Zadania – zajęcia 3

Przedmiotem zajęć będzie analiza algorytmu sortowania tablic – w tym przypadku na początek algorytmu sortowania bąbelkowego. Szczegóły i schemat blokowy zawarte są w prezentacji.

- 1. Napisać funkcję **`BubbleSort1`** sortującą niemalejąco daną tablicę liczb typu int o zadanym rozmiarze `length`. Funkcja powinna implementować algorytm sortowania według **pierwszego** slajdu. **Funkcja powinna zliczać wykonane operacje porównań elementów tablicy między sobą (proszę nie zliczać innych porównań) i zwracać tę liczbę (przez wskaźnik lub przez referencję). Proszę osobno napisać funkcję Swap do zamiany elementów o indeksach i oraz j miejscami w tablicy o nazwie tab i użyć jej w funkcji sortującej (tak jak na schematach blokowych zawartych w prezentacji).**

- 2. Napisać funkcję **`BubbleSort2`** sortującą niemalejąco daną tablicę liczb typu int o zadanym rozmiarze `length`. Funkcja powinna implementować algorytm sortowania według **drugiego** slajdu. **Funkcja powinna zliczać wykonane operacje porównań elementów tablicy między sobą (proszę nie zliczać innych porównań) i zwracać tę liczbę (przez wskaźnik lub przez referencję).**

Wyjaśnienie: celem pisania dalszych funkcji będzie badanie zachowania funkcji **`BubbleSort1`** i **`BubbleSort2`**. Chodzi o to, aby określić jak liczba porównań potrzebnych do posortowania tablicy zależy od rozmiaru tablicy. Jest to taka sama procedura jak w przypadku analizy algorytmu przeszukiwania liniowego. Sortowaniu będą podlegać tablice wypełnione liczbami losowymi (te same co w przypadku przeszukiwania – odpowiednie funkcje generujące te tablice są już gotowe). Sens funkcji pisanych w dalszych zadaniach jest również taki sam jak dla przypadku wyszukiwania liniowego.

- 3. Napisać funkcję o nazwie **`BubbleSortStatistics1`** której zadaniem będzie wielokrotne powtórzenie losowania tablicy zadanego rozmiaru oraz posortowanie jej przez wywołanie funkcji **`BubbleSort1`**.

Funkcja **`BubbleSortStatistics1`** powinna przyjmować argument `max` określający, ile razy ma być powtórzone losowanie tablicy i sortowanie jej, a także argument `length` określający rozmiar badanych tablic. Funkcja

BubbleSortStatistics1 powinna zwracać średnią wartość liczby porównań oraz odchylenie standardowe liczby porównań dla tablicy o danym rozmiarze (jako liczby typu `float`).
Z liczb porównań trzeba zatem policzyć średnią (czyli policzyć sumę liczb porównań i podzielić przez max – pamiętając, aby wynik był liczbą typu `float`). Następnie trzeba policzyć z tych liczb średni kwadrat (czyli policzyć sumę kwadratów liczb porównań i podzielić przez max – pamiętając, aby wynik był liczbą typu `float`).
Uwaga: proszę do zliczeń używać zmiennych typu `long long int`.
Na podstawie średniej i średniego kwadratu należy policzyć odchylenie standardowe. Jeśli `mean` jest średnią liczbą porównań, a `mean2` średnim kwadratem liczby porównań, to odchylenie standardowe wyraża się następującym wzorem: `sd = sqrt(mean2-mean*mean)`. Uwaga: do użycia `sqrt` potrzebna jest biblioteka `cmath`.

Dalej chodzi o to, aby zebrać dane dla rozmiarów tablic z zakresu 10...1000. Czyli trzeba wywołać funkcję **BubbleSortStatistics1** dla każdego z rozmiarów tablic i zapisać w pliku tekstowym w kolejnych wierszach następujące dane:

rozmiar tablicy średnia średnia-odchylenie standardowe średnia+odchylenie standardowe

W tym celu napiszemy kolejną funkcję:

- 4. Napisać funkcję **TestBubbleSort1**, która wywoła funkcję **BubbleSortStatistics1** dla rozmiarów tablic równych 10, 20, 30, ... ,1000 (ewentualnie można użyć większego od 1000 górnego rozmiaru tablic – zależy od czasu działania programu). Dla każdego z tych rozmiarów tablic powinno zostać wylosowane po 100 tablic i policzone średnie liczby porównań oraz odchylenia standardowe. Dane wyjściowe dla każdego rozmiaru tablicy powinny zostać zapisane w pliku tekstowym o nazwie „bubblesort1.dat", którego poszczególne wiersze powinny zawierać po kolei: rozmiar tablicy, średnią liczbę porównań, średnią liczbę porównań-odchylenie standardowe, średnią liczbę porównań+odchylenie standardowe.

Przykładowy plik danych powinien zawierać tego typu dane:

```
10      4,9873    2,39139     7,58321
20      16,9865   10,55199    25,421
.....
```

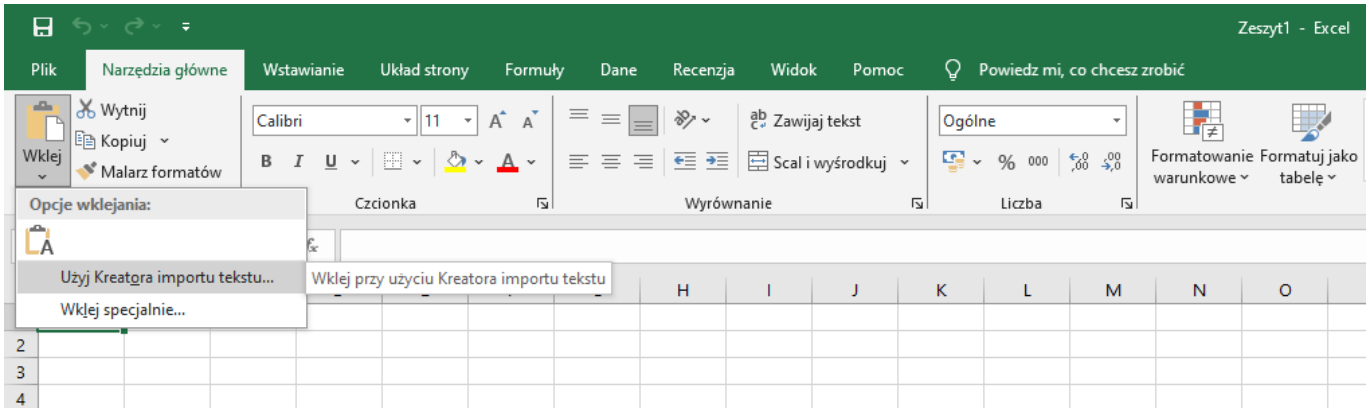
Mając plik proszę sporządzić w arkuszu kalkulacyjnym (Excel/Libre Office) wykres (powinien być to wykres typu punktowego) dla każdej z kolumn. Proszę do środkowej serii danych dodać linię trendu (wielomianową-stopnia 2, z opcją pokazania równania linii trendu).

- 5. Proszę (tak samo jak w punktach 3 i 4) napisać funkcje **`BubbleSortStatistics2`** oraz **`TestBubbleSort2`** realizujące te same zadania, tym razem dla funkcji sortującej **`BubbleSort2`**. Proszę zebrać analogiczne dane w pliku tekstowym o nazwie „bubblesort2.dat" i wykonać wspólny wykres dla danych z plików „bubblesort1.dat" i „bubblesort2.dat".

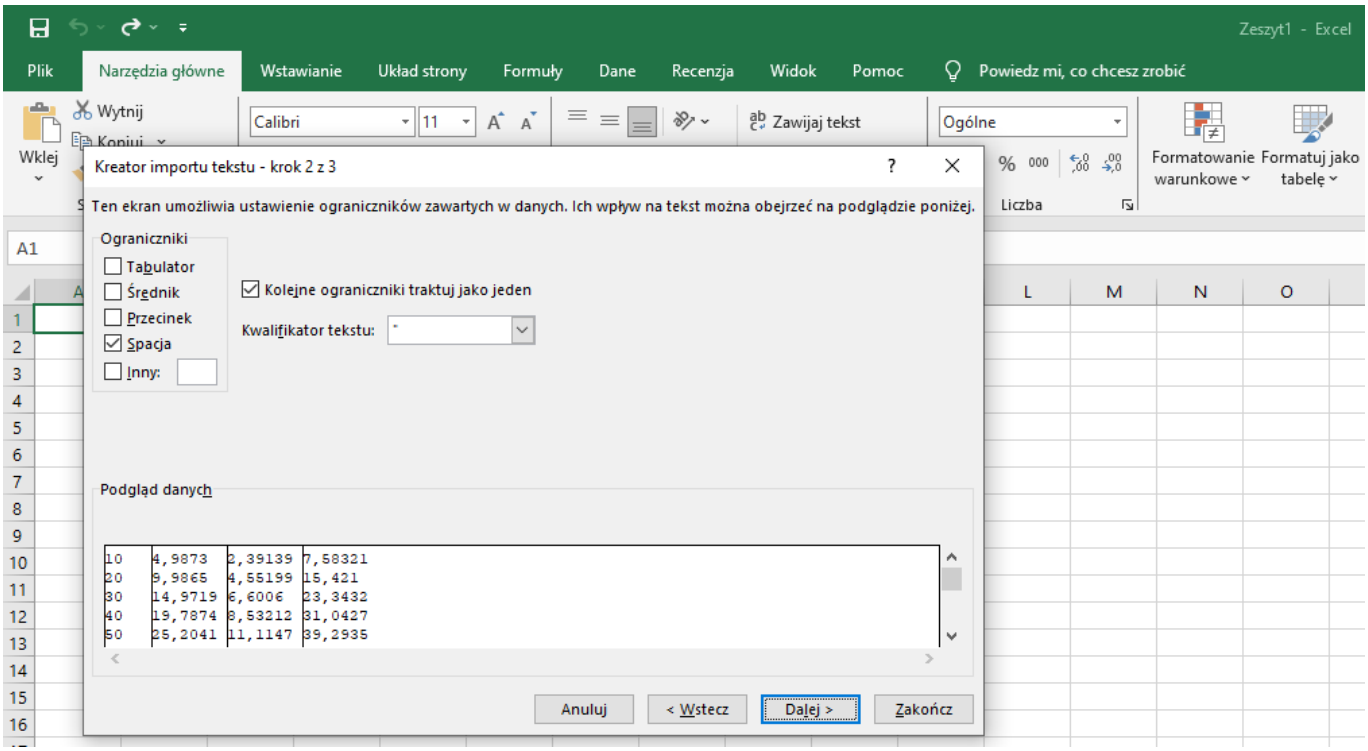
Dokładny sposób sporządzania wykresu (Excel):

1. Należy otworzyć zapisany plik tekstowy z danymi (np. w Notatniku), a następnie dokonać zamiany wszystkich kropek na przecinki (Ctrl+H, następnie wybrać znajdź . zamień na ,). Chodzi tu o poprawny odczyt danych przez arkusz z ustawiona polską wersją językową. Następnie zaznaczyć całą zawartość pliku (Ctrl+A) i skopiować (Ctrl+C).

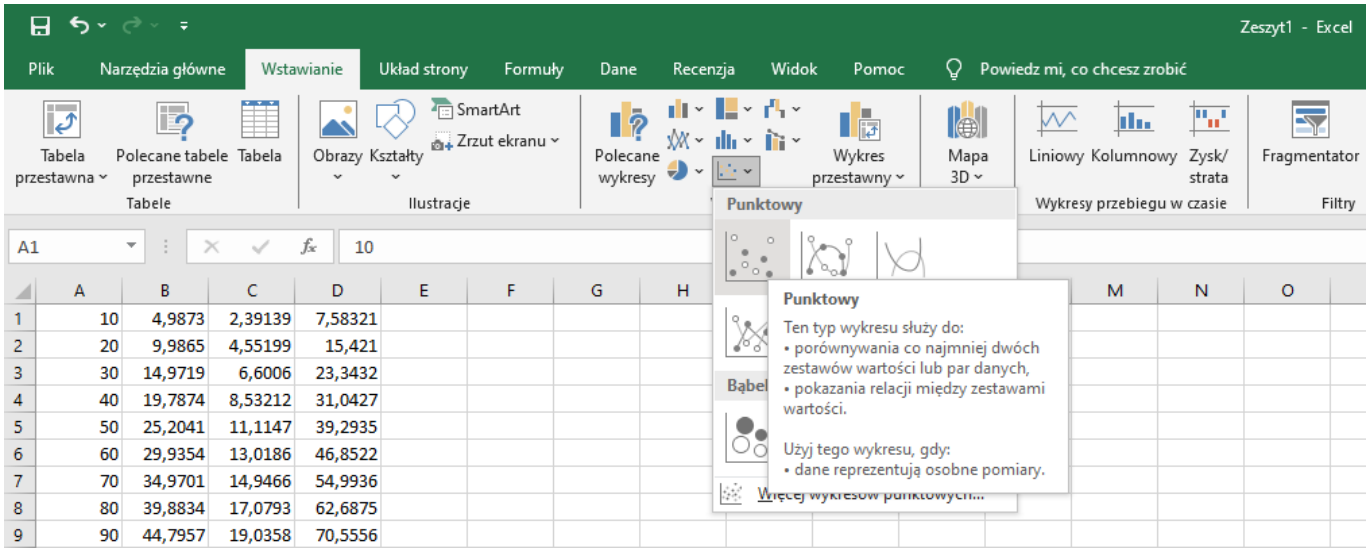
2. W pustym arkuszu kalkulacyjnym wybrać menu Wklej → Użyj kreatora importu tekstu.



Dalej wybrać typ pliku: rozdzielany i na następnym ekranie ogranicznik: spacja (lub wypróbować tabulator – sprawdzając na podglądzie, czy kolumny pliku są poprawnie zidentyfikowane). Potem można wybrać Zakończ.



3. Zaznaczyć wszystkie kolumny danych i z menu Wstawianie wybrać Wykresy → Punktowy.



4. Po wstawieniu wykresu zaznaczyć na nim środkową serię danych klikając i wybrać Dodaj linię trendu. Wskazać linię trendu wielomianową stopnia 2 i zaznaczyć opcję Wyświetl równanie na wykresie.

Celem tych zajęć laboratoryjnych jest ostatecznie wykonanie wykresu prezentującego otrzymane dane. Powinien on przedstawiać **zależność średniej liczby operacji porównań potrzebnej do posortowania tablicy metodą sortowania bąbelkowego (w obu wersjach) od rozmiaru tej tablicy**.

Dodatek. Komentarz do prezentacji.

Będziemy rozważać problem **sortowania tablicy** o długości length, zawierającej liczby, w kolejności niemalejącej (tablica może zawierać więcej niż jedną liczbę o danej wartości, więc nie mówimy wtedy o kolejności rosnącej, skoro nie wszystkie liczby są różne).

Jeden z najprostszych algorytmów sortowania to algorytm **sortowania bąbelkowego**.

Slajd 1:

Algorytm ten działa następująco: dokonujemy przebiegu tablicy od początku do końca (zmienna i na schemacie blokowym zmienia się od i←0 do i←length-2). Badamy kolejno pary elementów tablicy: tab[i] oraz tab[i+1]. Jeśli tab[i]>tab[i+1], to elementy te są w niewłaściwej kolejności (sortujemy niemalejąco!) i wtedy trzeba je zamienić miejscami. Jeśli kolejność elementów jest dobra, to przechodzimy do następnej pary.

Służy do tego funkcja Swap. Swap(tab,i,j) zamienia miejscami element tab[i] oraz tab[j] dla tablicy tab. Wszystkie funkcje sortujące bazują na zamianie elementów w tablicy miejscami, stąd taka pomocnicza funkcja.

Indeks i zmienia się od i←0 do length-2 (bo badamy pary tab[i],tab[i+1], czyli ostatnia badana para to tab[length-2], tab[length-1]). Wykonujemy pełen przebieg tablicy, badając wszystkie pary od tab[0],tab[1] do tab[length-2], tab[length-1]. Po takim przejściu całej tablicy na pewno największy jej element zostanie przesunięty na sam koniec tablicy.

Wyobraźmy sobie, że sortujemy niemalejąco taką tablicę (indeksujemy od 0 do 5, length=6):

10 8 5 12 9 3

Przy zmienianiu i od 0 do 4 badamy pary od tab[0],tab[1] do tab[4],tab[5] i po kolei wykonujemy zamiany.

Jeden przebieg dla i od 0 do 4 ma taką postać (badana para zaznaczona na żółto):

8 10 5 12 9 3 (po zbadaniu pary tab[0],tab[1] zamiana 8 i 10)
8 5 10 12 9 3 (po zbadaniu pary tab[1],tab[2] zamiana 10 i 5)
8 5 10 12 9 3 (po zbadaniu pary tab[2],tab[3] kolejność ok)
8 5 10 9 12 3 (po zbadaniu pary tab[3],tab[4] zamiana 12 i 9)
8 5 10 9 3 12 (po zbadaniu pary tab[4],tab[5] zamiana 3 i 12).

W wyniku jednorazowego przejścia całej tablicy największy element 12 znalazł się na swoim miejscu (na końcu tablicy).

Teraz przechodzimy tablicę drugi raz (ma ona po pierwszym przejściu postać
8 5 10 9 3 12)

5 8 10 9 3 12 (po zbadaniu pary tab[0],tab[1] zamiana 8 i 5)
5 8 10 9 3 12 (po zbadaniu pary tab[1],tab[2] kolejność ok)
5 8 9 10 3 12 (po zbadaniu pary tab[2],tab[3] zamiana 9 i 10)
5 8 9 3 10 12 (po zbadaniu pary tab[3],tab[4] zamiana 3 i 10)
5 8 9 3 10 12 (po zbadaniu pary tab[4],tab[5] kolejność ok).

Po drugim przejściu drugi największy element 10 znalazł się na swoim miejscu (drugi od końca w tablicy).

W taki sam sposób wykonujemy kolejne przejścia tablicy...ale jak długo?

Jeśli wykonamy pełen przebieg tablicy (czyli zbadamy wszystkie pary od tab[0],tab[1] do tab[length-2] i tab[length-1]) i nie wykonamy żadnej zamiany, to znaczy że wszystkie elementy są już we właściwej kolejności i tablica jest już posortowana - można zakończyć sortowanie.

Do określania czy przy przejściu całej tablicy były dokonane jakieś zamiany służy zmienna ch. Przed przechodzeniem tablicy ustawiamy ch←0 i jeśli przechodząc tablicę (czyli zmieniając i od i←0 do length-2) dokonujemy jakiejkolwiek zmiany, to przypisujemy ch←1.
Zatem wykonujemy przejścia całej tablicy od i←0 do i←length-2 tak długo, jak długo w przejściu takim wykonana zostaje jakaś zamiana. Kiedy nie wykonamy już żadnej zamiany to ch pozostanie równe 0 i można zakończyć sortowanie.

Slajd 2:

Zauważmy, że skoro w każdym pełnym przebiegu przez tablicę ustawiamy co najmniej jeden element na właściwym miejscu, to tak naprawdę w kolejnych przebiegach nie musimy badać wszystkich par dla i od i←0 do i←length-2. W każdym przebiegu możemy zmniejszać maksymalną wartość i o jeden (bo po 1 przebiegu jeden element jest na swoim miejscu, po 2 już dwa, po 3 trzy...itd.) Zatem wprowadzamy zmienną p numerującą przebiegi tablicy, po każdym przebiegu tablicy zwiększamy p o jeden. Zakres indeksów tablicy do przechodzenia to w takim razie od i←0 do i←length-2-p (gdzie p to liczba poprzednio wykonanych przejść).
W ten sposób unikamy zbędnego badania par na końcu tablicy, które to pary już na pewno zawierają elementy ustawione w poprawnej kolejności.

Przykład:
pierwszy przebieg (i od 0 do 4):
8 10 5 12 9 3 (po zbadaniu pary tab[0],tab[1] zamiana 8 i 10)
8 5 10 12 9 3 (po zbadaniu pary tab[1],tab[2] zamiana 10 i 5)
8 5 10 12 9 3 (po zbadaniu pary tab[2],tab[3] kolejność ok)
8 5 10 9 12 3 (po zbadaniu pary tab[3],tab[4] zamiana 12 i 9)
8 5 10 9 3 12 (po zbadaniu pary tab[4],tab[5] zamiana 3 i 12).

drugi przebieg (wystarczy i od 0 do 3)

5 8 10 9 3 12 (po zbadaniu pary tab[0],tab[1] zamiana 8 i 5)
5 8 10 9 3 12 (po zbadaniu pary tab[1],tab[2] kolejność ok)
5 8 9 10 3 12 (po zbadaniu pary tab[2],tab[3] zamiana 9 i 10)
5 8 9 3 10 12 (po zbadaniu pary tab[3],tab[4] zamiana 3 i 10)
5 8 9 3 10 12 (ta para ma już na pewno dobrą kolejność i nie trzeba jej badać!)

trzeci przebieg (wystarczy i od 0 do 2):

5 8 9 3 10 12 (po zbadaniu pary tab[0],tab[1] kolejność ok)
5 8 9 3 10 12 (po zbadaniu pary tab[1],tab[2] kolejność ok)
5 8 3 9 10 12 (po zbadaniu pary tab[2],tab[3] zamiana 9 i 3)
5 8 3 9 10 12 **(te dwie pary mają już dobrą kolejność i nie trzeba ich badać)**

czwarty przebieg (wystarczy i od 0 do 1):

5 8 3 9 10 12 (po zbadaniu pary tab[0],tab[1] kolejność ok)
5 3 8 9 10 12 (po zbadaniu pary tab[1],tab[2] zamiana 3 i 8)
5 3 8 9 10 12 **(te trzy pary mają już dobrą kolejność i nie trzeba ich badać)**

piąty przebieg (wystarczy i od 0 do 0, tylko jedna para):

3 5 8 9 10 12 (po zbadaniu pary tab[0],tab[1] zamiana 3 i 5)
3 5 8 9 10 12 **(te cztery pary mają już dobrą kolejność i nie trzeba ich badać)**

Tablica już posortowana!

Zatem par zaznaczonych na zielono już nie musimy badać, a w ten sposób wykonujemy mniej operacji i algorytm jest bardziej efektywny.