🧩 TiDA 01: Sortowanie bąbelkowe: indeksy i wskaźniki (wersja podstawowa)

🎯 Cel zajęć

Na tych zajęciach:

- Zrozumiesz działanie **sortowania bąbelkowego** (bubble sort).
- Porównasz wersję z **indeksami** i wersję ze **wskaźnikami**.
- Nauczysz się **mierzyć czas wykonania programu**.
- Zrozumiesz różnice między **Debug** i **Release**.
- Nauczysz się korzystać z **debuggera** w Visual Studio.

🧠 CZĘŚĆ TEORETYCZNA

🔷 Co robi bubble sort?

Bubble sort to prosty algorytm sortowania.

Działa tak:

1. Porównuje dwa sąsiednie elementy tablicy.
2. Jeśli są w złej kolejności – zamienia je miejscami.
3. Po każdym przejściu największy element „opada” na koniec tablicy, najmniejsze („najlżejsze”) niczym bąbelki „wypływają” na powierzchnię.
4. Powtarza aż wszystko będzie posortowane.

Kod programu – krok po kroku

Fragment 1 – nagłówki i początek programu

```
#include <iostream>
#include <vector>
#include <chrono>   // do mierzenia czasu
#include <cstdlib>  // do losowania liczb
using namespace std;

int main() {
    cout << "Porownanie sortowania babelkowego: indeksy vs wskazniki\n";
```

Wyjaśnienie:

- `chrono` – deklaracje funkcji do mierzenia czasu.
- `using namespace std;` – pozwala pisać krócej (`cout` zamiast `std::cout`).

Fragment 2 – przygotowanie danych

```
const int N = 1000; // liczba elementów
vector<int> dane(N);
srand(123); // stałe ziarno losowania

for (int i = 0; i < N; i++) {
    dane[i] = rand() % 10000;
}

vector<int> dane_kopia = dane; // kopia dla drugiego testu
```

Wyjaśnienie:

- Tworzymy wektor dane z losowymi wartościami.
- Kopiujemy tablicę, aby druga metoda miała te same dane.

Fragment 3 – bubble sort (indeksy)

```
auto start1 = chrono::high_resolution_clock::now();

for (int i = 0; i < N - 1; i++) {
    for (int j = 0; j < N - 1 - i; j++) {
        if (dane[j] > dane[j + 1]) {
            int temp = dane[j];
            dane[j] = dane[j + 1];
            dane[j + 1] = temp;
        }
    }
}

auto stop1 = chrono::high_resolution_clock::now();
auto czas1 = chrono::duration_cast<chrono::milliseconds>(stop1 - start1).count();

cout << "Czas sortowania (indeksy): " << czas1 << " ms\n";
```

Wyjaśnienie:

- Dwie pętle – zewnętrzna i wewnętrzna.
- Porównujemy dane[j] i dane[j+1].
- Mierzymy czas w milisekundach.

Fragment 4 – bubble sort (wskaźniki)

```
    int* tab = dane_kopia.data(); // wskaźnik do pierwszego elementu
    auto start2 = chrono::high_resolution_clock::now();

    for (int i = 0; i < N - 1; i++) {
        for (int* p = tab; p < tab + (N - 1 - i); p++) {
            if (*p > *(p + 1)) {
                int temp = *p;
                *p = *(p + 1);
                *(p + 1) = temp;
            }
        }
    }

    auto stop2 = chrono::high_resolution_clock::now();
    auto czas2 = chrono::duration_cast<chrono::milliseconds>(stop2 - start2).count();

    cout << "Czas sortowania (wskazniki): " << czas2 << " ms\n";
```

Wyjaśnienie:

- `int* p` – wskaźnik (adres elementu w pamięci).
- `*p` – wartość pod tym adresem.
- `*(p + 1)` – następny element.
- Porównujemy i zamieniamy miejscami wartości przez wskaźniki.

Fragment 5 – zakończenie programu

```
    cout << "Koniec programu." << endl;
    return 0;
}
```

Debug vs Release

Tryb	Cechy	Zastosowanie
Debug	Wolniejszy, więcej informacji do debugowania.	Do szukania błędów.
Release	Szybszy, kompilator stosuje optymalizacje.	Do testów wydajności.

Inlining – kompilator może wkleić treść krótkiej funkcji w miejscu wywołania.
Rejestry – bardzo szybka pamięć w procesorze używana do przechowywania bieżących danych.
Aliasing – dwa wskaźniki mogą wskazywać ten sam fragment pamięci, co utrudnia optymalizacje.

Debugger w Visual Studio

1. Otwórz projekt w **Visual Studio**.
2. Ustaw **breakpointy** (klik w lewy margines):
   - początek pętli `for (int i = 0; i < N - 1; i++)`,
   - linia z `if (dane[j] > dane[j + 1])`,
   - linia z `if (*p > *(p + 1))`.
3. Uruchom w trybie **Debug (F5)**.
4. Obserwuj zmienne w oknach **Locals** lub **Watch**.
5. Użyj **F10** (Step Over) i **F11** (Step Into) do przechodzenia po liniach.
6. Porównaj działanie programu w **Debug** i **Release (Ctrl+F5)**.

Cache i aliasing (dla ciekawych)

- **Cache** – pamięć podręczna procesora, dzięki której dane używane często wczytują się szybciej.
- Nawet ten sam program może mieć różne czasy działania, zależnie od stanu cache.

ZADANIA

1. **Przepisz** kod do własnego projektu w Visual Studio.
2. **Uruchom w trybie Debug (F5)**:
   - ustaw breakpointy,
   - sprawdź wartości wskaźników i elementów,
   - zrób zrzut ekranu z okna Watch lub Locals.
3. **Uruchom w trybie Release (Ctrl+F5)**:
   - zapisz czasy obu wersji sortowania,
   - uruchom kilka razy i porównaj wyniki.
4. **Zapisz krótkie wnioski:**
   - która wersja była szybsza,
   - dlaczego wyniki się różnią,
   - dlaczego pomiar w Debugu nie jest wiarygodny.
5. **(Dla chętnych)**: zwiększ `N` do 2000 lub 5000 i sprawdź, jak rośnie czas działania.