AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm sortowania przez scalanie (merge sort)

Autor:
Mateusz Basiaga

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Celem projektu jest implementacja i przetestowanie algorytmu sortowania przez scalanie (Merge Sort) w języku C++ w formie modułowego projektu. Projekt obejmuje nie tylko implementację samego algorytmu, ale również stworzenie dodatkowych funkcjonalności wspomagających obsługę tablic, a także opracowanie dokumentacji i procedur testowania.

1.1. Cel pracy

Głównym celem projektu jest:

- Implementacja algorytmu sortowania przez scalanie w sposób modularny, z podziałem na klasy i pliki źródłowe.
- Zastosowanie dobrej praktyki programowania poprzez oddzielenie logiki aplikacji od interfejsu użytkownika.
- Przeprowadzenie testów jednostkowych z wykorzystaniem frameworka Google Test w celu weryfikacji poprawności działania algorytmu w różnych scenariuszach.
- Stworzenie interaktywnej aplikacji umożliwiającej użytkownikowi manipulowanie tablicami i testowanie działania algorytmu w czasie rzeczywistym.
- Opracowanie szczegółowej dokumentacji projektu za pomocą narzędzi LaTeX i Doxygen.

1.2. Zakładane wyniki

Oczekiwane efekty projektu to:

- Sprawna i poprawna implementacja algorytmu Merge Sort, który będzie w stanie skutecznie sortować zarówno małe, jak i bardzo duże tablice.
- Zbiór testów jednostkowych pokrywających kluczowe przypadki użycia algorytmu, w tym tablice losowe, posortowane, odwrócone, zawierające liczby ujemne, duplikaty oraz przypadki graniczne, takie jak tablice puste czy jednoelementowe.

- Dokumentacja użytkownika i programisty opisująca zarówno sposób korzystania z aplikacji, jak i techniczne aspekty implementacji oraz założenia projektowe.
- Automatyzacja procesu testowania i budowania projektu dzięki zastosowaniu CMake, Ninja oraz GitHub Actions.
- Wynikowy projekt, który może być uruchamiany na różnych systemach operacyjnych, takich jak Windows (z wykorzystaniem MINGW-w64) oraz Linux.

1.3. Określenie ram projektowych

Projekt realizowany jest zgodnie z poniższymi wytycznymi:

- Algorytm Merge Sort zostanie zaimplementowany jako osobna klasa `MergeSorter`, której funkcjonalności będą odpowiednio odseparowane od pozostałych modułów aplikacji.
- Całość zostanie zaimplementowana z wykorzystaniem wzorców dobrych praktyk programistycznych, takich jak modularność, hermetyzacja kodu oraz dokumentowanie przy pomocy stylu Doxygen.
- Testy jednostkowe pokrywające wszystkie krytyczne aspekty działania algorytmu zostaną umieszczone w pliku `merge_sorter_test.cpp`.
- Dokumentacja będzie tworzona z użyciem szablonów i makr `LaTeX`, aby zapewnić spójność formatowania i estetyczny wygląd publikacji.

W dalszych rozdziałach omówiono szczegółowo implementację poszczególnych modułów projektu oraz uzyskane wyniki testów.

2. Analiza problemu

2.1. Zastosowanie algorytmu sortowania przez scalanie

Algorytm sortowania przez scalanie (Merge Sort) jest jednym z najczęściej wykorzystywanych algorytmów do sortowania dużych zbiorów danych w informatyce. Jego główną zaletą jest stabilność i efektywność w przypadku dużych zbiorów danych. Jest to algorytm typu „dziel i zwyciężaj", który dzieli zbiór na mniejsze podzbiory, a następnie scala je w sposób uporządkowany. Algorytm ten jest wykorzystywany w różnych dziedzinach, takich jak:

- Sortowanie dużych zbiorów danych w bazach danych.
- Przetwarzanie danych w systemach rozproszonych, gdzie duże ilości danych muszą być uporządkowane przed dalszym przetwarzaniem.
- Algorytmy związane z analizą danych, takie jak sortowanie wyników wyszukiwania.
- Problemy związane z inżynierią oprogramowania, w których dane muszą być uporządkowane w sposób stabilny (np. przy sortowaniu użytkowników po wielu kryteriach).

W niniejszym projekcie zaimplementowano algorytm Merge Sort, który jest wykorzystywany do sortowania tablic liczb całkowitych. Program implementuje ten algorytm w języku C++, zapewniając jego elastyczność i możliwość testowania w różnych scenariuszach.

2.2. Opis algorytmu sortowania przez scalanie

Algorytm Merge Sort działa na zasadzie podziału problemu na mniejsze podproblemy, co pozwala na efektywne sortowanie danych. Ogólny przebieg działania algorytmu przedstawia się następująco:

1. Jeśli tablica ma więcej niż jeden element, dzielimy ją na dwie części (połowy).
2. Rekursywnie sortujemy każdą z części.
3. Po posortowaniu każdej z części, łączymy (scalamy) je w sposób uporządkowany w jedną, posortowaną tablicę.

Przykład: Rozważmy tablicę liczb `[38, 27, 43, 3, 9, 82, 10]`. Algorytm Merge Sort wykonuje następujące kroki:

1. Dzieli tablicę na dwie części: `[38, 27, 43]` i `[3, 9, 82, 10]`.
2. Każdą z części dzieli dalej, aż do uzyskania tablic jednowymiarowych:
   - `[38, 27, 43]` → `[38, 27]` i `[43]` → `[38]` i `[27]` → `[38]` i `[27]`.
   - `[3, 9, 82, 10]` → `[3, 9]` i `[82, 10]` → `[3]` i `[9]` → `[3]` i `[9]` → `[82]` i `[10]`.
3. Następnie łączymy elementy w sposób uporządkowany:
   - `[38]` i `[27]` → `[27, 38]`.
   - `[38, 27]` i `[43]` → `[27, 38, 43]`.
   - `[3]` i `[9]` → `[3, 9]`.
   - `[82]` i `[10]` → `[10, 82]`.
   - `[3, 9]` i `[10, 82]` → `[3, 9, 10, 82]`.
4. Na koniec scalamy dwie posortowane tablice:
   - `[27, 38, 43]` i `[3, 9, 10, 82]` → `[3, 9, 10, 27, 38, 43, 82]`.

Ostateczny wynik to posortowana tablica: `[3, 9, 10, 27, 38, 43, 82]`.

2.3. Sposób wykorzystania algorytmu w projekcie

W projekcie algorytm Merge Sort został zaimplementowany jako część klasy `MergeSorter`, której zadaniem jest sortowanie tablicy liczb całkowitych. Użytkownik może interaktywnie dodawać, usuwać lub tasować elementy w tablicy, a następnie uruchomić algorytm, aby posortować tablicę. Proces ten jest uproszczony do kilku kroków, które opisano poniżej.

- Po uruchomieniu aplikacji użytkownik ma dostęp do menu, w którym może wybrać jedną z dostępnych operacji na tablicach.
- Wybór opcji "Sortowanie"uruchamia algorytm Merge Sort, który wykonuje operację sortowania na aktualnej tablicy.
- Użytkownik może także zainicjować generowanie losowej tablicy lub odwrócić kolejność elementów w tablicy, aby przeprowadzić sortowanie w różnych scenariuszach.

- Po zakończeniu sortowania, wynikowa tablica zostaje wyświetlona na ekranie.

Przykład: 1. Użytkownik generuje losową tablicę: `[38, 27, 43, 3, 9, 82, 10]`. 2. Uruchamia sortowanie przez scalanie. 3. Aplikacja wyświetla posortowaną tablicę: `[3, 9, 10, 27, 38, 43, 82]`.

2.4. Opis narzędzi i technologii użytych w projekcie

W projekcie zastosowano szereg narzędzi i technologii, które wspierają rozwój, testowanie oraz dokumentowanie aplikacji. Do głównych narzędzi należą:

- **C++**: Język programowania użyty do implementacji algorytmu oraz aplikacji. C++ zapewnia wysoką wydajność, co jest istotne przy pracy z dużymi zbiorami danych.
- **Google Test**: Framework do testowania jednostkowego w C++. Google Test umożliwia automatyczne uruchamianie testów oraz weryfikację poprawności działania algorytmu w różnych scenariuszach.
- **CMake**: System budowania, który umożliwia generowanie plików konfiguracyjnych dla różnych środowisk kompilacji. CMake zapewnia elastyczność w budowaniu projektu na różnych systemach operacyjnych.
- **Doxygen**: Narzędzie do automatycznego generowania dokumentacji z komentarzy w kodzie. Doxygen pozwala na tworzenie dokumentacji technicznej w formacie HTML lub PDF.
- **GitHub Actions**: Narzędzie do automatyzacji procesów CI/CD. GitHub Actions służy do automatycznego uruchamiania testów, generowania dokumentacji i wdrażania aplikacji.

3. Projektowanie

3.1. Użyte narzędzia i technologie

W projekcie zastosowane zostały następujące narzędzia i technologie, które wspierają zarówno rozwój aplikacji, jak i zapewniają jej odpowiednią jakość oraz funkcjonalność:

- **Język programowania C++**: Został wybrany ze względu na wysoką wydajność oraz wsparcie dla programowania obiektowego, co umożliwia elegancką implementację algorytmu sortowania przez scalanie oraz operacji na tablicach. C++ jest także powszechnie stosowany w projektach wymagających optymalizacji pamięci i szybkości działania.

- **Google Test**: Framework do testowania jednostkowego, który pozwala na automatyczne sprawdzanie poprawności algorytmu w różnych scenariuszach. Google Test ułatwia również integrację z systemem Continuous Integration (CI), dzięki czemu testy są uruchamiane automatycznie na każdym etapie rozwoju projektu.

- **CMake**: Używany do konfigurowania procesu budowy projektu. CMake generuje odpowiednie pliki dla różnych systemów operacyjnych i kompilatorów, co zapewnia elastyczność przy kompilacji na różnych platformach (Windows, Linux).

- **Ninja**: Lekki system budowania, który jest wykorzystywany razem z CMake do zapewnienia szybciej działających kompilacji. Ninja jest wybierany, ponieważ jest mniej zasobożerny niż tradycyjne systemy budowania.

- **Doxygen**: Narzędzie służące do generowania dokumentacji z komentarzy w kodzie. Doxygen automatycznie tworzy dokumentację w formacie HTML, która jest następnie udostępniana na stronie projektu. Użycie Doxygen pozwala na przejrzystość kodu oraz ułatwia zrozumienie jego działania.

- **GitHub Actions**: Platforma CI/CD umożliwiająca automatyczne uruchamianie testów, generowanie dokumentacji oraz wdrażanie aplikacji na repozytorium GitHub. GitHub Actions jest wykorzystywany do uruchamiania testów jednostkowych, weryfikacji konwencji commitów, generowania dokumentacji oraz publikacji wersji na GitHub Pages.

3.2. Szczegółowe ustawienia kompilatora

Projekt jest konfigurowany przy użyciu systemu budowania CMake, który generuje odpowiednie pliki konfiguracyjne w zależności od wybranego środowiska. Poniżej przedstawiono szczegóły ustawień kompilatora:

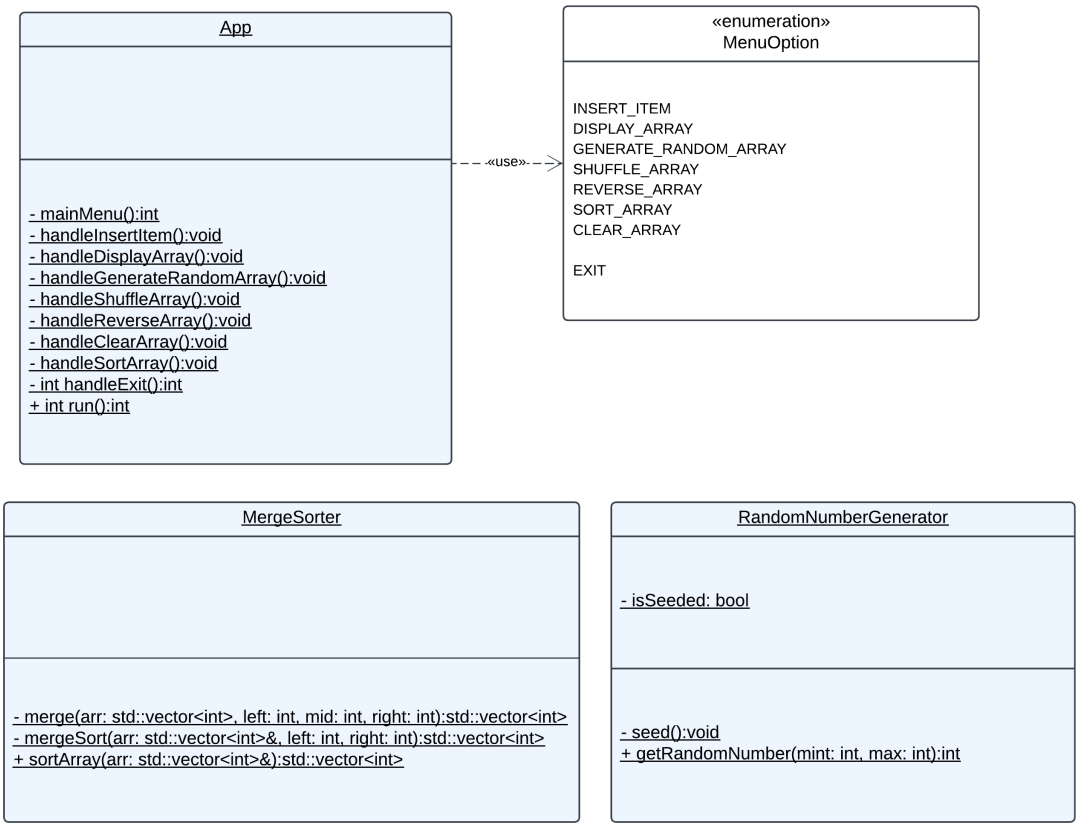
- **Kompilacja dla systemu Linux**: Projekt jest kompilowany za pomocą kompilatora `g++` (GCC) dostępnego na systemach Linux. Ustawienia CMake są zoptymalizowane pod kątem kompilatora GCC, zapewniając optymalną wydajność podczas kompilacji.
- **Kompilacja dla systemu Windows**: Na systemie Windows projekt jest kompilowany przy użyciu `MINGW-w64`, który pozwala na kompilację kodu C++ w środowisku zgodnym z GNU (GCC) w systemie Windows. Kompilator ten jest wykorzystywany w połączeniu z MSYS2.
- **Ustawienia CMake**: W projekcie zastosowano plik `CMakeLists.txt`, który definiuje, jakie biblioteki oraz pliki źródłowe są wykorzystywane w projekcie. Plik konfiguracyjny ustawia również wymagania dotyczące systemu budowania oraz ścieżki do zależności.
- **System Ninja**: W połączeniu z CMake, projekt wykorzystuje system budowania `Ninja`, który jest lekki i szybki. Dzięki temu proces kompilacji jest szybszy w porównaniu do tradycyjnych systemów, takich jak `Make`.

3.3. Struktura kodu i diagram klas

Aby ułatwić zrozumienie struktury aplikacji, na listingu 3.1 (s. 11) przedstawiono diagram klas oraz opis poszczególnych komponentów aplikacji.

Opis diagramu klas:

- **App**: Główna klasa aplikacji odpowiedzialna za interakcję z użytkownikiem. Umożliwia wybór różnych operacji na tablicach, takich jak dodawanie, usuwanie, tasowanie i sortowanie.
- **MergeSorter**: Klasa implementująca algorytm sortowania przez scalanie. Zawiera metody odpowiedzialne za dzielenie tablicy, sortowanie oraz scalanie wyników.
- **ArrayUtils**: Biblioteka zawierająca pomocnicze funkcje do manipulacji tablicami, takie jak tasowanie (algorytm Fishera-Yatesa), odwracanie kolejności czy wyświetlanie zawartości tablicy.



Rys. 3.1. Diagram klas aplikacji MergeSort

- **RandomNumberGenerator**: Klasa generująca losowe liczby całkowite w określonym zakresie. Jest wykorzystywana do generowania danych testowych oraz do inicjalizowania tablic w różnych scenariuszach.

3.4. Schemat działania algorytmu

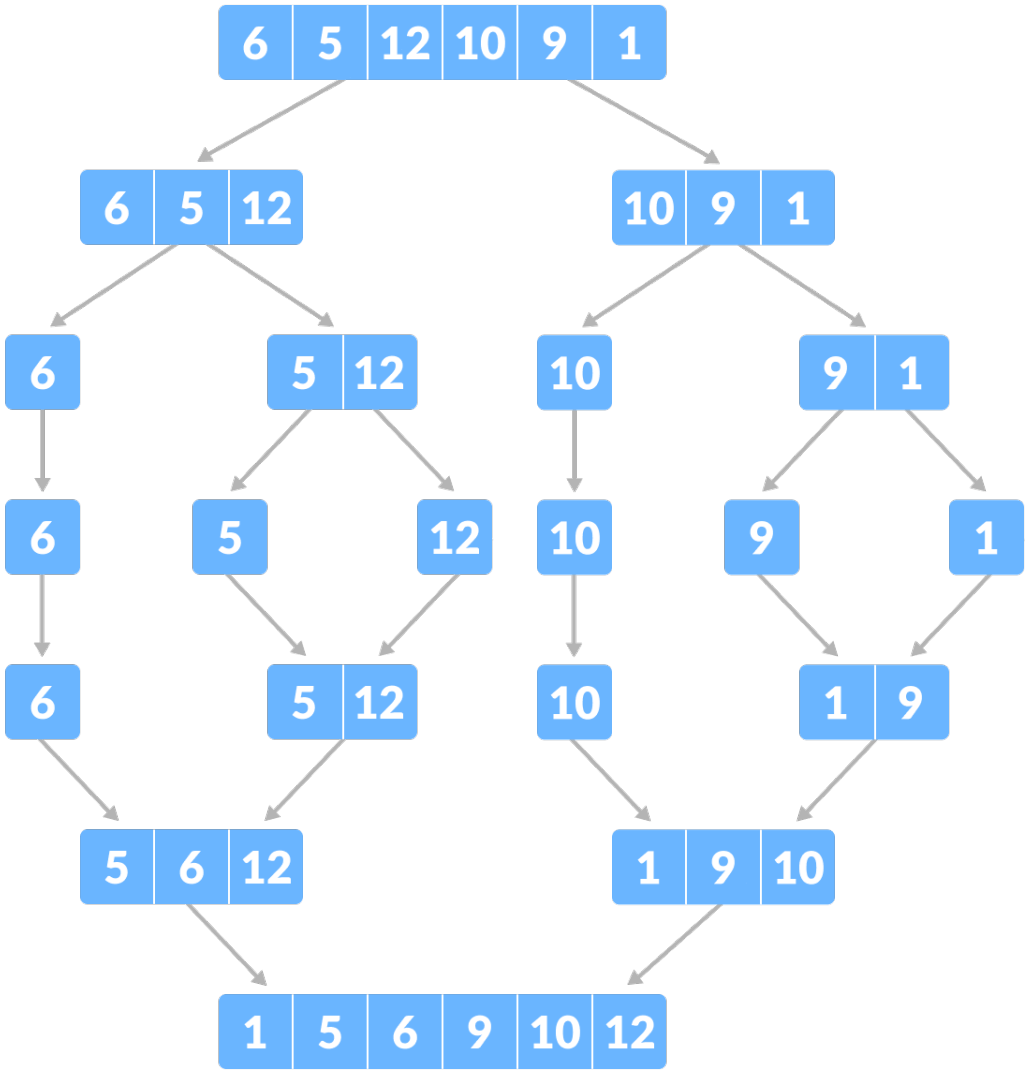
Algorytm Merge Sort działa na zasadzie podziału i scalania, co jest kluczowe do zrozumienia jego działania. Na rysunku 3.2 (s. 12) przedstawiono schemat działania algorytmu.

Opis schematu algorytmu:

1. Początkowo tablica jest dzielona na dwie części, aż do uzyskania pojedynczych elementów.

2. Następnie elementy są scalane w sposób uporządkowany. Scalenie polega na porównaniu pierwszego elementu z obu tablic i wstawieniu mniejszego z nich

[1] Źródło: https://www.programiz.com/dsa/merge-sort



Rys. 3.2. Schemat działania algorytmu Merge Sort [1]

do nowej tablicy.

3. Proces ten jest powtarzany, aż wszystkie elementy zostaną scalone w jednej, uporządkowanej tablicy.

3.5. Zarządzanie wersjami i użycie GIT

Do zarządzania wersjami kodu wykorzystano system kontroli wersji Git. Projekt jest hostowany na platformie GitHub, gdzie korzysta się z następujących funkcji:

- **Branching**: Rozwój projektu odbywa się na osobnych gałęziach, co pozwala na wprowadzenie nowych funkcji lub poprawek bez wpływu na główną wersję

projektu (gałąź `main`).

- **Pull Requests**: Zmiany wprowadzane na gałęziach są łączone z główną gałęzią poprzez pull requesty, co umożliwia przeglądanie kodu oraz automatyczne uruchamianie testów przed połączeniem zmian.
- **Commit Hooks**: Zastosowanie `Git Hooks` umożliwia automatyczne uruchamianie testów jednostkowych przed każdą zmianą w repozytorium, co zapewnia, że tylko poprawny kod zostanie zaakceptowany.
- **GitHub Actions**: Zautomatyzowane procesy CI/CD pozwalają na uruchamianie testów, generowanie dokumentacji oraz publikację wyników na GitHub Pages.

3.6. Testy jednostkowe

Testy jednostkowe są kluczowym elementem procesu weryfikacji poprawności działania algorytmu Merge Sort. Celem testów jest zapewnienie, że implementacja algorytmu działa zgodnie z oczekiwaniami w różnych scenariuszach, a także wykrywanie potencjalnych błędów w kodzie. W projekcie zastosowano framework `Google Test`, który pozwala na pisanie i uruchamianie testów w sposób automatyczny. Poniżej przedstawiono szczegóły dotyczące testów jednostkowych.

3.6.1. Przykłady testów jednostkowych

Testy jednostkowe w projekcie obejmują szeroki zakres przypadków testowych, mających na celu sprawdzenie poprawności algorytmu Merge Sort w różnych warunkach. Poniżej przedstawiono kilka przykładów testów:

- **Testowanie tablicy już posortowanej**: Sprawdzany jest przypadek, gdy algorytm otrzymuje już posortowaną tablicę. Oczekiwany wynik to ta sama tablica, ponieważ algorytm nie powinien wprowadzać żadnych zmian w takim przypadku.
- **Testowanie tablicy w odwrotnej kolejności**: Testuje się przypadek, gdy tablica jest posortowana w odwrotnej kolejności. Merge Sort powinien prawidłowo posortować tę tablicę w porządku rosnącym.
- **Testowanie tablicy z duplikatami**: Sprawdzany jest przypadek, w którym tablica zawiera elementy powtarzające się. Algorytm powinien poprawnie poradzić sobie z duplikatami, zachowując ich obecność w posortowanej tablicy.

- **Testowanie tablicy z liczbami ujemnymi**: Algorytm jest testowany w sytuacji, gdy tablica zawiera liczby ujemne. Oczekiwany wynik to posortowanie tych liczb w porządku rosnącym.
- **Testowanie dużych tablic**: Sprawdzany jest przypadek, gdy tablica zawiera dużą liczbę elementów (np. 1000 lub więcej). Algorytm powinien działać wydajnie, bez przekroczenia limitów czasowych i pamięciowych.

3.6.2. Struktura testów jednostkowych

Wszystkie testy jednostkowe zostały zorganizowane w oddzielnym pliku `merge_sorter_test.cpp`. Każdy test sprawdza określoną funkcjonalność algorytmu, a wyniki testów są raportowane za pomocą frameworka `Google Test`, który automatycznie generuje raporty o wynikach każdego z testów.

Przykład testu jednostkowego w projekcie:

```
1 TEST(TestMergeSorter, KeepsAlreadySortedArrayIntact) {
2   std::vector<int> test_array = { 1, 2, 3, 4, 5, 7, 8, 9 };
3   MergeSorter::sortArray(test_array);
4
5   ASSERT_THAT(test_array, testing::ElementsAre(1, 2, 3, 4, 5, 7, 8,
      9));
6 }
```

Listing 1. Przykład testu jednostkowego

W przykładzie pokazanym na listingu 1 (s. 15) testowana jest tablica, która już jest posortowana. Algorytm Merge Sort powinien zwrócić tę samą tablicę. Funkcja `EXPECT_EQ` sprawdza, czy wynik sortowania jest zgodny z oczekiwanym.

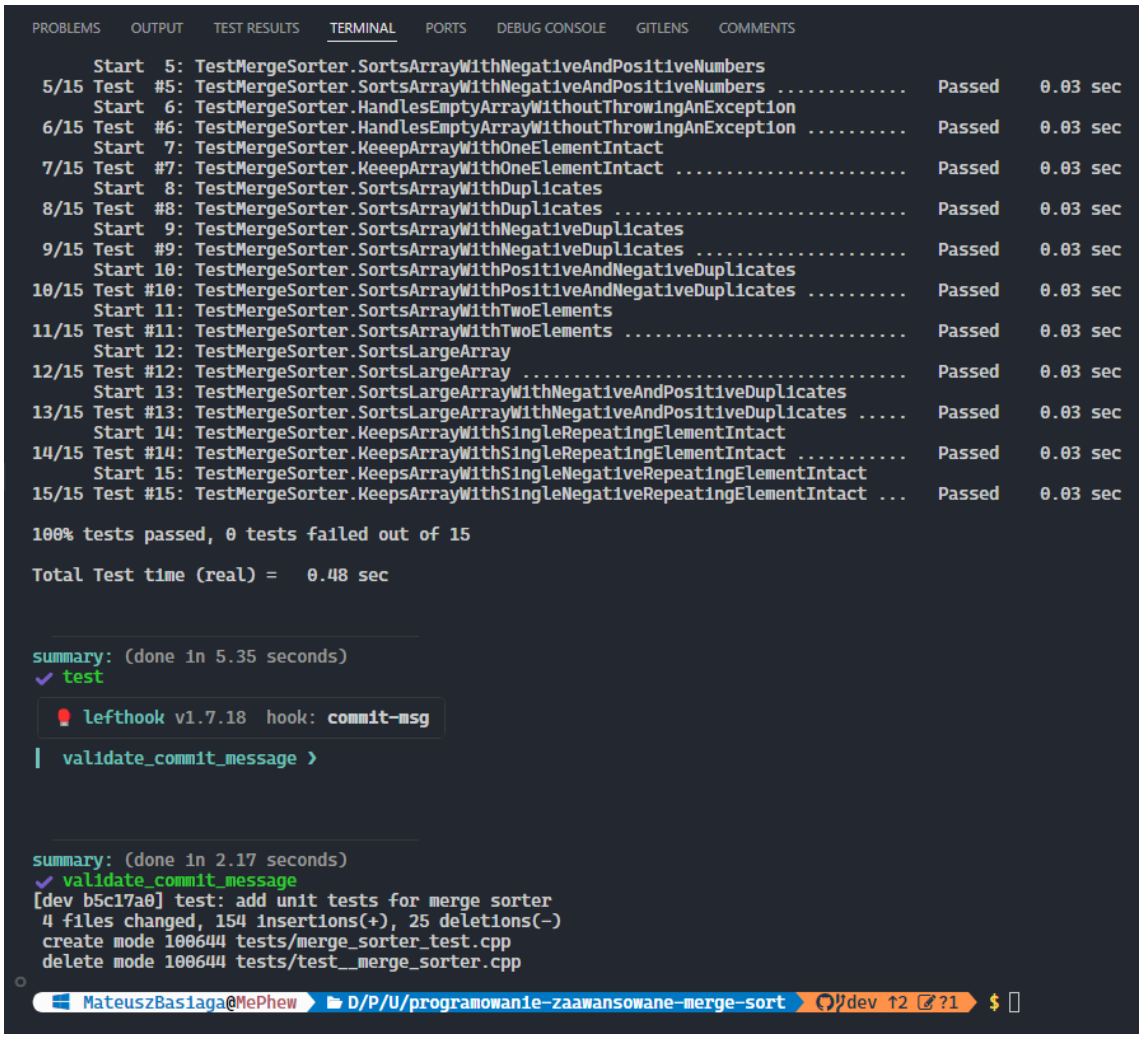
3.6.3. Integracja testów z systemem CI/CD

Testy jednostkowe są integralną częścią procesu Continuous Integration (CI), co oznacza, że są one automatycznie uruchamiane za każdym razem, gdy kod jest zmieniany i wysyłany do repozytorium. Dzięki integracji z `GitHub Actions`, każda zmiana w kodzie uruchamia zestaw testów jednostkowych, weryfikując poprawność nowego kodu przed jego połączeniem z główną gałęzią projektu.

- **Automatyczne uruchamianie testów**: Testy są uruchamiane automatycznie na każdej gałęzi projektu, a wyniki są raportowane na platformie GitHub. Dzięki temu deweloperzy mogą natychmiast zobaczyć, czy ich zmiany wprowadzają jakiekolwiek błędy.

  Na rysuku 3.3 (s. 16) pokazano przebieg testów uruchomiony automatycznie po wykonaniu commita. Aby commmit został wykonany wynik testów musi być pozytywny.

- **Przegląd wyników testów**: Każdy commit generuje raport o wynikach testów, który jest dostępny na stronie z historią commitów. Umożliwia to szybkie wykrycie błędów w kodzie oraz ich naprawienie.



Rys. 3.3. Przykładowy przebieg testów po wykonaniu commita

- **Zabezpieczenie przed błędami**: Każda próba połączenia kodu z gałęzią `main` wymaga, aby testy jednostkowe zakończyły się sukcesem. Dzięki temu, kod w głównej gałęzi projektu zawsze jest sprawdzony i działa zgodnie z oczekiwaniami.

4. Implementacja

W tej sekcji opisano implementację algorytmu Merge Sort, który jest głównym elementem projektu. Algorytm ten jest zaimplementowany w języku C++ i stanowi część aplikacji, która umożliwia operowanie na tablicach liczb całkowitych. Omówimy tu poszczególne komponenty projektu, przedstawiając kluczowe fragmenty kodu i wyniki uzyskane po jego uruchomieniu.

4.1. Główna struktura programu

Program został zaprojektowany z wykorzystaniem podejścia obiektowego. Główne klasy w projekcie to:

- **MergeSorter** - zawiera implementację samego algorytmu Merge Sort.
- **App** - odpowiedzialna za interakcję z użytkownikiem, umożliwiająca dodawanie, sortowanie i manipulowanie tablicami.
- **RandomNumberGenerator** - generuje losowe liczby całkowite w zadanym zakresie.
- **ArrayUtils** - zawiera pomocnicze funkcje operujące na tablicach, takie jak ich wyświetlanie, tasowanie, odwracanie itp.

Każda z tych klas została zaprojektowana zgodnie z zasadami hermetyzacji i modułowości, co pozwoliło na zachowanie przejrzystości kodu i łatwości w jego modyfikacji.

4.2. Algorytm Merge Sort

Implementacja algorytmu Merge Sort w klasie `MergeSorter` bazuje na klasycznej wersji algorytmu rekurencyjnego. Kluczowe metody tej klasy to:

- `merge()` - funkcja odpowiedzialna za scalanie dwóch posortowanych tablic w jedną.
- `mergeSort()` - metoda rekurencyjna, która dzieli tablicę na mniejsze części, sortuje je, a następnie scala.
- `sortArray()` - publiczna metoda interfejsu, która przygotowuje tablicę do posortowania, wywołując funkcje sortujące.

4.2.1. Fragment kodu - Merge Sort

Na listingu 2 (s. 18) przedstawiono kluczowy fragment kodu odpowiedzialny za implementację algorytmu Merge Sort:

```
1  void MergeSorter::mergeSort(std::vector<int>& arr, int left, int
       right) {
2      if (left < right) {
3          int mid = left + (right - left) / 2;
4
5          mergeSort(arr, left, mid);
6          mergeSort(arr, mid + 1, right);
7
8          merge(arr, left, mid, right);
9      }
10 }
```

Listing 2. Funkcja mergeSort

`merge()` jest funkcją scalającą dwie posortowane tablice w jedną posortowaną. `mergeSort()` to rekurencyjna funkcja, która dzieli tablicę na dwie części, wywołuje je rekurencyjnie oraz wywołuje funkcję scalania `merge()`.

4.3. Wyniki

Po zaimplementowaniu algorytmu Merge Sort, program został uruchomiony na różnych zestawach danych wejściowych. Oczekiwanym wynikiem jest prawidłowo posortowana tablica, niezależnie od początkowego układu elementów. Testy przeprowadzone na różnych tablicach, w tym na tablicach już posortowanych, w odwrotnej kolejności oraz zawierających duplikaty, potwierdziły, że algorytm działa zgodnie z oczekiwaniami.

4.3.1. Fragment kodu - Test jednostkowy (losowa tablica)

Test jednostkowy weryfikuje poprawność działania algorytmu na dużej tablicy losowych liczb. Przy użyciu generatora pseudolosowego tworzona jest tablica, która następnie podlega sortowaniu. Test upewnia się, że po sortowaniu każda kolejna liczba w tablicy jest mniejsza bądź równa poprzedniej.

```
TEST(TestMergeSorter,
    SortsLargeArrayWithNegativeAndPositiveDuplicates) {
  std::vector<int> test_array;

  for (int i = 0; i < 500; ++i) {
    test_array.push_back(RandomNumberGenerator::getRandomNumber
    (-99, 99));
  }

  MergeSorter::sortArray(test_array);

  for (std::vector<int>::size_type i = 0; i < test_array.size() -
    1; ++i) {
    ASSERT_LE(test_array[i], test_array[i + 1]);
  }
}
```

Listing 3. Test algorytmu na losowej tablicy

3 (s. 19) przedstawia szczegóły implementacji jednego z kluczowych testów jednostkowych. Korzysta on z klasy `RandomNumberGenerator` do tworzenia losowych danych wejściowych oraz klasy `MergeSorter` do sortowania. Dzięki temu możemy zweryfikować, że algorytm działa zgodnie z oczekiwaniami w przypadku dużej i zróżnicowanej tablicy.

Test został uruchomiony w środowisku testowym Google Test, zapewniając wiarygodność i łatwość integracji w procesie ciągłej integracji.

5. Wnioski

W ramach przeprowadzonego projektu udało się zrealizować pełną implementację algorytmu sortowania przez scalanie (Merge Sort), który został zastosowany do sortowania tablic liczb całkowitych. Aplikacja została zaprojektowana zgodnie z zasadami obiektowości, co pozwoliło na zachowanie elastyczności i łatwości w rozbudowie oraz utrzymaniu kodu. Wszystkie operacje na tablicach, takie jak dodawanie elementów, sortowanie, tasowanie czy odwracanie, zostały zaimplementowane w sposób modularny, co ułatwia późniejsze modyfikacje oraz rozbudowę aplikacji.

Najważniejszym elementem tego projektu, poza samą implementacją algorytmu, było zastosowanie testów jednostkowych. Testy te stanowiły kluczowy element procesu weryfikacji poprawności działania algorytmu oraz innych funkcji aplikacji. Dzięki testom jednostkowym udało się upewnić, że algorytm działa poprawnie w różnych przypadkach granicznych, takich jak tablice puste, posortowane, odwrotnie posortowane czy zawierające duplikaty. Testy te są niezbędnym narzędziem w procesie zapewniania jakości oprogramowania, ponieważ pozwalają na szybkie wykrycie błędów oraz weryfikację, że zmiany wprowadzone w kodzie nie wprowadzają nowych problemów.

5.1. Zastosowanie testów jednostkowych

Testy jednostkowe stanowiły fundament procesu rozwoju projektu. Dzięki zastosowaniu frameworka Google Test, możliwe było automatyczne testowanie każdego z komponentów aplikacji, co zapewniło, że każda funkcjonalność działa zgodnie z oczekiwaniami. Przeprowadzone testy jednostkowe obejmowały zarówno proste przypadki, jak i bardziej złożone, takie jak testowanie dużych tablic oraz tablic z duplikatami. Użycie testów jednostkowych pozwoliło także na szybkie wykrycie problemów, które mogłyby pojawić się w przypadku modyfikacji kodu, zapewniając tym samym większą stabilność i niezawodność aplikacji.

5.2. Podsumowanie wyników

Algorytm Merge Sort, implementowany w ramach tego projektu, działa poprawnie na różnych zestawach danych wejściowych. Został on skutecznie zaimplementowany i przetestowany, a jego działanie zostało potwierdzone przez pozytywne wyniki testów jednostkowych. Testy wykazały, że algorytm działa zgodnie z oczekiwaniami, niezależnie od układu początkowego tablicy. Algorytm efektywnie radzi sobie z tablicami o dużych rozmiarach oraz tymi zawierającymi duplikaty, co potwierdza jego poprawność.

5.3. Wnioski końcowe

Projekt ten pokazał, jak ważne jest stosowanie testów jednostkowych w procesie programowania. Dzięki testom jednostkowym, cały projekt zyskał większą stabilność, a błędy zostały szybko zidentyfikowane i naprawione. Przeprowadzenie testów na różnych zestawach danych umożliwiło wyciągnięcie wniosków na temat działania algorytmu w różnych scenariuszach, a także zapewniło, że algorytm spełnia wszystkie założenia funkcjonalne.

Z punktu widzenia implementacji, algorytm Merge Sort sprawdził się jako skuteczne narzędzie do sortowania tablic. Jego złożoność czasowa $\mathcal{O}(n \log n)$ okazała się odpowiednia do sortowania dużych zbiorów danych, co potwierdziły testy na tablicach o dużej liczbie elementów. Dodatkowo, zastosowanie podejścia rekurencyjnego w połączeniu z metodą scalania zapewnia, że algorytm działa w sposób wydajny, a jego implementacja jest stosunkowo prosta do zrozumienia i utrzymania.

Spis rysunków

Spis tabel

Spis listingów