Laboratorium AiSD

Lista 5

Proste algorytmy sortowania

Proszę pamiętać, że **część rozwiązania** zadania stanowi również **zestaw testów** zaimplementowanych algorytmów i/lub struktur danych. Dodatkowo, proszę zwracać uwagę na **powtarzające się fragmenty** kodu i wydzielać je do osobnych funkcji/klas.

1. Wykorzystując przygotowaną paczkę kodu zaimplementuj i przetestuj następujące warianty poznanych algorytmów:
   a. Sortowanie przez wstawianie z przeszukiwaniem binarnym (*ang. binary search*),
   b. Sortowanie przez wybór z jednoczesnym wyszukiwaniem minimum i maksimum,
   c. Sortowanie koktajlowe (*ang. Shaker sort*) jako modyfikacja sortowania bąbelkowego.

   Na potrzeby zadania przygotowano paczkę kodu nazwaną **SortingTester** (wersja 2.0) zawierającą zestaw klas do testowania algorytmów sortowania.

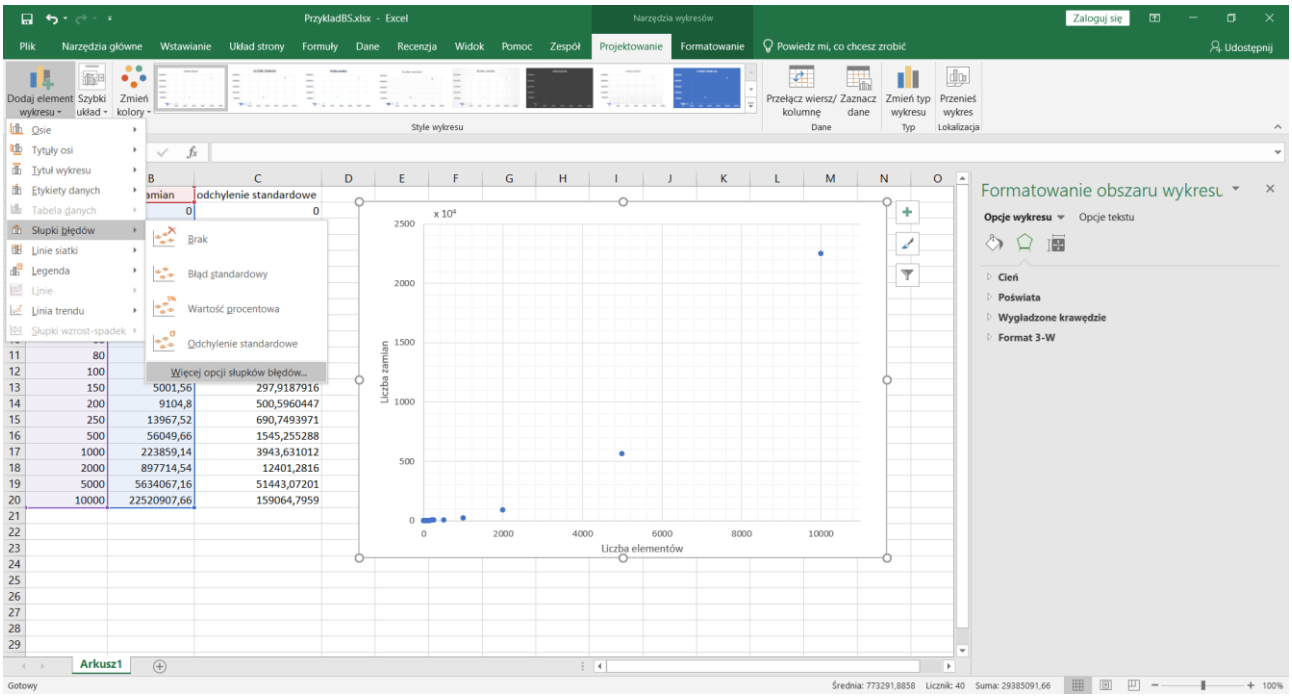
   Najważniejsze klasy/pakiety:

   - ***AbstractSwappingSortingAlgorithm<T>*** - abstrakcyjna klasa bazowa dla algorytmów sortowania opartych o zamianę elementów, zamieszczona w pakiecie *core*. Najważniejsze metody to *swap* i *compare* – klasa automatycznie zlicza ich użycia. W klasie pochodnej zdefiniować metodę abstrakcyjną *List<T> sort(List<T> list)*,
   - **testing.generation** – pakiet zawierający klasy generujące przypadki testowe (listy) do sortowania:
     - **OrderedIntegerArrayGenerator** – klasa generująca tablicę (*ArrayList<Integer>*) kolejnych *N* liczb całkowitych od 0 do *N* – 1,
     - **ReversedIntegerArrayGenerator** – klasa generująca tablicę (*ArrayList<Integer>*) kolejnych *N* liczb całkowitych od *N* - 1 do 0,

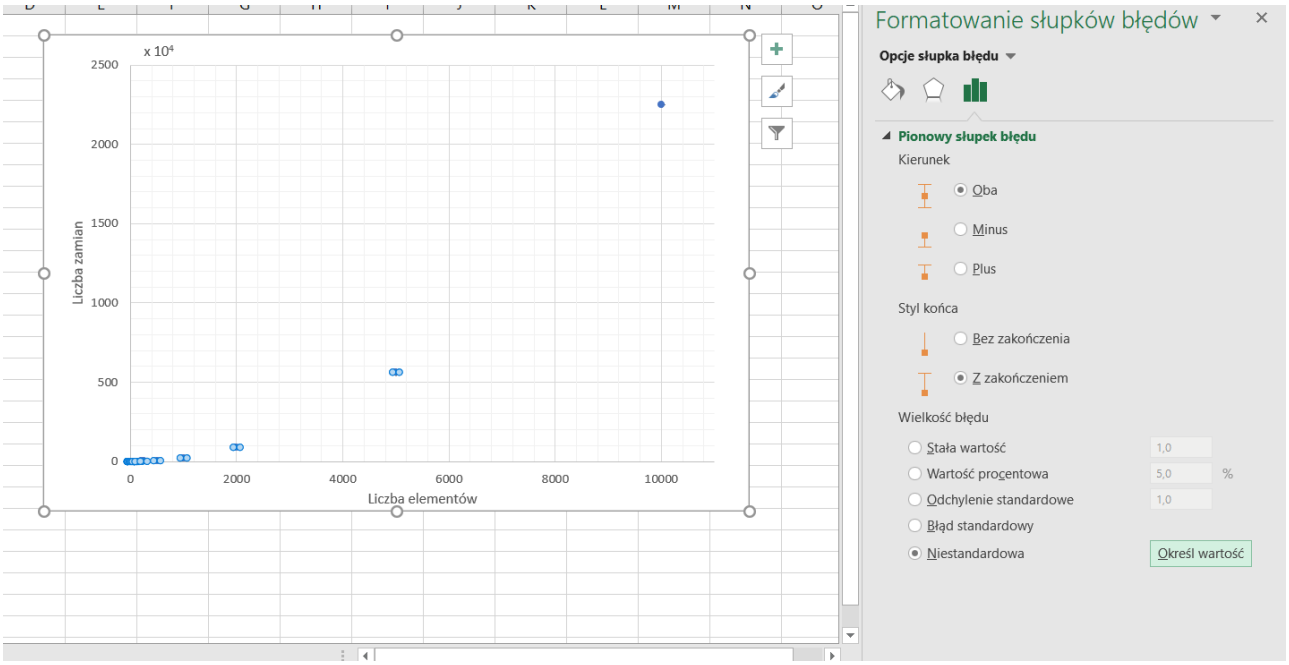
    - **ShuffledIntegerArrayGenerator** – klasa generująca tablicę (*ArrayList<Integer>*) *N* liczb całkowitych od 0 do *N* – 1 ustawionych w losowej kolejności,
    - **RandomIntegerArrayGenerator** – klasa generująca tablicę (*ArrayList<Integer>*) *N* liczb całkowitych losowanych z przedziału od 0 do *maxValue* (parametr konstruktora). Liczby mogą się powtarzać,
- **testing.generation.conversion** – pakiet zawierający dwie klasy do konwersji generowanych list:
    - ***MarkingGenerator<T>*** - klasa opakowywująca (*ang. wrapper class*) inny generartor, zamykająca wartości w obiektach typu *MarkedValue<T>* (opisane dalej),
    - ***LinkedListGenerator<T>*** - klasa opakowywująca (*ang. wrapper class*) inny generator, zamieniająca listę wartości w listę dowiązaniową (*ang. Linked List*),
- **testing** – pakiet zawierający szereg klas wykorzystywanych do testowania algorytmów sortowania:
    - ***Tester*** – klasa zawierająca metody służące do testowania algorytmów. Metoda *runOnce* wykonuje pojedyncze uruchomienie procedury sortującej dla zadanego generatora i rozmiaru kolekcji. Zwraca obliczone metryki wykonania. Metoda *runNTimes* wykonuje *repetitions* – krotne uruchomienie metody *runOnce* (w każdym teście powstaje nowa kolekcja do posortowania) i zwraca średnie oraz odchylenia standardowe dla metryk z tych *repetitions* uruchomień,
    - ***MarkedValue<T>*** - klasa reprezentująca parę (wartość, marker). Marker jest liczbą całkowitą reprezentującą kolejność, w której ta sama wartość została wygenerowana. Klasa istotna dla sprawdzenia, czy sortowanie jest stabilne,
- **testing.results.swapping** – pakiet zawierający klasy wyników dla algorytmów bazujących na zamianie elementów:
    - ***RunResult*** – klasa zawierająca wynik (metryki) pojedynczego uruchomienia algorytmu:
        - Czas w milisekundach,
        - Liczbę porównań elementów (*ang. comparisons*),
        - Liczbę zamian elementów (*ang. swaps*),
        - Czy w wyniku działania algorytmu lista została posortowana,
        - Czy sortowanie jest stabilne,

- ***Result*** – klasa zawierająca statystyki metryk – średnie i odchylenia standardowe - z wielu uruchomień.

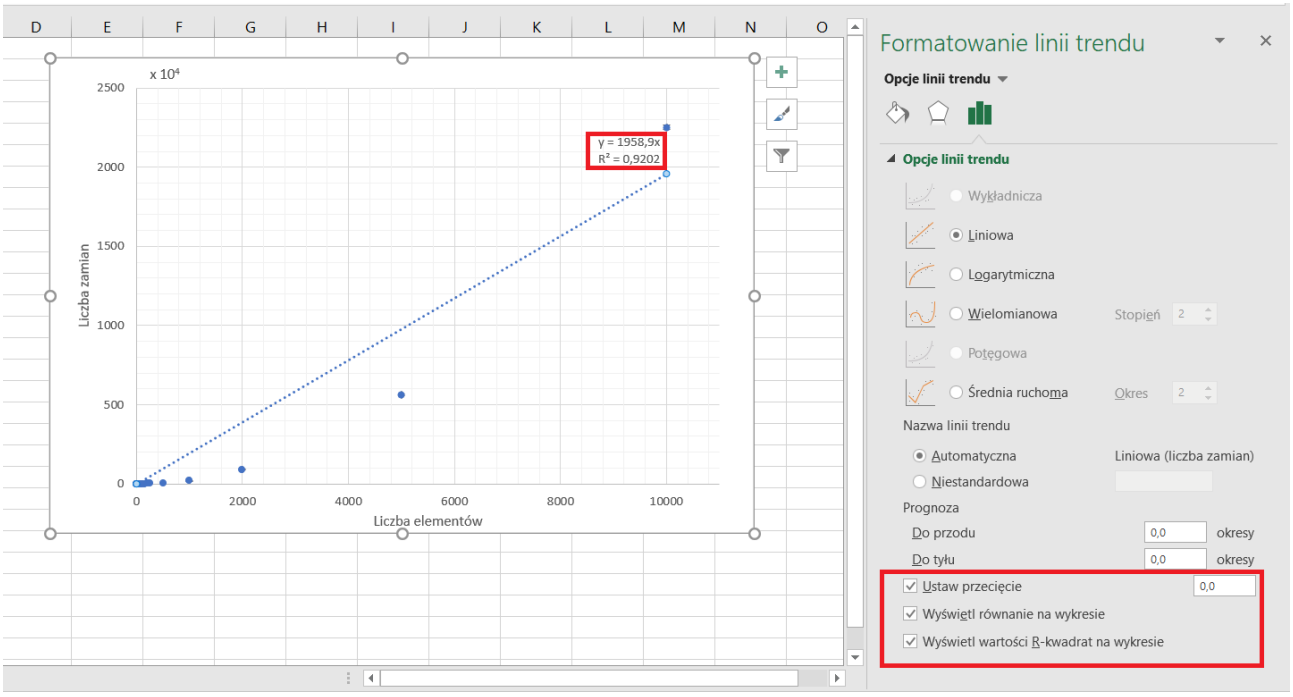
Na podstawie otrzymanych wyników (średnie i odchylenia standardowe) przygotować wykresy pokazujące zależność między **rodzajem** i **rozmiarem** wejścia, a zachowaniem metody. Przez rodzaj wejścia rozumiemy listy o różnych cechach – kolejność, powtórzenia elementów (z pakietu *testing.generation*). **Za liczbę powtórzeń przyjąć wartość 20**. Zastosować wykresy punktowe dla średnich, do punktów (serii) dodać **pionowe słupki błędów** reprezentujące odchylenie standardowe – błąd - pomiaru (**ustawić wartość otrzymaną z pomiaru, a NIE stosować opcji „odchylenie standardowe" z Excela**) (Rysunek 1 i 2). Dla danej serii dodać linię trendu pokazującą przybliżony kształt zależności między rozmiarem wejścia, a metryką. Zastosować wielomianową linię trendu o możliwie najmniejszej wartości wykładnika, ale dobrze oddającą charakter danych. Wykładnik 1 jest osobną opcją – liniową funkcją trendu. Wstawiając funkcję trendu wybrać opcje wyświetlania **wzoru funkcji** oraz **parametru R^2** – im wartość parametru bliższa 1, tym lepiej linia trendu oddaje chmurę punktów (Rysunek 3 i 4). Przykład poprawnego wykresu zamieszczono na Rysunku 5.



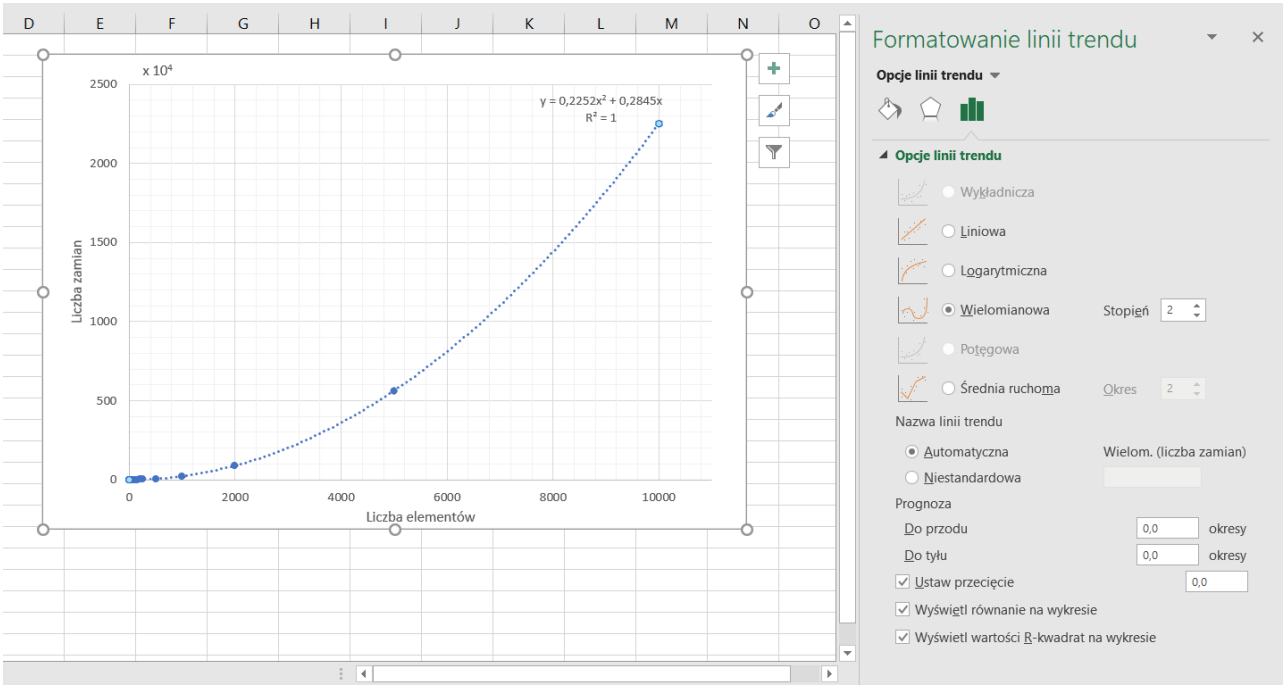
Rysunek 1 Wstawianie słupków błędów



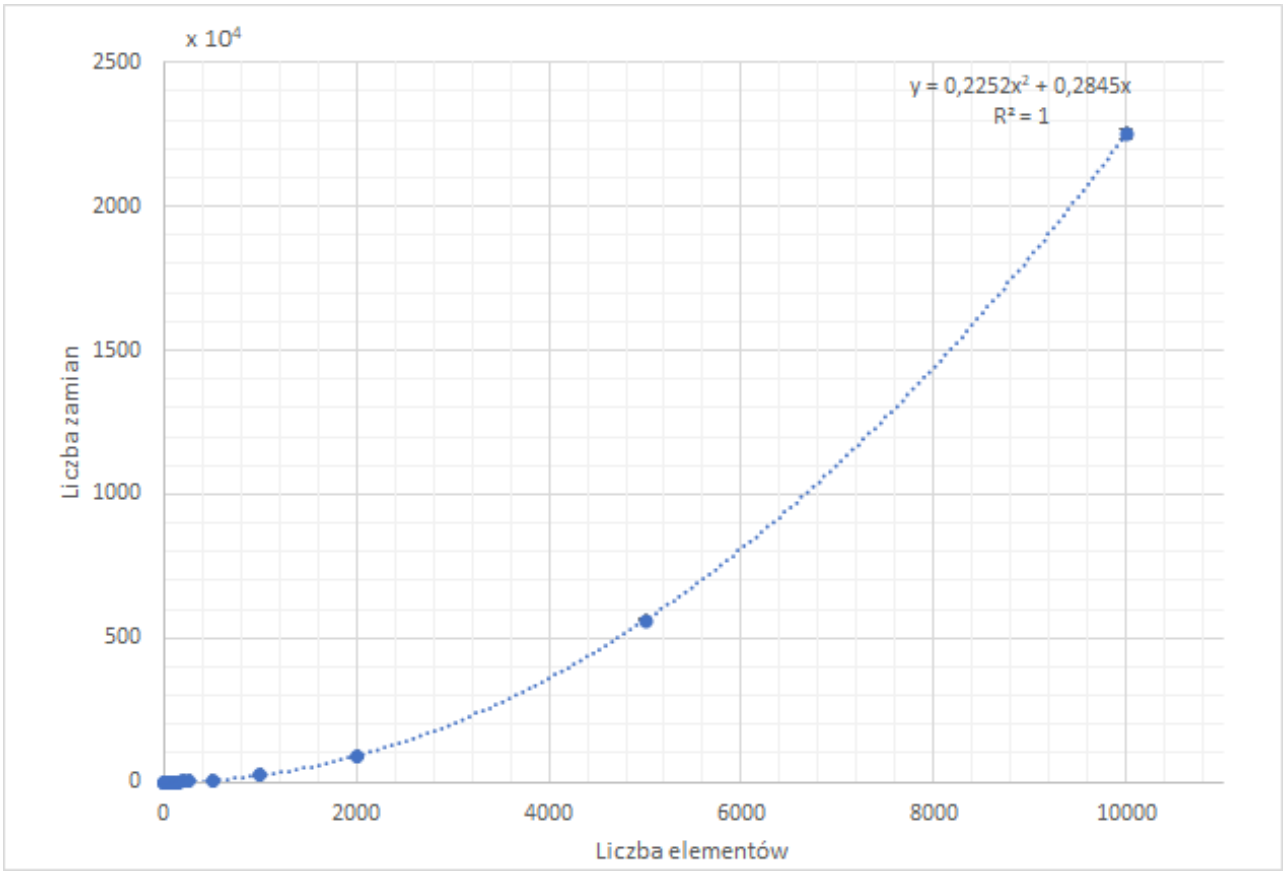
Rysunek 2 Opcje słupków błędów - wybór wartości



Rysunek 3 Opcje linii trendu



Rysunek 4 Dobór linii trendu



Rysunek 5 Przykład dobrze przygotowanego wykresu (słupki błędu prawie niewidoczne). Wykres zależności liczby zamian od rozmiaru danych dla sortowania bąbelkowego