AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm MergeSort
z zastosowaniem testów jednostkowych

Autor:
Gargula Adrian

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

1.1. Cel Projektu

Projekt ma na celu implementacje algorytmu sortowania przez scalanie (Merge Sort) w postaci klasy w C++ oraz przetestowanie jego poprawności przy użyciu testów jednostkowych z wykorzystaniem Google Test. Projekt sprawdza, czy algorytm jest w stanie poprawnie sortować tablice liczb w różnych przypadkach.

1.2. Sortowanie przez Scalanie

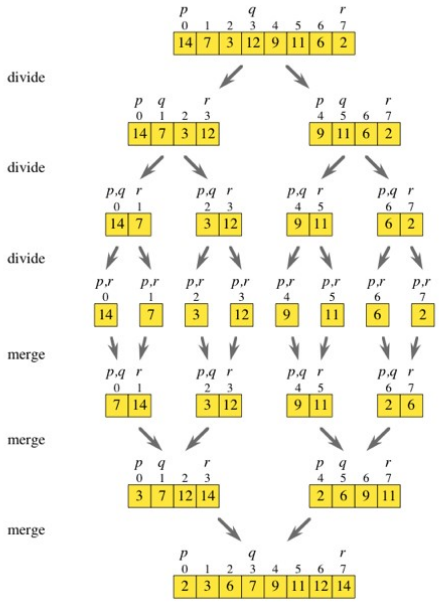
Merge Sort używa algorytmu dziel-i-rządź do sortowania, musimy zdecydować jak będą wyglądać nasze podproblemy..[1]

Oto jak sortowanie przez scalanie używa algorytmu dziel-i-rządź:

- Podziel przez znalezienie liczby q na pozycji w połowie pomiędzy p i r. Robimy to w ten sam sposób w jaki znaleźliśmy środek w przeszukiwaniu binarnym: zsumuj p i r, podziel przez 2, i zaokrąglij w dół.
- Rządź poprzez rekurencyjne sortowanie podtablic w każdym z dwóch podproblemów, stworzonych w kroku 1. Oznacza to: rekurencyjnie posortuj podtablicę array[p..q] i rekurencyjnie posortuj podtablicę array[q+1..r].
- Połącz przez scalanie obie posortowane podtablice z powrotem w jedną posortowaną podtablicę array[p..r].

[1] Khan Academy- Merge Sort[1]

Rysunek 1.1 przedstawia cały algorytm sortowania przez scalanie.



Rys. 1.1. Sortowanie przez scalanie

1.3. Operacja scalania

Operacja scalania polega na porównywaniu pierwszych elementów posortowanych podtablic i przenoszeniu mniejszego z nich (lub większego, jeśli sortujemy malejąco) do nowej tablicy. Jeśli w jednej z podtablic nie ma już elementów, trzeba kolejno przenosić do tablicy z wynikami kolejne elementy drugiej.[2]

1.4. Implementacja

Merge sort jest zaimplementowane w klasie, a jego funkcjonalność obsłużona metodami. W algorytmie wykorzystamy nastepujace metody:

- Scal,
- Sortuj,

[2]Operacje scalania [2]

1.5. Integracja z GitHubem

Projekt jest realizowany pojedynczo. Do pracy użyto systemu kontroli wersji GitHub. W projekcie należy wykonać nastepujace testy:

- Test sprawdza, czy algorytm zachowuje tablicę niezmienioną, gdy ona jest już posortowana rosnąco
- Test sprawdza, czy algorytm potrafi posortować tablicę, która jest posortowana w odwrotnej kolejności,
- Test sprawdza, czy algorytm poprawnie sortuje losową tablicę liczb,
- Test sprawdza, czy algorytm poprawnie sortuje tablice tylko z liczbami ujemnymi,
- Test sprawdza, czy algorytm poprawnie sortuje tablice z liczbami ujemnymi i liczbami dodatnimi,
- Test sprawdza, czy algorytm obsługuje pustą tablicę bez rzucania wyjątkiem,
- Test sprawdza, czy algorytm nie zmienia tablicy, która zawiera tylko jeden element,
- Test sprawdza, czy algorytm poprawnie sortuje tablicę z duplikatami liczb,
- Test sprawdza, czy algorytm poprawnie sortuje tablice ujemną z duplikatami,
- Test sprawdza, czy algorytm poprawnie sortuje tablice z liczbami ujemnymi, dodatnimi oraz duplikatami,
- Test sprawdza, czy algorytm poprawnie sortuje tablicę zawierającą tylko dwa elementy w kolejności rosnącej,
- Test sprawdza, czy algorytm poprawnie sortuje dużą tablicę zawierającą ponad 100 elementów,
- Test sprawdza, czy algorytm poprawnie sortuje dużą tablicę zawierającą ponad 100 elementów z liczbami ujemnymi, dodatnimi oraz duplikatami,

2. Analiza problemu

2.1. Zastosowanie sortowania przez scalanie

Algorytm sortowania przez scalanie (merge sort) znajduje zastosowanie w wielu sytuacjach, szczególnie tam, gdzie wymagane jest efektywne sortowanie dużych zbiorów danych. Poniżej przedstawiam kilka konkretnych zastosowań merge sort:

1. Sortowanie plików: Merge sort jest często używany do sortowania zawartości plików, zwłaszcza gdy dane nie mieszczą się w całości w pamięci operacyjnej. Działa on efektywnie w przypadku sortowania danych przechowywanych na dysku.

2. Aplikacje bazodanowe: W bazach danych, zwłaszcza przy sortowaniu dużych tabel, merge sort może być zastosowany do efektywnego uporządkowania rekordów według określonych kryteriów.

3. Systemy operacyjne: W systemach operacyjnych merge sort może być używany do sortowania informacji o plikach, katalogach lub innych strukturach systemu plików.

4. Aplikacje wielowątkowe: W środowiskach wielowątkowych merge sort może być używany do efektywnego sortowania danych równolegle, co przyspiesza proces sortowania.

Warto podkreślić, że merge sort jest algorytmem stabilnym i ma stałą złożoność czasową O(n log n), co sprawia, że jest on wybierany w sytuacjach, gdzie wymagane jest efektywne sortowanie dla dużych ilości danych.

2.2. Zalety i wady algorytmu

1. Zalety algorytmu
    - Prostota implementacji.
    - Wydajność.
    - Stabilność.
    - Algorytm sortuje zbiór n-elementowy w czasie proporcjonalnym do liczby nlogn
2. Wady algorytmu
    - Podczas scalania potrzebny jest dodatkowy obszar pamięci przechowujący kopie podtablic do scalenia

2.3. Działania algorytmu sortowania przez scalanie

1. Podział listy:
    - Algorytm zaczyna od podziału listy na dwie równoliczne (lub zbliżone) części. Wybiera element środkowy jako punkt podziału i dzieli listę na dwie podlisty: lewą i prawą.
2. Sortowanie rekurencyjne:
    - Następnie algorytm rekurencyjnie sortuje obie połowy listy. To oznacza, że dla lewej i prawej połowy listy ponownie stosuje ten sam algorytm sortowania przez scalanie.
3. Scalanie posortowanych podlist:
    - Gdy obie połowy są już posortowane, algorytm przechodzi do fazy scalania. Wykorzystuje dwie pomocnicze listy: jedną dla lewej połowy, drugą dla prawej.
4. Porównywanie i Scalanie:
    - Algorytm porównuje elementy z obu podlist i dodaje mniejszy z nich do wynikowej listy. W miarę dodawania elementów, wskaźniki są przesuwane, a porównywanie i dodawanie trwa, aż jedna z podlist się skończy.

5. Dodawanie pozostałych elementów:

    - Gdy jedna z podlist się zakończy, algorytm dodaje pozostałe elementy z drugiej podlisty do wynikowej listy.

6. Powtarzanie procesu:

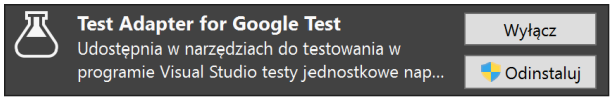
    - Cały proces podziału, rekurencyjnego sortowania i scalania jest powtarzany aż do momentu, gdy wszystkie podlisty są jednoelementowe. Wtedy zostają one scalane w jedną posortowaną listę.

7. Zwracanie posortowanej listy:

    - Ostatecznym wynikiem jest posortowana lista, która jest zwracana jako wynik algorytmu.

2.4. Google Test

Google Test znany także jako gtest to framework do testowania jednostkowego dla języka C++ widac to rozszerzenie na rysuneku 2.1 (s. 10). Jest to narzędzie, które umożliwia programistom pisanie, organizowanie i uruchamianie testów jednostkowych w ich kodzie C++. Poniżej znajduje opis podstawowego działania Google Test w C++ w środowisku Visual Studio Code. Testy jednostkowe uchodzą za najszybsze z wspomnianych przeze mnie typów testów. Testując kod jednostkowo, sprawdzamy jedynie mały fragment kodu (w przeciwieństwie do testów integracyjnych, gdzie możemy korzystać np. z mechanizmów automatycznego wstrzykiwania zależności, dyspozytora komunikatów itp).



Rys. 2.1. Rozszerzenie Google Test

Testując jednostkowo, próbujemy wejść w interakcję z wybraną przez nas jednostką, kontrolując jej zachowania. Do tych zachowań zaliczyć możemy:

- Zgodność parametrów przekazywanych do jednostki
- Zgodność zwracanych przez jednostkę wartości
- Wyrzucane przez jednostkę wyjątki

2.5. Jak działa Google Test

Pisanie testów W Google Test testy są definiowane w specjalnych funkcjach testowych. Każdy test opisuje jedno konkretne zachowanie kodu i sprawdza, czy działa ono zgodnie z oczekiwaniami. Testy są organizowane w tzw. test suites, czyli zestawy testów grupujących testy o podobnej tematyce.
Google Test oferuje zestaw wbudowanych makr do przeprowadzania asercji. Asercje pozwalają porównywać wyniki rzeczywiste z oczekiwanymi. Oto niektóre z nich:

- EXPECT-EQ(val1, val2) – sprawdza, czy val1 jest równy val2.
- EXPECT-NE(val1, val2) – sprawdza, czy val1 nie jest równy val2.

- EXPECT-TRUE(condition) / EXPECT-FALSE(condition) – sprawdzają, czy warunek jest spełniony.
- EXPECT-THROW(expression, ExceptionType) – sprawdza, czy dana funkcja rzuca oczekiwany wyjątek.

W przypadku krytycznych błędów można użyć ASSERT-*, które natychmiast przerywa wykonanie danego testu, jeśli asercja zakończy się niepowodzeniem.
Po napisaniu testów należy je skompilować razem z biblioteką Google Test. Framework generuje binarkę, która uruchamia wszystkie testy i przedstawia wyniki w przejrzystej formie. Wyniki są podsumowane w postaci:

- liczby testów, które zakończyły się sukcesem,
- liczby testów, które nie przeszły.

Zalety Google Test:

- Elastyczność i prostota: Framework wspiera testy różnych typów, od jednostkowych po testy regresji.
- Dokładne raportowanie: Google Test jasno informuje, które testy przeszły, a które zakończyły się niepowodzeniem, wskazując przy tym dokładne linie kodu.
- Obsługa wyjątków i kodu wielowątkowego: Można przetestować działanie kodu, który rzuca wyjątki lub wykonuje operacje równoległe.

Wady Google Test:

- Zależność od zewnętrznych narzędzi: Aby korzystać z Google Test, należy zintegrować go z projektem i skonfigurować środowisko.
- Brak testów funkcjonalnych i wydajnościowych: Framework skupia się głównie na testach jednostkowych.

3. Projektowanie

3.1. Wykorzystane Narzędzia

3.1.1. Visual Studio 2022

Visual Studio 2022 to zaawansowane zintegrowane środowisko programistyczne (IDE) opracowane przez firmę Microsoft. Jest to potężne narzędzie do tworzenia różnego rodzaju aplikacji, w tym aplikacji konsolowych w języku C++.

3.1.2. Kompilator C++

Kompilator C++ jest niezbędny do przekształcenia kodu źródłowego napisanego w języku C++ na kod maszynowy, który może być uruchamiany na komputerze. Visual Studio zawiera wbudowany kompilator C++, który wykonuje to zadanie.

3.1.3. Git

Git to rozproszony system kontroli wersji (VCS), który został stworzony przez Linusa Torvaldsa w 2005 roku. Git jest szeroko stosowanym narzędziem w dziedzinie programowania i zarządzania projektem, umożliwiającym śledzenie zmian w kodzie źródłowym oraz skuteczne zarządzanie projektem.

3.1.4. Język programowania C++

Język programowania, w tym przypadku jest to C++, będzie Naszym głównym narzędziem do tworzenia programu. C++ to język o dużym potencjale i jest często wykorzystywany do programowania systemów, gier komputerowych i aplikacji.

3.1.5. Google test

Google Test, z kolei, to framework do testowania jednostkowego w języku C++. W połączeniu te narzędzia pozwalaja na efektywne pisanie, debugowanie i testowanie kodu C++.

To są podstawowe narzędzia, które będziemy używać w procesie tworzenia programu w języku C++ przy użyciu środowiska Visual Studio oraz kontrolowania zmian w kodzie źródłowym za pomocą systemu kontroli wersji Git.

4. Implementacja

Algorytm Merge Sort to efektywna metoda sortowania, która operuje na zasadzie dziel i zwyciężaj. Działa poprzez rekurencyjne dzielenie zbioru danych na mniejsze części, a następnie scalanie ich w posortowaną całość. Jest to algorytm stabilny, co oznacza, że nie zmienia względnej kolejności równych elementów. Poniżej znajduje sie implementacji algorytmu Merge Sort w języku C++

```
#pragma once
#ifndef KLASA_H
#define KLASA_H


class SortowaniePrzezScalanie
{
public:
    SortowaniePrzezScalanie() {};
    ~SortowaniePrzezScalanie() {};
    void scal(int tablica[], int lewy, int prawy);
    void sortuj(int tablica[], int lewy, int prawy);
    void sortujOdwrotnie(int tablica[], int lewy, int prawy);


};
#endif // !KLASA_H
```

Listing 1. Klasa.h

4.1. Rozłożenie fragmentu kodu na cześć

4.1.1. Klasa MergeSort

- Na listingu 2 (s. 13) widzimy pragma once ktory zapobiega wielokrotnemu dołączaniu pliku nagłówkowego, co poprawia czytelność kodu i może zwiększyć wydajność kompilacji. Ifndef i define KLASA-H to standardowy idiom zabezpieczający przed wielokrotnym dołączaniem pliku nagłówkowego. Sprawdza, czy KLASA-H nie zostało zdefiniowane wcześniej w kodzie.

```
#pragma once
#ifndef KLASA_H
#define KLASA_H
```

Listing 2. Dyrektywy preprocesora

- Class SortowaniePrzezScalanie to deklaracja klasy o nazwie SortowaniePrzezScalanie, public jest to specyfikator dostępu, oznacza, że wszystkie elementy klasy są dostępne publicznie widac to na listingu 3 (s. 14).

```
class SortowaniePrzezScalanie
{
public:
    SortowaniePrzezScalanie() {};
    ~SortowaniePrzezScalanie() {};
    void scal(int tablica[], int lewy, int prawy);
    void sortuj(int tablica[], int lewy, int prawy);
    void sortujOdwrotnie(int tablica[], int lewy, int prawy);
};
```

Listing 3. Deklaracja klasy

- Na listingu 4 (s. 14)endif kończy blok obejmujący dyrektywy ifndef i define

```
#endif // !KLASA_H
```

Listing 4. Dyrektywa preprocesora zamykająca

4.1.2. Metoda scal

- Na listingu 5 (s. 14) widac include "pch.h"jest to nagłówek prekompilowany (precompiled header), który może być używany w środowisku Visual Studio. Umożliwia przyspieszenie procesu kompilacji poprzez wstępne przetwarzanie często używanych nagłówków oraz widać include "Klasa.h"jest to dołączenie pliku nagłówkowego.

```
#include "pch.h"
#include "Klasa.h"
```

Listing 5. Nagłówki

- Void SortowaniePrzezScalanie::scal(int tablica[], int lewy, int prawy): Rozpoczyna definicję funkcji scal należącej do klasy SortowaniePrzezScalanie. Parametry funkcji to tablica tablica, oraz zakres sortowania od indeksu lewy do indeksu prawy widoczne jest to na listingu 6 (s. 14)

```
void SortowaniePrzezScalanie::scal(int tablica[], int lewy, int
    prawy)
{
}
```

Listing 6. Metoda scal

- Na listingu 7 (s. 15) widac inicjalizacje zmienych. Środek oblicza indeks środka podzbioru, co jest kluczowe dla podziału tablicy na dwie części,i, j, k inicjalizacja indeksów dla dwóch podzbiorów tablicy oraz tablicy wynikowej oraz tab dynamicznie alokowana tablica pomocnicza o rozmiarze równym liczbie elementów do scalenia.

```
    int srodek = (prawy + lewy) / 2;
    int i = lewy;
    int j = srodek + 1;
    int k = 0;
    int* tab = new int[prawy - lewy + 1];
```

Listing 7. Inicjalizacja zmiennych

- Pętla while porównuje elementy dwóch podzbiorów tablicy i scalając je, umieszcza wynik w tablicy pomocniczej tab widać to na listingu 8 (s. 15).Natomiast warunek if sprawdza, który element jest mniejszy i umieszcza go w tablicy wynikowej.

```
    while (i <= srodek && j <= prawy)
{
    if (tablica[i] < tablica[j])
    {
        tab[k] = tablica[i];
        i++;
    }
    else
    {
        tab[k] = tablica[j];
        j++;
    }
    k++;
}
```

Listing 8. Porównywanie i Scalanie Elementów

- Na listingu 9 (s. 16) widać sprawdzenie, czy istnieją jeszcze elementy w jednym z podzbiorów. Jeśli tak, dodaje pozostałe elementy z lewej lub prawej strony do tablicy pomocniczej tab.

```
    if (i <= srodek)
{
    while (i <= srodek)
    {
        tab[k] = tablica[i];
        i++;
        k++;
    }
}
else
{
    while (j <= prawy)
    {
        tab[k] = tablica[j];
        j++;
        k++;
    }
}
```

Listing 9. Dodawanie Pozostałych Elementów z Lewej i Prawej Strony

- Na listingu 10 (s. 16) pętla for przenosi wyniki z tablicy pomocniczej tab z powrotem do oryginalnej tablicy tablica, aktualizując jej zawartość.

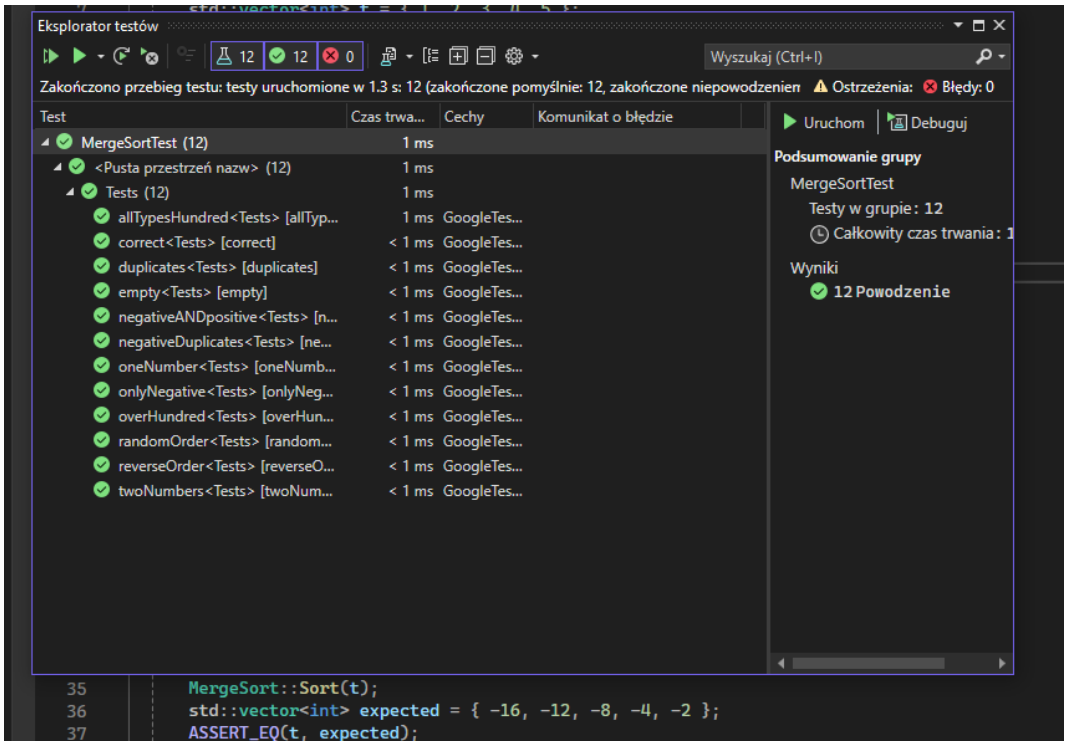
```
   for (k = 0; k <= prawy - lewy; k++)
{
    tablica[k + lewy] = tab[k];
}
```

Listing 10. Aktualizacja Tablicy Wynikowej

4.2. Wynik testow

Wszystkie 13 testów na algorytmie sortowania przez scalanie zostały przeprowadzone pomyślnie, a każdy z nich zakończył się pozytywnie.Poprawnosc testow mozemy zobaczyc na kilka sposob.

– W eksploratorze testow widoczne jest to rysuneku 4.1 (s. 17).Widzimy nazwe testu i czy zakonczyl sie powodzeniem a takze w kodzie gdzie znajduja sie testy przeprowadzane pojawi sie zielony znaczek 4.2 (s. 18).



Rys. 4.1. Eksplorator test

– Po uruchomieniu debugowania pojawia sie konsola w ktorej możemy zauważyc poprawnosc przeprowanych testow obrazuje to rysunek 4.3 (s. 18).

```
117  TEST(Tests, allTypesHundred) {
118
119      std::vector<int> t(201);
120      std::vector<int> expected(201);
121
122      for (int i = -100; i < 101; i++) {
123
124          int index = i + 100;
125          expected[index] = i * 2;
126          t[index] = i * 2;
127
128      }
129
130      t[20] = 16;
131      expected[20] = 16;
132      t[20] = 6;
133      expected[20] = 6;
134      std::sort(expected.begin(), expected.end());
135      MergeSort::Sort(t);
136      ASSERT_EQ(t, expected);
137
138  }
```

Rys. 4.2. Potwierdzenie pozytywnego testu w kodzie

```
Konsola debugowania programu Microsoft Visual Studio
Running main() from D:\a\_work\1\s\googletest\googletest\src\gtest_main.cc
[==========] Running 12 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 12 tests from Tests
[ RUN      ] Tests.correct
[       OK ] Tests.correct (0 ms)
[ RUN      ] Tests.reverseOrder
[       OK ] Tests.reverseOrder (0 ms)
[ RUN      ] Tests.randomOrder
[       OK ] Tests.randomOrder (0 ms)
[ RUN      ] Tests.onlyNegative
[       OK ] Tests.onlyNegative (0 ms)
[ RUN      ] Tests.negativeANDpositive
[       OK ] Tests.negativeANDpositive (0 ms)
[ RUN      ] Tests.empty
[       OK ] Tests.empty (0 ms)
[ RUN      ] Tests.oneNumber
[       OK ] Tests.oneNumber (0 ms)
[ RUN      ] Tests.duplicates
[       OK ] Tests.duplicates (1 ms)
[ RUN      ] Tests.negativeDuplicates
[       OK ] Tests.negativeDuplicates (0 ms)
[ RUN      ] Tests.twoNumbers
[       OK ] Tests.twoNumbers (0 ms)
[ RUN      ] Tests.overHundred
[       OK ] Tests.overHundred (1 ms)
[ RUN      ] Tests.allTypesHundred
[       OK ] Tests.allTypesHundred (0 ms)
[----------] 12 tests from Tests (5 ms total)

[----------] Global test environment tear-down
[==========] 12 tests from 1 test case ran. (5 ms total)
[  PASSED  ] 12 tests.

C:\Users\Adrian\Desktop\MergeSort\MergeSortTest\x64\Debug\MergeSortTest.exe (proces 49928) zakończono z kodem 0 (0x0).
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Rys. 4.3. Konsola-test

5. Wnioski

W trakcie projektu potwierdzono skuteczność algorytmu sortowania przez scalanie poprzez przeprowadzenie 13 testów za pomocą frameworka Google Test. Testy obejmowały różne scenariusze, takie jak tablice posortowane rosnąco, malejąco, puste tablice, jednoelementowe.Wyniki testów potwierdziły poprawność działania algorytmu dla różnorodnych przypadków. Algorytm zachował stabilność i skuteczność nawet dla specyficznych przypadków, takich jak obecność elementów powtarzających się, liczby ujemne czy zera . Framework Google Test okazał się skutecznym narzędziem do przeprowadzania testów jednostkowych. Jego struktura ułatwiła pisanie, uruchamianie i analizowanie testów, co przyczyniło się do skutecznego sprawdzenia poprawności implementacji algorytmu.

Bibliografia

[1] *Strona internetowa khanacadem.* URL: https://pl.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/overview-of-merge-sort.

[2] *Strona internetowa Encyklopedia Algorytmów.* URL: http://algorytmy.ency.pl/artykul/sortowanie_przez_scalanie.

Spis rysunków

Spis listingów