

# Sprawozdanie z zajęć (WZÓR)

**Temat:** Tablice dynamiczne i statyczne, stos i sterta, testy funkcjonalne i wydajnościowe

**Uczeń:** Jan Kowalski

**Data:** 26.10.2025

**Klasa:** 4D (technik programista)

---

## Cel ćwiczenia

1. Zrozumienie różnicy między tablicą **statyczną** (alokowaną na stosie) a **dynamiczną** (alokowaną na stercie).
  2. Zbadanie wpływu rozmiaru tablicy na czas sortowania.
  3. Wykonanie testów funkcjonalnych i wydajnościowych.
  4. Porównanie działania programu w architekturach **x86 (32-bit)** i **x64 (64-bit)**.
  5. Zidentyfikowanie limitu stosu (moment wystąpienia błędu *stack overflow*).
- 

## Środowisko testowe

Parametr	Wartość
System operacyjny	Windows 11 64-bit
Kompilator	Microsoft Visual C++ 2022
Tryby komplikacji	x86 (32-bit), x64 (64-bit)
Typ projektu	Konsolowa aplikacja C++
Procesor	Intel Core i5-10400F @ 2.9 GHz
Pamięć RAM	16 GB

---

## Opis programu

Program sortuje tablicę liczb losowych metodą **sortowania bąbelkowego (Bubble Sort)** i mierzy czas wykonania. Testuje dwa przypadki:

1. **Tablica statyczna** – utworzona na stosie.
2. **Tablica dynamiczna** – utworzona na stercie (za pomocą `new` i `delete`).

Dodatkowo program wykonuje **test funkcjonalny**, który sprawdza:

- czy tablica istnieje,
- czy zawiera wartości z zadanego zakresu (0–1000),
- czy jest posortowana rosnąco.

Pomiar czasu jest wykonywany w mikrosekundach przy pomocy biblioteki `<chrono>`.

---

## Przebieg doświadczenia

1. Uruchomiono program dla tablic o rozmiarach: **1000, 10 000, 100 000** elementów.
  2. Dla każdego rozmiaru wykonano dwa testy:
    - tablica **statyczna**,
    - tablica **dynamiczna**.
  3. Dodatkowo sprawdzono maksymalny rozmiar tablicy statycznej możliwy do zaallokowania bez błędu *stack overflow*.
  4. Program uruchomiono osobno w wersjach **x86** i **x64**.
- 

## Wyniki pomiarów

Tabela 1 – Czas sortowania (w mikrosekundach)

Rozmiar tablicy	Typ tablicy	Czas (x86)	Czas (x64)	Test funkcjonalny
1000	statyczna	4 560	3 890	 zaliczony
1000	dynamiczna	4 740	3 960	 zaliczony
10 000	statyczna	471 200	422 800	 zaliczony
10 000	dynamiczna	489 300	437 900	 zaliczony
100 000	statyczna	<i>brak – Stack overflow</i>	4 720 000	
100 000	dynamiczna	5 020 000	4 510 000	 zaliczony

---

Tabela 2 – Maksymalny rozmiar tablicy statycznej

Architektura	Maks. rozmiar (elementów)	Komentarz
x86 (32-bit)	~40 000	Błąd Stack Overflow powyżej tej wartości
x64 (64-bit)	~80 000	Większy stos pozwala na większe tablice

---

## Analiza wyników

1. **Tablice dynamiczne** są nieco wolniejsze od statycznych — wynika to z faktu, że znajdują się na stercie, a dostęp do niej jest mniej lokalny w pamięci (cache).
2. **Kod 64-bitowy** jest szybszy o ok. 10–15% – ma więcej rejestrów i lepszą optymalizację pamięci.
3. W architekturze **x86** wystąpił błąd *Stack Overflow* przy tablicy statycznej >40 000 elementów.
4. **Testy funkcjonalne** potwierdziły poprawne działanie programu:
  - tablice zawierały liczby w zakresie 0–1000,
  - po sortowaniu były rosnące,
  - nie wystąpiły błędy pamięci.
5. Wersja 64-bitowa poradziła sobie lepiej przy większych tablicach – stos ma większy domyślny rozmiar.

---

## Wnioski końcowe

1. **Tablice statyczne** są szybsze, ale ograniczone rozmiarem stosu.
  2. **Tablice dynamiczne** są bardziej elastyczne, działają na stercie, dzięki czemu mogą przechowywać bardzo duże ilości danych.
  3. Kod **x64** jest wydajniejszy i stabilniejszy dla dużych struktur danych.
  4. Warto stosować **testy funkcjonalne** po każdym sortowaniu, aby upewnić się, że program nie tylko działa szybko, ale też poprawnie.
  5. Błąd *stack overflow* jest naturalną granicą stosu i należy go unikać poprzez stosowanie alokacji dynamicznej dla dużych struktur.
- 

## Załącznik – przykładowe fragmenty kodu

```
bool czyTablicaPoprawna(int* tablica, int rozmiar, int MIN, int MAX) {  
    if (!tablica) return false;  
    for (int i = 0; i < rozmiar; i++)  
        if (tablica[i] < MIN || tablica[i] > MAX) return false;  
    return true;  
}  
  
bool czyTablicaPosortowana(int* tablica, int rozmiar) {  
    for (int i = 1; i < rozmiar; i++)  
        if (tablica[i - 1] > tablica[i]) return false;  
    return true;  
}
```