę `int[]` do sortowania

Zadanie 3: Metoda MergeSort (rekurencja)

Cel: napisać metodę, która *dzieli tablicę* i wywołuje się sama.

- Sygnatura metody:
    - przyjmuje tablicę `int[]`
    - zwraca posortowaną tablicę `int[]`

- Sprawdź warunek stopu:
    - jeśli długość tablicy ≤ 1 → zwróć ją
- Podziel tablicę:
    - wylicz środek
    - utwórz lewą i prawą podtablicę
- Wywołaj `MergeSort` osobno dla lewej i prawej części

Na tym etapie **nie scalasz jeszcze tablic**

Zadanie 4: Metoda Merge (scalanie)

Cel: połączyć **dwie posortowane tablice** w jedną posortowaną.

- Metoda przyjmuje:
    - dwie tablice `int[] left`, `int[] right`
- Tworzysz:
    - nową tablicę wynikową
    - indeksy dla lewej, prawej i wynikowej tablicy
- Porównujesz elementy:
    - mniejszy element trafia do tablicy wynikowej
- Po zakończeniu porównań:
    - dopisz pozostałe elementy z lewej lub prawej tablicy

Ta metoda **nie używa rekurencji**

Zadanie 5: Połączenie MergeSort z Merge

Cel: złożyć algorytm w całość.

- W metodzie `MergeSort`:
    - po posortowaniu lewej i prawej tablicy
    - wywołaj `Merge(left, right)`
    - zwróć wynik scalania

✔️ Tu powstaje **pełny Merge Sort**

Zadanie 6: Testowanie algorytmu

Cel: upewnić się, że wszystko działa.

- W `Main`:
    - wypisz tablicę przed sortowaniem
    - wywołaj `MergeSort`
    - wypisz tablicę po sortowaniu
- Przetestuj:
    - pustą tablicę
    - tablicę z jednym elementem
    - tablicę z duplikatami
    - tablicę odwrotnie posortowaną