AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm sortowania przez scalanie z zastosowaniem Testów Google

Autor:
Grzegorz Fiejtek

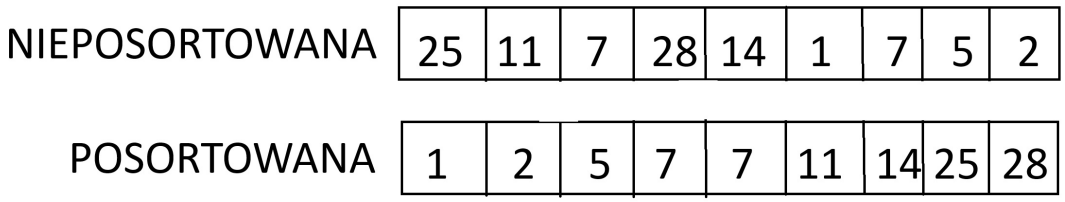
Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Projekt polega na stworzeniu programu polegającego na sortowaniu liczb metodą: "Sortowanie przez scalanie". Sortowanie to polega na rozdzieleniu liczb na 1-elementowe tablice a następnie w zależności od indexu łączenie i jednoczesne sortowanie elementów aż złączą się wszystkie. Widok tablicy nieposortowanej i posortowanej w 1.1



Rys. 1.1. Posortowana i nie posortowana tablica

Kolejną częścią projektu jest wykorzystanie testów Google do sprawdzenia czy program działa poprawnie, za pomocą tego narzędzia jesteśmy w stanie łatwo sprawdzić czy program działa czy ma jakieś problemy.

2. Analiza problemu

2.1. Co to algorytm sortowania przez scalanie

Sortowanie przez scalanie to algorytm sortujący, który należy do rodziny algorytmów opartych na podejściu "dziel i zwyciężaj". Działa w sposób rekurencyjny i efektywnie sortuje dane, dzieląc je na coraz mniejsze fragmenty, a następnie łącząc je w posortowaną całość.

2.2. Zastosowanie

Sortowanie przez scalanie jest szeroko stosowane w różnych dziedzinach, takich jak:

-Sortowanie dużych zbiorów danych: Algorytm jest bardzo efektywny, szczególnie w przypadku pracy z dużymi zbiorami danych, gdzie inne algorytmy, takie jak sortowanie bąbelkowe czy przez wstawianie, mogą być zbyt wolne.

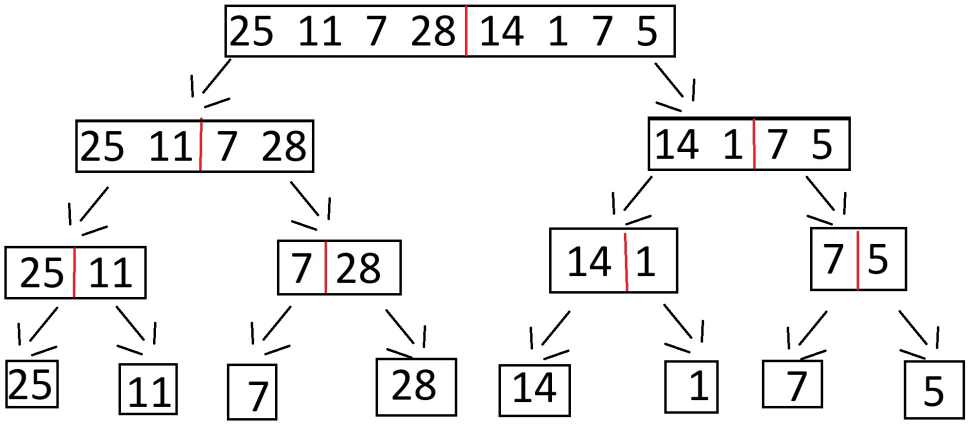
-Stabilność sortowania: Sortowanie przez scalanie jest algorytmem stabilnym, co oznacza, że zachowuje kolejność elementów o tej samej wartości. To jest ważne w przypadkach, gdy istotna jest zachowanie oryginalnej kolejności danych o tych samych wartościach.

-Sortowanie zewnętrzne: W przypadku bardzo dużych zbiorów danych, które nie mieszczą się w pamięci operacyjnej (RAM), sortowanie przez scalanie może być zastosowane w wersji zewnętrznej, wykorzystując dysk twardy do przechowywania tymczasowych fragmentów danych.

2.3. Sposób działania

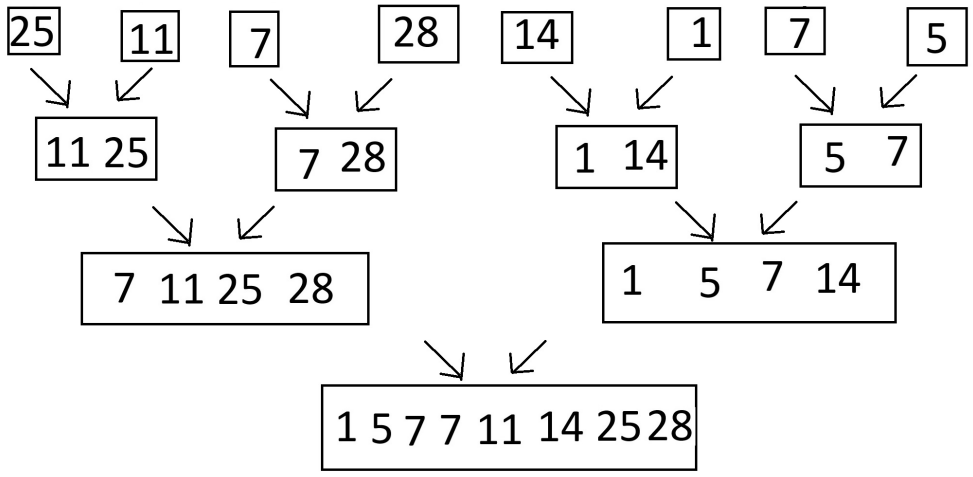
Algorytm sortowania przez scalanie działa w dwóch głównych etapach: dzieleniu i scalaniu. Oto szczegółowy opis działania:

-Dziel: Na początku tablica danych jest dzielona na dwie mniejsze części. Proces dzielenia powtarza się rekurencyjnie, aż każda część będzie zawierała tylko jeden element, ponieważ pojedynczy element jest już posortowany jak pokazano na rysunku 2.1.



Rys. 2.1. ozdzielenie tablicy

-Scalaj: Następnie rozpoczyna się proces łączenia mniejszych części w większe, przy czym każda z tych części jest już posortowana. Podczas scalania łączymy dwie posortowane tablice w jedną większą tablicę w sposób taki, że elementy są dodawane w odpowiedniej kolejności jak pokazano na rysunku 2.2.



Rys. 2.2. Scalenie tablicy

3. Projektowanie

Google tests

3.0.1. Co to gtest

Google Test to framework, który umożliwia tworzenie i uruchamianie testów jednostkowych, czyli testów, które sprawdzają poprawność pojedynczych fragmentów kodu (najczęściej funkcji lub metod). Dzięki Google Test można automatycznie testować, czy kod zachowuje się zgodnie z oczekiwaniami, co pomaga w wykrywaniu błędów i regresji w aplikacjach.

3.0.2. zastosowanie

Google Test ma szerokie zastosowanie w programowaniu w języku C++, a oto kilka głównych obszarów, w których jest wykorzystywany:

-Testowanie jednostkowe (Unit Testing): Google Test jest głównie używane do testowania jednostkowego, czyli testowania pojedynczych funkcji lub metod w izolacji. Pomaga to w szybkim wykrywaniu błędów na wczesnym etapie rozwoju oprogramowania.

-Testowanie regresyjne: Po wprowadzeniu nowych funkcjonalności do kodu, Google Test pozwala na uruchomienie testów, które sprawdzają, czy nowe zmiany nie wprowadziły błędów w już działających częściach aplikacji.

-Testowanie bibliotek i komponentów: Google Test jest używane do testowania zewnętrznych bibliotek lub własnych komponentów w projekcie C++. Pozwala na upewnienie się, że interfejsy API działają zgodnie z oczekiwaniami.

-Testowanie w systemach CI/CD: Google Test jest często używane w ramach procesów Continuous Integration i Continuous Deployment, gdzie testy są automatycznie uruchamiane na każdym etapie procesu tworzenia oprogramowania.

-Testowanie wielowątkowości: Google Test umożliwia także testowanie aplikacji wielowątkowych. Chociaż nie jest specjalnie zaprojektowane z myślą o tym, istnieją narzędzia i techniki, które pozwalają na integrację Google Test z testowaniem kodu wielowątkowego.

3.0.3. Sposób działania

1. Tworzenie testów: Aby stworzyć test w Google Test, definiujemy klasę testową, która dziedziczy po klasie ::testing::Test. W tej klasie definiujemy metody testowe,

które będą wykonywane w ramach testów. Każdy test jest opakowany w makra, takie jak TEST lub TEST_F, w zależności od tego, czy chcemy używać wspólnego zestawu danych dla wszystkich testów.

2. Definiowanie testów: Testy są definiowane za pomocą makr takich jak:

TEST(TestName, TestBody): Tworzy pojedynczy test, gdzie TestName to nazwa grupy testowej, a TestBody to nazwa samego testu. EXPECT_EQ(a, b): Sprawdza, czy a jest równe b. Jest to jedna z wielu asercji (sprawdzania warunków) dostępnych w Google Test.

3. Uruchamianie testów: Po napisaniu testów uruchamiamy je za pomocą kompilatora C++ oraz Google Test, który sprawdza, czy testy przeszły pomyślnie (czy wyniki testów są zgodne z oczekiwaniami).

4. Raportowanie wyników: Google Test generuje raporty o wynikach testów. Raporty te wskazują, które testy zakończyły się sukcesem, a które nie powiodły się (z podaniem przyczyny błędu).

5. Integracja z systemem budowania: Google Test jest kompatybilny z popularnymi narzędziami do kompilacji i budowania projektów, takimi jak CMake, co ułatwia integrację z istniejącymi projektami.

3.0.4. Wykorzystanie w projekcie

gtest zsotał wykorzystany do sprawdzenia poprawnego działania algorytmu napisanego w c++. Odpowiedni kod znajduje się na listingu 11[1]

3.1. c++

3.1.1. Co to c++

C++ to język programowania ogólnego przeznaczenia, który rozszerza język C o mechanizmy programowania obiektowego, a także inne zaawansowane funkcje. Jego ikona znajduje się na rysunku .3.1

3.1.2. zastosowanie

C++ jest używany w wielu dziedzinach, w tym:

Tworzenie systemów operacyjnych: np. fragmenty systemów takich jak Windows czy Linux.

Aplikacje desktopowe i mobilne: edytory tekstu, programy graficzne, oprogramowanie inżynieryjne.

Rys. 3.1. ikona c++

Programowanie gier: ze względu na wydajność i możliwość pracy z grafiką oraz pamięcią.

Systemy wbudowane: np. oprogramowanie dla mikrokontrolerów. Przetwarzanie wysokowydajne

3.1.3. Sposób działania

C++ umożliwia programowanie zarówno proceduralne, jak i obiektowe. Jego cechy obejmują:

Programowanie obiektowe: C++ wprowadza klasy i obiekty, co pozwala na grupowanie danych oraz funkcji w struktury, co ułatwia zarządzanie złożonością kodu.

Dziedziczenie: Możliwość tworzenia nowych klas na podstawie istniejących, co wspiera ponowne używanie kodu.

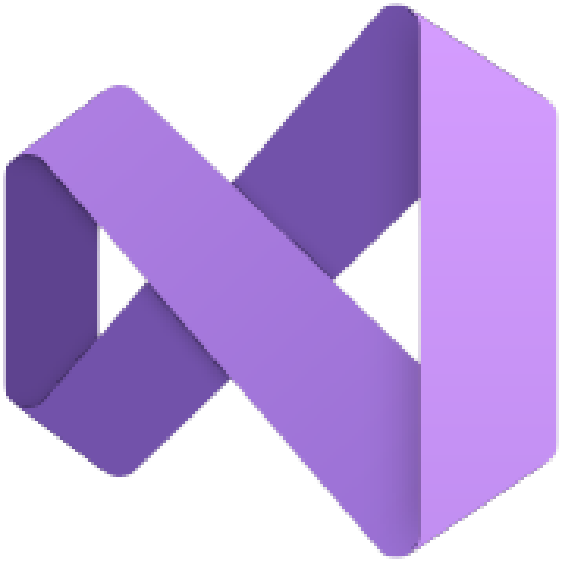
Polimorfizm: Pozwala funkcjom i obiektom przyjmować różne formy w zależności od kontekstu.

Zarządzanie pamięcią: W C++ mamy pełną kontrolę nad pamięcią, co pozwala na dynamiczne alokowanie i zwalnianie zasobów.[2]

3.2. visual studio

3.2.1. Co to Visual Studio

Visual Studio to zintegrowane środowisko programistyczne stworzone przez firmę Microsoft. Jest to narzędzie używane do tworzenia, testowania i debugowania aplikacji w różnych językach programowania, takich jak C++, C#, Python, JavaScript, i wielu innych. Jego ikona znajduje się na rysunku 3.2



Rys. 3.2. Ikona VS

3.2.2. Zastosowanie

Visual Studio jest używane przez programistów do tworzenia różnych typów oprogramowania:

Aplikacje desktopowe (Windows Forms, WPF),

Aplikacje webowe (ASP.NET, HTML, CSS, JavaScript),

Aplikacje mobilne (Xamarin, .NET MAUI),

Gry (Unity, Unreal Engine),

Usługi chmurowe (Microsoft Azure).

3.2.3. sposób działania

Visual Studio ułatwia tworzenie oprogramowania, oferując szereg funkcji, które wspierają proces tworzenia kodu:

Edytor kodu: Oferuje wsparcie dla składni wielu języków, podświetlanie kodu, autouzupełnianie, a także sugestie podczas pisania.

Kompilator i debugger: Visual Studio pozwala kompilować kod do plików wykonywalnych i automatycznie wykrywać oraz naprawiać błędy w kodzie.

IntelliSense: Zaawansowana funkcja podpowiedzi podczas pisania kodu, która analizuje kod i proponuje odpowiednie funkcje, metody czy zmienne.

Integracja z systemami kontroli wersji: Visual Studio obsługuje Git, co pozwala zarządzać kodem, śledzić zmiany i współpracować z zespołem bezpośrednio z poziomu środowiska IDE.

Debugowanie i profilowanie: Narzędzia do śledzenia błędów oraz wydajności kodu, które pomagają analizować i optymalizować aplikacje.

Szablony projektów: Visual Studio oferuje szereg gotowych szablonów projektów, które ułatwiają rozpoczęcie pracy nad aplikacjami webowymi, desktopowymi czy mobilnymi.[3]

3.3. doxygen

3.3.1. Co to doxygen?

Doxygen to narzędzie do automatycznego generowania dokumentacji technicznej z kodu źródłowego. Obsługuje wiele języków programowania, takich jak C, C++, Java, Python, Fortran, PHP i inne. Doxygen skanuje kod źródłowy i na podstawie specjalnych komentarzy generuje szczegółową dokumentację w formacie HTML, PDF, LaTeX, czy RTF. Jego ikona znajduje się na rysunku 3.3

Rys. 3.3. Ikona doxygen

3.3.2. zastosowanie

Doxygen jest szeroko stosowany w projektach programistycznych, aby generować dokumentację bezpośrednio na podstawie kodu. Jest szczególnie popularny w zespołach pracujących z językami C i C++, ale także w innych środowiskach, które wymagają dokumentacji technicznej. Przykładowe zastosowania:

Dokumentacja bibliotek i API: Umożliwia automatyczne tworzenie dokumentacji dla publicznych interfejsów, funkcji, klas, metod i zmiennych.

Projekty open-source: Doxygen jest popularny w projektach open-source, ponieważ pozwala na łatwe udostępnienie dokumentacji użytkownikom i programistom pracującym nad kodem.

Zarządzanie dużymi projektami: Dzięki automatycznemu generowaniu dokumentacji, Doxygen pomaga programistom zrozumieć struktury dużych i złożonych projektów.

3.3.3. sposób działania

Doxygen działa poprzez analizowanie specjalnych komentarzy w kodzie, które są oznaczane za pomocą znaczników podobnych do tych w języku HTML. Po uruchomieniu Doxygen generuje dokumentację na podstawie tych komentarzy oraz samego kodu, opisując struktury takie jak klasy, funkcje, zmienne i ich relacje.[4]

4. Implementacja

Program implementuje sortowanie przez scalanie w języku C++.

4.1. Klasa

4.1.1. Deklaracja klasy i jej metod

```
class merger {
private:
   string* tab;
   int* tabs;
   public:

   merger(string a);
   ~merger();
   int check(int i, string a);
   int tablen(string a);
   void returntab(string a);
   void tabtotabs(int a);
   void sort(int a);
   string returntabs(string a);
   int* tabser();
};
```

Listing 1. Klasa merger

Klasa definjuje zmienne tab i tabs oraz metody zgodne z listingiem 1.

4.1.2. konstruktor

```

   merger::merger(string a) {
   if (a != "") {
       int i = 0;
       int tabcheck = 0;
       int spaces = tablen(a);
       tab = new string[spaces + 1];
       tabs = new int[spaces + 1];

       while (i < a.length()) {
           if (a[i] == ' ') {
               i += 1;
```

```
13                 continue;
14             }
15
16             int wordLength = check(i, a);
17
18             tab[tabcheck] = a.substr(i, wordLength);
19             tabcheck += 1;
20
21             i += wordLength;
22         }
23         tabtotabs(spaces);
24         sort(spaces);
25     }
26     else {
27         tab = new string[1];
28         tab[0] = "";
29         tabs = new int[0];
30     }
31
32 }
```

Listing 2. Konstruktor

Konstuktor wykorzystuje pozostałe metody aby posortować podaną tablicę 2.

4.1.3. Destruktor

```
1 merger::~merger() {
2     delete[] tab;
3     delete[] tabs;
4 };
```

Listing 3. Destruktor

Destruktor usuwa zmienne dynamiczne tab i tabs 3.

4.1.4. check

```
1 int merger::check(int i, string a) {
2     int length = 0;
3     while (i + length < a.length() && a[i + length] != ' ') {
4         length += 1;
5     }
6     return length;
7 }
```

Listing 4. check

metoda sprawdza długość podanego elementu tablicy 4

4.1.5. tablen

```
int merger::tablen(string a) {
    int spaces = 0;
    for (int j = 0; j < a.length(); j++) {
        if (a[j] == ' ') {
            spaces += 1;
        }
    }
    return spaces + 1;
}
```

Listing 5. tablen

metoda sprawdza jaką długość powinna mieć posortowana tablica 5

4.1.6. returntab

```
void merger::returntab(string a) {
    int i = 0;
    while (i < tablen(a)) {
        cout << tab[i] << " , ";
        i++;
    }
}
```

Listing 6. returntab

Metoda wypisuje na ekranie podany ciąg liczb do posortowania 6

4.1.7. tabtotabs

```
void merger::tabtotabs(int spaces) {
    int i = 0;
    while (i < (spaces)) {
        tabs[i] = stoi(tab[i]);
        i++;
    }
}
```

Listing 7. tabtotabs

Metoda konwertuje string tab do tablicy int tabs 7

4.1.8. sort

```
void merger::sort(int spaces) {
    int mnoznik = 1;

    while (mnoznik < spaces) {
        for (int i = 0; i < spaces; i += (2 * mnoznik)) {
            int mid = min(i + mnoznik, spaces);
            int end = min(i + 2 * mnoznik, spaces);


            int* temp = new int[spaces];
            int left = i, right = mid, k = i;


            while (left < mid && right < end) {
                if (tabs[left] <= tabs[right]) {
                    temp[k++] = tabs[left++];
                }
                else {
                    temp[k++] = tabs[right++];
                }
            }


            while (left < mid) {
                temp[k++] = tabs[left++];
            }


            while (right < end) {
                temp[k++] = tabs[right++];
            }


            for (int j = i; j < end; j++) {
                tabs[j] = temp[j];
            }

            delete[] temp;
        }

        mnoznik *= 2;
    }
```

```
}
```

Listing 8. sort

Metoda sortuje tablicę tabs 8

4.1.9. returntabs

```
string merger::returntabs(string a) {
    if (tab[0] == "") {
        return "";
    }
    else {
        int i = 0;
        string r;
        while (i < tablen(a)) {
            r += to_string(tabs[i]);
            if (i != (tablen(a) - 1)) {
                r += " ";
            }
            i++;
        }
        return  r;
    }
}
```

Listing 9. returntabs

Metoda zwraca tablicę tabs zamieniając ją na string 9

4.1.10. tabser

```
    int* merger::tabser() {
    return tabs;
}
```

Listing 10. tabser

metoda zwraca wskaźnik do tablicy tabs10

4.2. testy

4.2.1. TEST1

```
TEST(Testyogolne, Zachowanie_niezmienionej_tablicy) {
  string input = ("1 2 3 4 5 6 7");
```

```
3   merger a(input);
4   EXPECT_EQ(input, a.returntabs(input));
5 }
```

Listing 11. TEST1

Test ten sprawdza czy program nie zmieni już posortowanej tablicy 11

4.2.2. TEST2

```
1 TEST(Testyogolne, sortowanie_odwrotnej_tablicy) {
2   string input = ("10 9 8 7 6 5 4 3 2 1 0");
3   merger a(input);
4   EXPECT_EQ("0 1 2 3 4 5 6 7 8 9 10", a.returntabs(input));
5 }
```

Listing 12. TEST2

Test ten sprawdza czy program posortuje odwróconą tablicę 12

4.2.3. TEST3

```
1     TEST(Testyogolne, randomowa_tablica) {
2   srand(static_cast<unsigned int>(time(0)));
3   int i = 0;
4   string r;
5   int t = rand() % 100 + 1;
6   while (i < t) {
7     r += to_string(rand() % 10000 - 2001);
8     if (i != (t - 1)) {
9       r += " ";
10    }
11    i++;
12  }
13
14  merger a(r);
15
16  int spaces = 0;
17  for (int j = 0; j < r.length(); j++) {
18    if (r[j] == ' ') {
19      spaces += 1;
20    }
21  }
22
23
24  int j = 0;
```

```
25   int* tab = a.tabser();
26   while (j < spaces - 1) {
27     if (tab[j] == tab[j + 1]) {
28       EXPECT_EQ(tab[j], tab[j + 1]);
29     }
30     else {
31       EXPECT_LT(tab[j], tab[j + 1]);
32     }
33     j += 1;
34   }
35 }
```

Listing 13. TEST3

Test ten sprawdza czy program posortuje randomową tablicę 13

4.2.4. TEST4

```
1 TEST(Testyogolne, ujemne_elementy) {
2   string input = ("-1, -2 -54 -67 -213 -678 -12 -9 -78 -3 -123342
      -1456");
3   merger a(input);
4   EXPECT_EQ("-123342 -1456 -678 -213 -78 -67 -54 -12 -9 -3 -2 -1",
      a.returntabs(input));
5 }
```

Listing 14. TEST4

Test ten sprawdza czy program posortuje tablicę z ujemnymi elementami 14

4.2.5. TEST5

```
1 TEST(Testyogolne, elementy_dodatnie_i_ujemne) {
2   string input = ("5 1 7 2 4 3 6 8 0 -2 -1");
3   merger a(input);
4   EXPECT_EQ("-2 -1 0 1 2 3 4 5 6 7 8", a.returntabs(input));
5 }
```

Listing 15. TEST5

Test ten sprawdza czy program posortuje tablicę z ujemnymi i dodatnimi elementami 15

4.2.6. TEST6

```
TEST(Testyogolne, sortowanie_tablicy_bez_elemenow) {
  string input = ("");
  merger a(input);
  EXPECT_EQ("", a.returntabs(input));
}
```

Listing 16. TEST6

Test ten sprawdza czy program nie wyrzuci wyjątku przy pustej tablicy 16

4.2.7. TEST7

```
TEST(Testyogolne, sortowanie_tablicy_jeden_element) {
  string input = ("1");
  merger a(input);
  EXPECT_EQ("1", a.returntabs(input));
}
```

Listing 17. TEST7

Test ten sprawdza czy program psosrtuje tablicę z jednym elementem 17

4.2.8. TEST8

```
TEST(Testyogolne, sortowanie_tablicy_powtarzajace_elementy) {
  string input = ("1 1 1 3 3 3 2 2 2 2 2 25 12 12 12");
  merger a(input);
  EXPECT_EQ("1 1 1 2 2 2 2 2 3 3 3 12 12 12 25", a.returntabs(input
   ));
}
```

Listing 18. TEST8

Test ten sprawdza czy program posortuje tablicę z powtórzającymi się elementami 18

4.2.9. TEST9

```
TEST(Testyogolne, sortowanie_tablicy_powtarzajace_ujemne_elementy)
   {
  string input = ("-1 -1 -1 -3 -3 -3 -2 -2 -2 -2 -2 -25 -12 -12 -12
   ");
  merger a(input);
  EXPECT_EQ("-25 -12 -12 -12 -3 -3 -3 -2 -2 -2 -2 -2 -1 -1 -1", a.
   returntabs(input));
```

```
}
```

Listing 19. TEST9

Test ten sprawdza czy program posortuje tablicę z powtórzającymi się ujemnymi elementami 19

4.2.10. TEST10

```
TEST(Testyogolne,
    sortowanie_tablicy_powtarzajace_elementy_dodatnie_ujemne) {
  string input = ("-1 -1 1 3 -3 -3 2 2 -2 -2 2 25 12 -12 12");
  merger a(input);
  EXPECT_EQ("-12 -3 -3 -2 -2 -1 -1 1 2 2 2 3 12 12 25", a.
    returntabs(input));
}
```

Listing 20. TEST10

Test ten sprawdza czy program posortuje tablicę z powtórzającymi się ujemnymi i dodatnimi elementami 20

4.2.11. TEST11

```
TEST(Testyogolne, sortowanie_tablicy_dwa_elementy_rosnaco) {
  string input = ("1 3");
  merger a(input);
  EXPECT_EQ("1 3", a.returntabs(input));
}
```

Listing 21. TEST11

Test ten sprawdza czy program posortuje 2 elementy rosnąco 21

4.2.12. TEST12

```
TEST(Testyogolne, randomowa_tablica_wiŹksza_niz_100){
  srand(static_cast<unsigned int>(time(0)));
  int i = 0;
  string r;
  int t = rand() % 100 + 100;
  while (i < t) {
    r += to_string(rand() % 10000 - 2001);
    if (i != (t - 1)) {
      r += " ";
    }
```

```
    i++;
  }

  merger a(r);

  int spaces = 0;
  for (int j = 0; j < r.length(); j++) {
    if (r[j] == ' ') {
      spaces += 1;
    }
  }


  int j = 0;
  int* tab = a.tabser();
  while (j < spaces - 1) {
    if (tab[j] == tab[j + 1]) {
      EXPECT_EQ(tab[j], tab[j + 1]);
    }
    else {
      EXPECT_LT(tab[j], tab[j + 1]);
    }
    j += 1;
  }
}
```

Listing 22. TEST12

Test ten sprawdza czy program posortuje randomową tablicę o wielkości większej niż 100 22

4.2.13. TEST13

```
TEST(Testyogolne, randomowa_tablica_wieksza_niz_100_dod_uj_dup) {
  srand(static_cast<unsigned int>(time(0)));
  int i = 0;
  string r;
  int t = rand() % 100 + 100;
  while (i < t) {
    int y = rand() % 4000 - 2001;
    r += to_string(y);
    r += " ";
    r += to_string(y);
    if (i != (t - 1)) {
      r += " ";
```

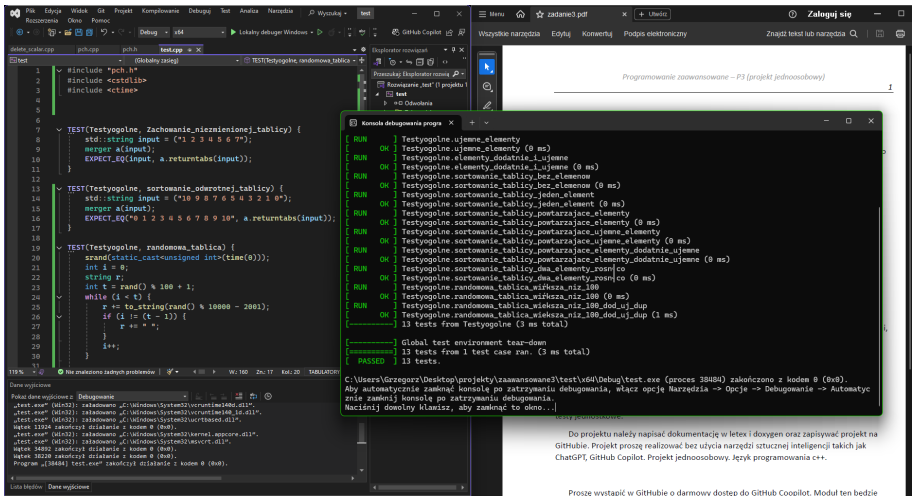
```
13      }
14      i++;
15    }
16
17    merger a(r);
18
19    int spaces = 0;
20    for (int j = 0; j < r.length(); j++) {
21      if (r[j] == ' ') {
22        spaces += 1;
23      }
24    }
25
26
27    int j = 0;
28    int* tab = a.tabser();
29    while (j < spaces - 1) {
30      if (tab[j] == tab[j + 1]) {
31        EXPECT_EQ(tab[j], tab[j + 1]);
32      }
33      else {
34        EXPECT_LT(tab[j], tab[j + 1]);
35      }
36      j += 1;
37    }
38  }
```

Listing 23. TEST13

Test ten sprawdza czy program posortuje randomową tablicę o wielkości większej niż 100 z elementami dodatnimi, ujemnymi i duplikatami 23

4.3. Wynik działania programu

Rysunek 4.1 przedstawia wynik poprawnego działania programu.



Rys. 4.1. działający program

5. Wnioski

Program poprawnie sortuje podane liczby co pokazują wyniki gtest 4.1.

Podczas realizacji mojego projektu dotyczącego implementacji sortowania przez scalanie, używałem Google Test do testowania poprawności algorytmu. Moja opinia na temat tego narzędzia jest bardzo pozytywna, zwłaszcza w kontekście testowania takich algorytmów jak sortowanie, gdzie precyzyjne sprawdzenie wyników jest kluczowe.

Dzięki Google Test proces tworzenia testów jednostkowych stał się niezwykle prosty. Chociaż algorytm sortowania przez scalanie jest stosunkowo skomplikowany, Google Test umożliwił mi łatwe i szybkie stworzenie testów sprawdzających różne przypadki, takie jak sortowanie już posortowanych danych, danych w odwrotnej kolejności, a także przypadków brzegowych z małą liczbą elementów.

W projekcie musiałem zweryfikować, czy sortowanie jest przeprowadzane poprawnie, zwłaszcza w odniesieniu do stabilności algorytmu (czy elementy o tej samej wartości są sortowane w tej samej kolejności). Dzięki bogatemu zestawowi asercji w Google Test, mogłem bez problemu stworzyć odpowiednie warunki testowe i upewnić się, że algorytm działa zgodnie z oczekiwaniami. Automatyzacja testów:

Google Test świetnie sprawdziło się w automatyzacji testowania. Dzięki możliwości uruchomienia testów w jednym kroku mogłem zaoszczędzić sporo czasu, szczególnie przy testowaniu różnych zestawów danych wejściowych. Integracja testów z procesem kompilacji była również bardzo prosta, co pozwoliło mi szybko reagować na zmiany w kodzie. Przejrzystość wyników:

Kolejnym atutem Google Test była przejrzystość wyników testów. Gdy któryś

z testów nie przeszedł, raport zawierał dokładne informacje o tym, która asercja zawiodła, co znacznie przyspieszyło debugowanie i poprawę błędów w algorytmie. Zalety dla rozwoju projektu:

Google Test pomogło mi nie tylko w testowaniu poprawności samego algorytmu sortowania, ale także w rozwoju projektu. Każda zmiana w kodzie była natychmiastowo sprawdzana pod kątem błędów, co zminimalizowało ryzyko wprowadzenia nowych problemów do istniejącego kodu. Podsumowując, Google Test okazało się doskonałym narzędziem do testowania mojego projektu o sortowaniu przez scalanie. Umożliwiło mi ono dokładne i efektywne testowanie algorytmu, co pozwoliło na szybkie wykrycie ewentualnych błędów oraz zapewnienie, że implementacja działa zgodnie z założeniami. Bez wątpienia polecam Google Test do każdego projektu C++, w którym testowanie jednostkowe jest kluczowe.

Bibliografia

[1] *Wikipedia Google test*. URL: https://en.wikipedia.org/wiki/Google_Test.

[2] *Wikipedia c++*. URL: https://pl.wikipedia.org/wiki/C%2B%2B.

[3] *Strona internetowa Visual studio*. URL: https://visualstudio.microsoft.com/pl/.

[4] *Strona internetowa Doxygen'a*. URL: https://doxygen.nl/.

Spis rysunków

Spis listingów