AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm listy dwukierunkowej z zastosowaniem GitHub

Autor:
Kamil Trzópek

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Napisz program sortowanie przez scalanie (MergeSort). Algorytm musi być zaimplementowany w klasie. Funkcja main ma być zaimplementowana w osobnym pliku. Za pomocą „Google test" należy wykonać testy jednostkowe algorytmu. W Visual Studio wybiera się „Google Test" (unit tests, c++). Testami należy sprawdzić czy algorytm:

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

liczbami ujemnymi, dodatnimi oraz duplikatami,

2. Analiza problemu

Merge Sort (sortowanie przez scalanie) to jeden z najbardziej wydajnych algorytmów sortowania, oparty na podejściu "dziel i zwyciężaj"(divide and conquer). Jego głównym założeniem jest podzielenie problemu na mniejsze części, uporządkowanie ich, a następnie scalenie w jedną całość (Strona intrnetowa zawierajaca informacje o marge sort:[1]).

Działanie magre sort:

Dla tablicy [8, 4, 7, 3, 2, 6, 5, 1]:

Krok 1: Podział: Tablica dzieli się na dwie części: [8, 4, 7, 3] i [2, 6, 5, 1].

Krok 2: Sortowanie rekurencyjne:

[8, 4, 7, 3] → [4, 8] i [3, 7] → [3, 4, 7, 8]

[2, 6, 5, 1] → [2, 6] i [1, 5] → [1, 2, 5, 6]

Krok 3: Scalanie:

[3, 4, 7, 8] i [1, 2, 5, 6] → [1, 2, 3, 4, 5, 6, 7, 8]

Cechy algorytmu sortowania przez scalanie:

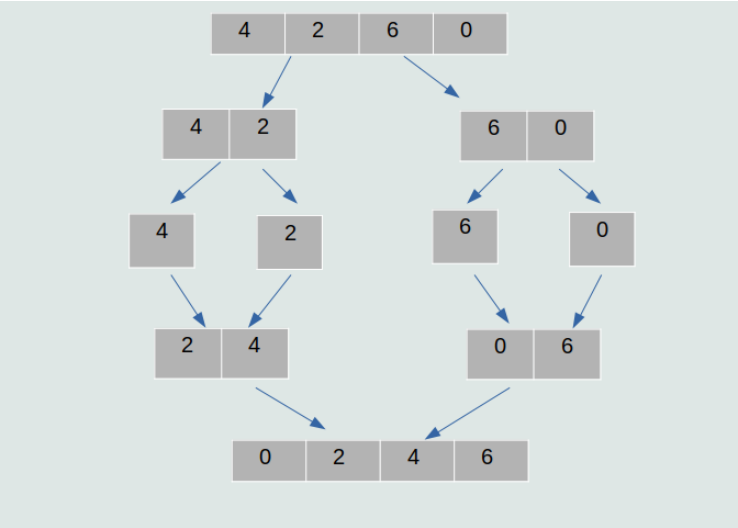
- Złożonośc czasowa: w najgorszym przypadku jest to O(nlogn)
- Złożoność pamięciowa: O(n) (wymagana dodatkowa tablica do scalania)]
- Stabilność: Merge Sort jest algorytmem stabilnym, co oznacza, że nie zmienia kolejności równoważnych elementów.

Prosty przykład działania algorytmu sortowania morge sort (Rys:10)

```
for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];
```

Listing 1. marge sort lewo

```
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
```



Rys. 2.1. Coline

```
8           }
9           k++;
10      }
11
12      while (i < n1) {
13          arr[k] = L[i];
14          i++;
15          k++;
16      }
17
18      while (j < n2) {
19          arr[k] = R[j];
20          j++;
21          k++;
22      }
23  }
```

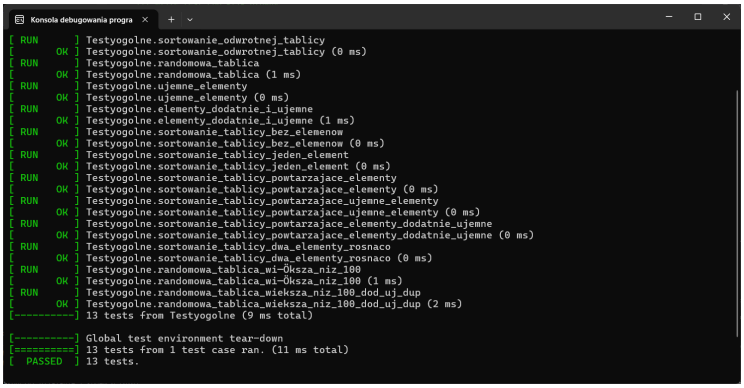
Listing 2. marge sort

3. Projektowanie

Program do projektu został napisany w języku programowania C++, przy użyciu edytora Visual Studio Code **??** i kompilatora zawartego w programie Visual Studio 2022.Repozytorium projektu zostało utworzone w serwisie Github(Strona projektu:[2]), także dokumentacja została wygenerowana automatycznie przy użyciu "Doxywizard"obsługiwana przez program doxygen(Strona programu doxygen:Ppozwala on na szybkie przetestowanie itewww4).

Co to Gtest?:

GTest (Google Test) to popularna biblioteka do testowania jednostkowego (unit testing) w języku C++. Została stworzona przez Google i jest szeroko stosowana do testowania aplikacji w tym języku. Pozwala on na szybkie przetestowanie poprawnego przetestowania programu w kalku przypadkach na końcu zwracając informację (Informacja przedstawienie:(Rys3.1))



Rys. 3.1. Test

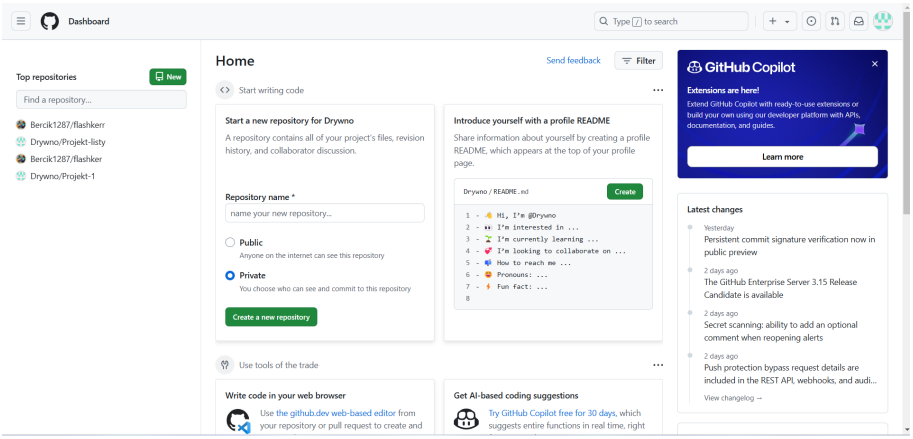
Gtest w projekcie został zastosowany w celu sprawdzenia poprawności sortowania w podanych przypadkach (Wynik:(Rys3.1))

Github:

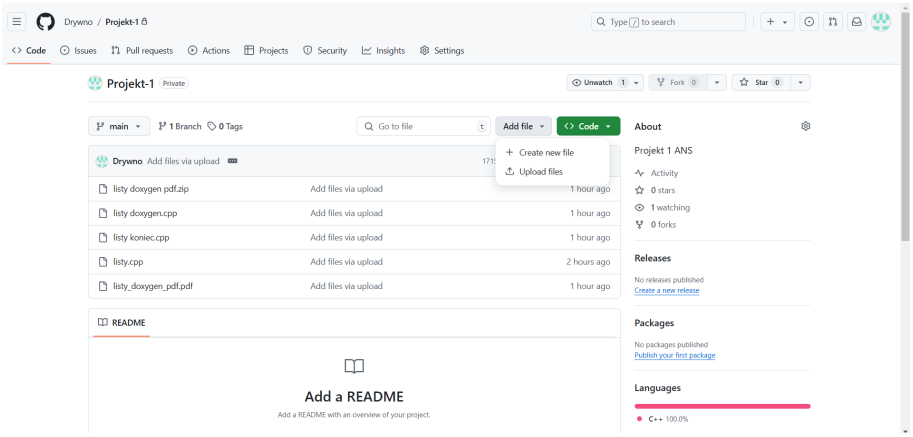
Github pozwala nam na dodawanie commitów. W projekcie używam wercji przeglądarkowej znajdującej się na stronie githuba (link github:(Link:[3]), gdzie wykonujemy to za pomocą przycisku upload files, oraz wybieramy do jakiej gałęzi przypisujemy commit 3.3. (Wcześniej dodajemy repetutoruim o ile juz nie mamy gotowych3.2).

Wybieramy w interesujacy nas plik, oraz dodajemy do niego komentarz a następnie naciskamy przycisk "commit changes"co poskutkuje ich dodaniem 3.4.

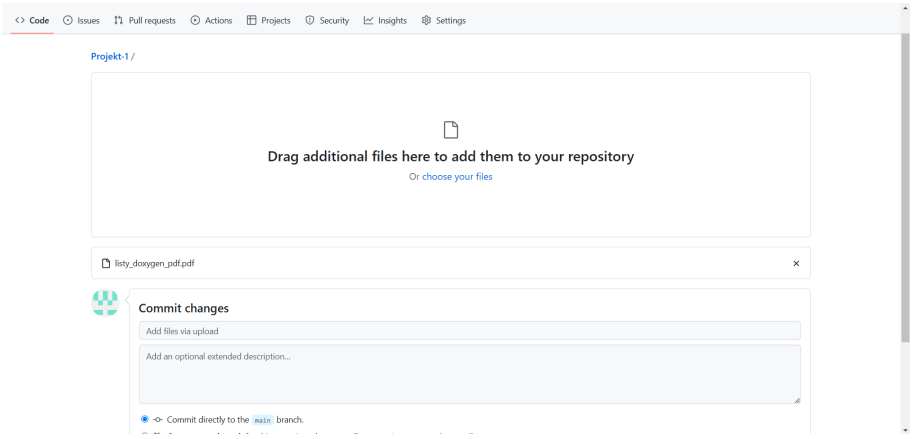
Generowanie dokumentacji za pomocą doxygen:



Rys. 3.2. Dodawanie plików github



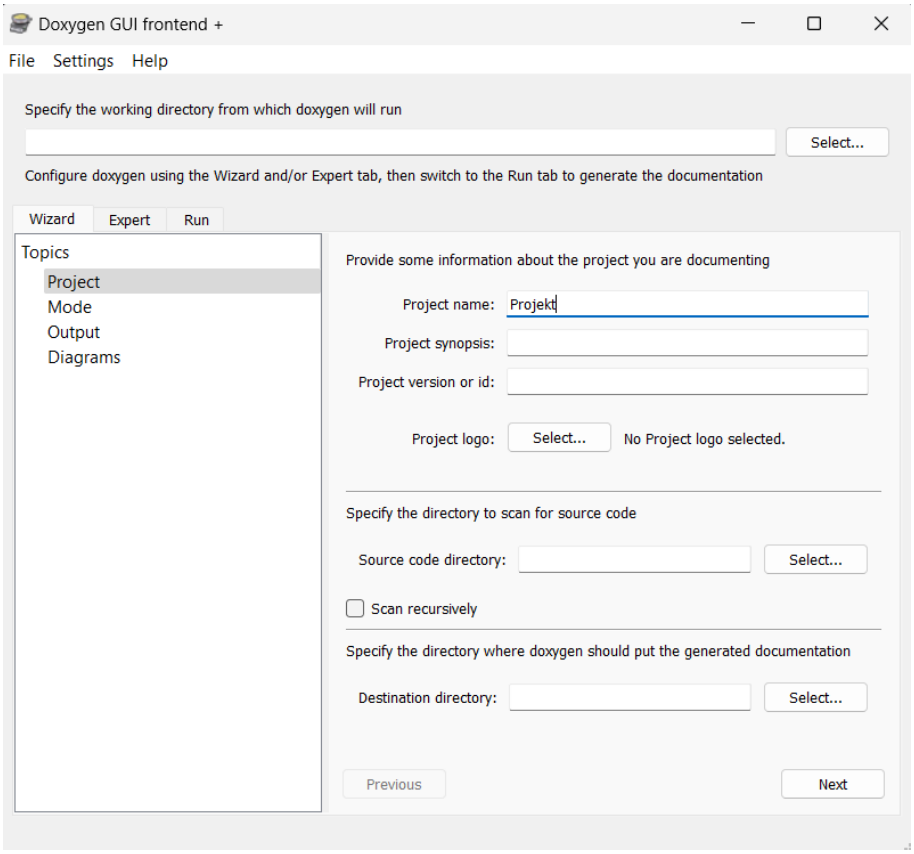
Rys. 3.3. Dodawanie plików github



Rys. 3.4. Plik github

Pobieramy program doxwizard ze strony (strona doxygen (link:[4])), oraz go instalujemy. Następnie po uruchomieniu podejmy miejsce z którego pliku ma generować doxygen (Rys:3.5), oraz w jakim miejscu ma zostać umieszczony odpowiedni

plik (jesli nie podamy lokalizacji zostanie on wygenerowany w miejscu wskazanym do generowania (Rys:3.6) (latex,pdf,html itp...), następnie w zakładce run włączamy generowanie (przycisk run (Rys:3.7)).



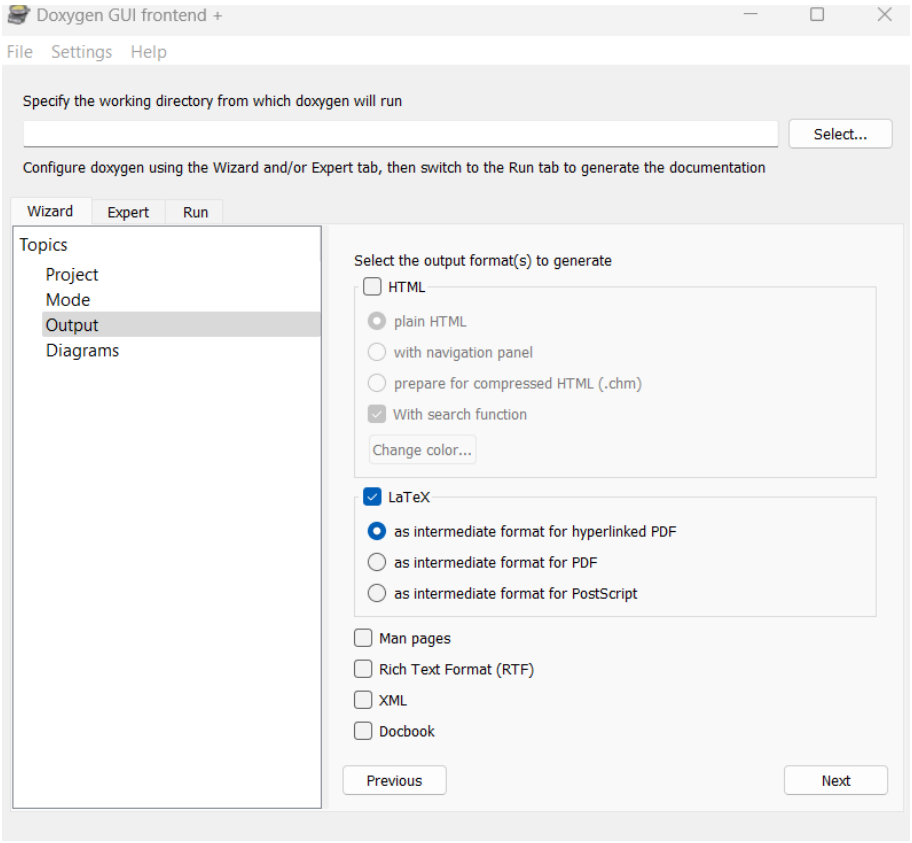
Rys. 3.5. generacja doxygen

Język programowania C++:

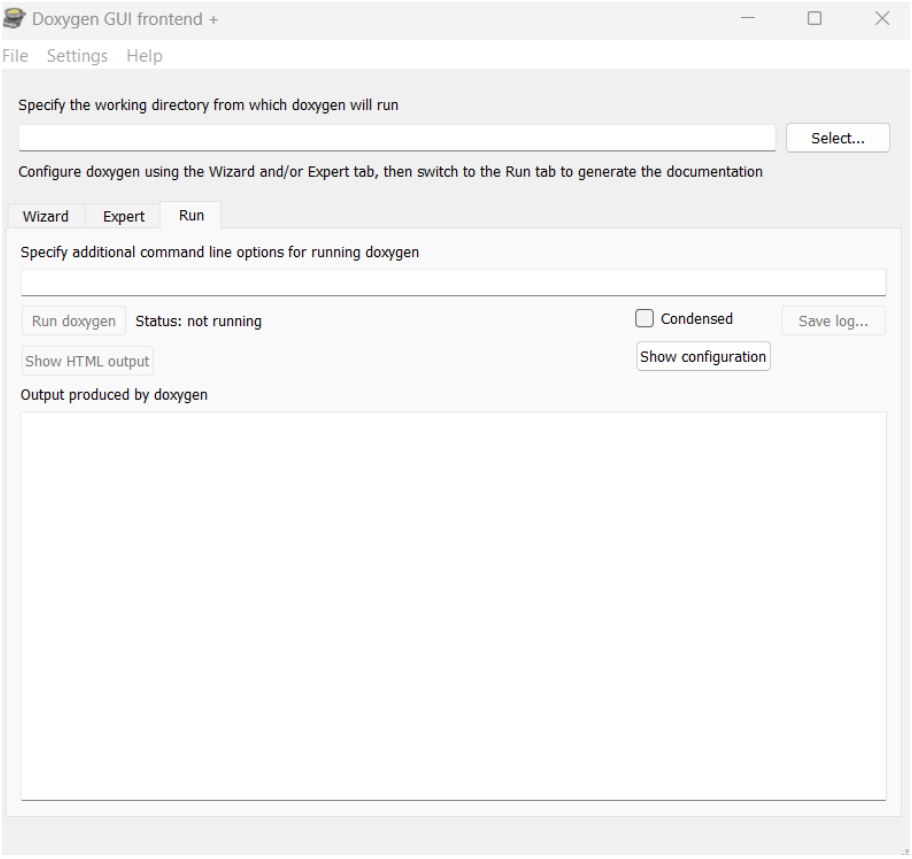
C++ to wszechstronny i wydajny język programowania. Dzięki swojej elastyczności i możliwościom niskopoziomowym, C++ znalazł zastosowanie w wielu dziedzinach, takich jak rozwój oprogramowania systemowego, gry komputerowe, aplikacje wbudowane. Język porogramowania został zastosowany w projekcie w celu wykonania sortowania marge sort.

Visual Studio 2022:

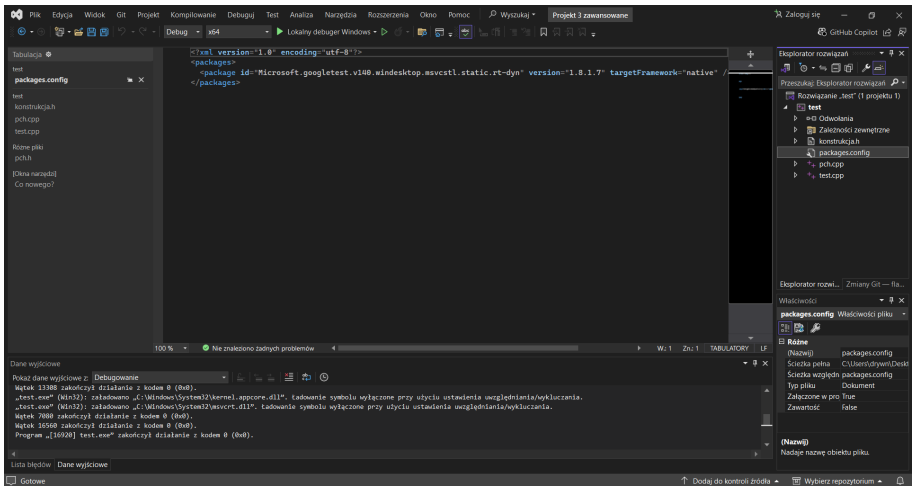
Visual Studio 2022 (Rys:3.8) to wszechstronne środowisko programistyczne (IDE) firmy Microsoft, które oferuje zaawansowane wsparcie dla programistów języka C++. Jest to jedno z najpopularniejszych narzędzi używanych do tworzenia aplikacji w tym języku, dzięki szerokiej gamie funkcji ułatwiających pisanie, debugowanie, testowanie i wdrażanie kodu. Program można zainstalować: (link:[5]).



Rys. 3.6. generacja doxygen



Rys. 3.7. generacja doxygen



Rys. 3.8. Visual studio 2022

4. Implementacja

Przedstawienie zawartości pliku pch.h (konstrukcyjnego) zawierającego klasę prywatną tablica, oraz tablicapostowana (linia kodu 3 i 4), dająca możliwość stworzenia konstrukcji tablicy. (czyli w skrócie deklaracja klasy i metod) (Listing:3)

```
1  class scalacz {
2  private:
3      string* tablica; ///< wska nik do zmiennej zawieraj cej
       nieposortowan  tablic Ź typu string
4      int* tablicaposortowana; ///< wska nik do tablicy
       zawieraj cej posortowan  tablic Ź
5
6  public:
7      scalacz(string a); ///< Konstruktor klasy scalacz
8      ~scalacz(); ///< Destruktor klasy scalacz
9      int sprawdz(int i, string a); ///< Funkcja do sprawdzania
       element w w tablicy
10     int dlugoscTablicy(string a); ///< Funkcja zwracaj ca
       d Ćugo Ż  tablicy
11     void zwrocTablice(string a); ///< Funkcja zwracaj ca tablic Ź
       (nieposortowan )
12     void tablicatotabposortowana(int a); ///< Funkcja do konwersji
       tablicy
13     void sortuj(int a); ///< Funkcja sortuj ca tablic Ź
14     string zwracajTablice(string a); ///< Funkcja zwracaj ca
       posortowan  tablic Ź jako string
15     int* pobierzTablice(); ///< Funkcja zwracaj ca wska nik na
       posortowan  tablic Ź
16 };
```

Listing 3. Kontrukcja

Przedstawiony fragment kodu (Listing:4) to kontruktor (deklaracja linia kodu:1), który pozwala na stwoerzenie, oraz przejście przez utworzoną tablice (pętla while linia kodu:9-25)

```
1  scalacz::scalacz(string a) {
2      if (a != "") {
3          int i = 0;
4          int tabcheck = 0;
5          int spaces = dlugoscTablicy(a);
6          tablica = new string[spaces + 1];
7          tablicaposortowana = new int[spaces + 1];
8
```

```
9           while (i < a.length()) {
10              if (a[i] == ' ') {
11                  i += 1;
12                  continue;
13              }
14
15              int wordLength = sprawdz(i, a);
16
17              tablica[tabcheck] = a.substr(i, wordLength);
18              tabcheck += 1;
19
20              i += wordLength;
21          }
22          tablicatotabposortowana(spaces);
23          sortuj(spaces);
24      }
25  };
```

Listing 4. Kontruktor

Destruktor (Listing:5) usuwający zmienne dynamiczne (linia kadu:2-3)

```
1  scalacz::~scalacz() {
2      delete[] tablica;
3      delete[] tablicaposortowana;
4  };
```

Listing 5. Destruktor

Sprawdzanie długości danego elementu tablicy (Listing:6

```
1  int scalacz::sprawdz(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 6. Sprawdzanie długości elemetu

Metoda dlugoscTablicy pozwalająca na sprawdzanie jaką długość powinna mieć podana tablica (Listing:7

```
1  int scalacz::dlugoscTablicy(string a) {
2      int spaces = 0;
3      for (int j = 0; j < a.length(); j++) {
4          if (a[j] == ' ') {
5              spaces += 1;
```

```
6           }
7       }
8       return spaces + 1;
9   }
```

Listing 7. Sprawdzanie długości tablicy

Metoda zwrocTablice pozwalajaca na wypisanie tablicy która ma być possortowana (Listing:8)

```
1  void scalacz::zwrocTablice(string a) {
2      int i = 0;
3      while (i < dlugoscTablicy(a)) {
4          cout << tablica[i] << " , ";
5          i++;
6      }
7  }
```

Listing 8. Wypisanie tablicy

Metoda tablicatotabposortowana konwertująca string tablicy na int konieczna do dalszej realizacji zadania (Listing:9)

```
1  void scalacz::tablicatotabposortowana(int spaces) {
2      int i = 0;
3      while (i < (spaces)) {
4          tablicaposortowana[i] = stoi(tablica[i]);
5          i++;
6      }
7  }
```

Listing 9. Koinwersja string na int

W kodzie (Listing:10) tablica jest sortowana za pomocą marge sort działającym w następujący sposób:

- Inicjalizacja mnożnika Linia 2: Zmienna mnoznik jest ustawiona na wartość 1. Określa ona długość fragmentów tablicy, które będą łączone w bieżącej iteracji.
- Główna pętla sortowania Linia 4: while (mnoznik ¡ spaces) – algorytm wykonuje kolejne iteracje dopóki długość fragmentów (mnoznik) jest mniejsza od liczby elementów w tablicy (spaces).
- Podział na fragmenty Linia 5: for (int i = 0; i ¡ spaces; i += (2 * mnoznik)) – tablica jest podzielona na fragmenty o długości 2 * mnoznik. W każdej iteracji i wskazuje początek aktualnego fragmentu.

- Wyznaczanie granic fragmentów Linia 6: int mid = min(i + mnoznik, spaces); mid to granica między lewym a prawym podfragmentem, ale nie wychodzi poza rozmiar tablicy.

  Linia 7: int end = min(i + 2 * mnoznik, spaces); end to koniec prawego podfragmentu.

- Tworzenie tablicy tymczasowej Linia 9: int* temp = new int[spaces]; Tablica temp jest tworzona, aby przechowywać posortowane elementy aktualnego fragmentu.

- Scalanie podtablic Linie 10-19: W tej sekcji scalane są dwa fragmenty:

- Linia 12: Jeśli element z lewej podtablicy jest mniejszy lub równy elementowi z prawej, jest kopiowany do temp[k]. Linia 14: W przeciwnym razie kopiowany jest element z prawej podtablicy. Indeksy left, right (dla obu podtablic) i k (dla temp) są zwiększane. Kopiowanie pozostałych elementów Linia 20-23: Jeśli w lewym podfragmencie zostały jeszcze elementy, są one kopiowane do temp. Linia 25-28: Podobnie, pozostałe elementy z prawego fragmentu również trafiają do temp.

- Przepisanie do tablicy głównej Linia 30-32: for (int j = i; j ¡ end; j++) – elementy z temp są przepisywane z powrotem do odpowiedniego fragmentu tablicy tablicaposortowana.

- Usunięcie tablicy tymczasowej Linia 34: delete[] temp; – zwalniamy pamięć zarezerwowaną na temp.

- Podwajanie długości fragmentów Linia 36: mnoznik *= 2; – długość scalanych podtablic jest podwajana, przygotowując algorytm do kolejnej iteracji.

```
1  void scalacz::sortuj(int spaces) {
2      int mnoznik = 1;
3
4      while (mnoznik < spaces) {
5          for (int i = 0; i < spaces; i += (2 * mnoznik)) {
6              int mid = min(i + mnoznik, spaces);
7              int end = min(i + 2 * mnoznik, spaces);
8
9              int* temp = new int[spaces];
10             int left = i, right = mid, k = i;
11
12             while (left < mid && right < end) {
```

```
13                  if (tablicaposortowana[left] <= tablicaposortowana[
    right]) {
14                      temp[k++] = tablicaposortowana[left++];
15                  }
16                  else {
17                      temp[k++] = tablicaposortowana[right++];
18                  }
19              }
20
21              while (left < mid) {
22                  temp[k++] = tablicaposortowana[left++];
23              }
24
25              while (right < end) {
26                  temp[k++] = tablicaposortowana[right++];
27              }
28
29              for (int j = i; j < end; j++) {
30                  tablicaposortowana[j] = temp[j];
31              }
32
33              delete[] temp;
34          }
35
36          mnoznik *= 2;
37      }
38  }
```

Listing 10. Sortowanie tablicy po przez marge sort

Metoda teraz odwrotnie zwracająca tablice jako string do jej wypisania jako wyniku działania programu (Listing:11)

```
1  int scalacz::dlugoscTablicy(string a) string scalacz::
       zwracajTablice(string a) {
2      if (tablica[0] == "") {
3          return "";
4      }
5      else {
6          int i = 0;
7          string r;
8          while (i < dlugoscTablicy(a)) {
9              r += to_string(tablicaposortowana[i]);
10             if (i != (dlugoscTablicy(a) - 1)) {
11                 r += " ";
12             }
```

```
13              i++;
14          }
15          return r;
16      }
17  }
```

Listing 11. Zwracanie tablicy jako string

Wskaznik do tablicy: (Listing:13)

```
1  int* scalacz::pobierzTablice() {
2      return tablicaposortowana;
3  }
```

Listing 12. Wskaznik do tablicy

```
1  int* scalacz::pobierzTablice() {
2      return tablicaposortowana;
3  }
```

Listing 13. Wskaznik do tablicy

Kod z pliku test.cpp (Listing:14) przedstawia główny cel zadania czyli zaznajominie się z działaniem testów jednostkowych. Całośc testó polega na wypisaniu funkcji o podanych wartościach za pomocą TEST, oraz jej przewidywanego wyniku za pomocą EXPECT EQ. Poniższe testy sprawdzaja m.in:

- Zachowanie$_n$$iezmienionej_t$$ablicy$: $Testsprawdza, czyfunkcjaniezmieniajuposortowanejtablicy.$ 6), $gdziewprowadzasitablicinput, anastpniesprawdza, czywynikzwrconyprzezfunkcjzwracajTa$

- sortowanie$_o$$dwrotnej_t$$ablicy$: $Testweryfikujepoprawnesortowanietablicywodwrotnejkolejnoci.Kodzn$ 11), $gdziezadanajestodwrotnieposortowanatablica, afunkcjapowinnaposortowajwporzdkurosncym.$

- randomowa$_t$$ablica$: $Testsprawdza, czyfunkcjapoprawniesortujetablicolosowychelementach.(Linia14$ 34)$generujelosowtablic, apotemzapomocfunkcjipobierzTabliceiptliporwnywaneskolejneelementytejt$

- ujemne$_e$$lementy$: $Testoceniazdolnofunkcjidosortowaniatablicyzawierajcejtylkoujemneliczby.W(lin$ 39)$zdefiniowanajesttablicaliczbujemnych, ktrapowinnazostapoprawnieposortowanawporzdkurosncy$

- elementy$_d$$odatnie_{iu}$$jemne$: $Testsprawdzapoprawnosortowaniatablicyzdodatnimiiujemnymiliczbami.$ 44)$jestzadanatablicazmieszanymiliczbami, ktrepowinnyzostauporzdkowanewsposbrosncy, uwzgldnia$

- sortowanie$_t$$ablicy_b$$ez_e$$lemenow$: $Testweryfikujezachowaniefunkcjiprzypustejtablicy.W(liniach47-$ 49)$tablicawejciowajestpusta, awynikfunkcjizwracajTablicerwniepowinienbypusty, cojesttestowane.$

- sortowanie$_t$$ablicy_j$$eden_e$$lement$: $Testsprawdza, czyfunkcjapoprawnieobsugujetabliczjednymelemente$ 54)$przedstawiatabliczjednymelementem, ktrypo"posortowaniu"pozostajebezzmian.$

- $\text{sortowanie}_t\text{ablicy}_p\text{owtarzajace}_e\text{lementy} : Testweryfikujesortowanietablicyzpowtarzajcymisido$
  $59)znajdujesitablicazawierajcapowtarzajcesiliczby, ktrapowinnazostaposortowanawporzdkuro$

- $\text{sortowanie}_t\text{ablicy}_p\text{owtarzajace}_u\text{jemne}_e\text{lementy} : Testsprawdzasortowanietablicyzpowtarzajcym$
  $64)znajdujesitablicazawierajcapowtarzajcesiliczbyujemne, ktrepowinnyzostaposortowanewpo$

- $\text{sortowanie}_t\text{ablicy}_p\text{owtarzajace}_e\text{lementy}_d\text{odatnie}_u\text{jemne} : Testsprawdzasortowanietablicyzpowta$
  $69)zawieratabliczliczbamimieszanymi, ktrefunkcjapowinnaposortowawporzdkurosncym, uwzg$

- $\text{sortowanie}_t\text{ablicy}_d\text{wa}_e\text{lementy}_r\text{osnaco} : Testweryfikujepoprawnesortowanietablicyzdwomaelem$
  $74)pokazujetabliczdwomaelementami, ktrepoposortowaniupozostajwtejsamejkolejnoci.$

```
/// @brief Test1, sprawdza czy tablica zostanie niezmieniona
    je eli wpiszemy ju  posortowan  tablic Ź
TEST(Testyogolne, Zachowanie_niezmienionej_tablicy) {
    string input = ("2 4 6 8 10 12 14");
    scalacz a(input);
    EXPECT_EQ(input, a.zwracajTablice(input));
}

/// @brief Test2, Sprawdza czy dobrze posortuje odwr con
    tablic Ź
TEST(Testyogolne, sortowanie_odwrotnej_tablicy) {
    string input = ("20 18 16 14 12 10 8 6 4 2 0");
    scalacz a(input);
    EXPECT_EQ("0 2 4 6 8 10 12 14 16 18 20", a.zwracajTablice(input
    ));
}

/// @brief Test3, sprawdza czy posortuje randomow  tablic Ź
TEST(Testyogolne, randomowa_tablica) {
    srand(static_cast<unsigned int>(time(0)));
    int i = 0;
    string r;
    int t = rand() % 50 + 5;
    while (i < t) {
        r += to_string(rand() % 5000 - 1000);
        if (i != (t - 1)) {
            r += " ";
        }
        i++;
    }

    scalacz a(r);

```

```
31      int spaces = 0;
32      for (int j = 0; j < r.length(); j++) {
33          if (r[j] == ' ') {
34              spaces += 1;
35          }
36      }
37
38      int j = 0;
39      int* tab = a.pobierzTablice();
40      while (j < spaces - 1) {
41          if (tab[j] == tab[j + 1]) {
42              EXPECT_EQ(tab[j], tab[j + 1]);
43          }
44          else {
45              EXPECT_LT(tab[j], tab[j + 1]);
46          }
47          j += 1;
48      }
49  }
50
51  /// @brief Test4 sprawdza czy posortuje tablicŹ z ujemnymi
        elementami
52  TEST(Testyogolne, ujemne_elementy) {
53      string input = ("-3 -7 -22 -47 -102 -345 -23 -15 -50 -8 -25000
        -950");
54      scalacz a(input);
55      EXPECT_EQ("-25000 -950 -345 -102 -50 -47 -23 -22 -15 -8 -7 -3",
        a.zwracajTablice(input));
56  }
57
58  /// @brief Test5 sprawdza czy posortuje tablicŹ z ujemnymi i
        dodatnimi elementami
59  TEST(Testyogolne, elementy_dodatnie_i_ujemne) {
60      string input = ("9 3 12 6 8 5 10 -3 -1 -7 0");
61      scalacz a(input);
62      EXPECT_EQ("-7 -3 -1 0 3 5 6 8 9 10 12", a.zwracajTablice(input)
        );
63  }
64
65  /// @brief Test6 sprawdza czy wyskoczy b Ć d przy tablicy bez
        element w
66  TEST(Testyogolne, sortowanie_tablicy_bez_elemenow) {
67      string input = ("");
68      scalacz a(input);
69      EXPECT_EQ("", a.zwracajTablice(input));
```

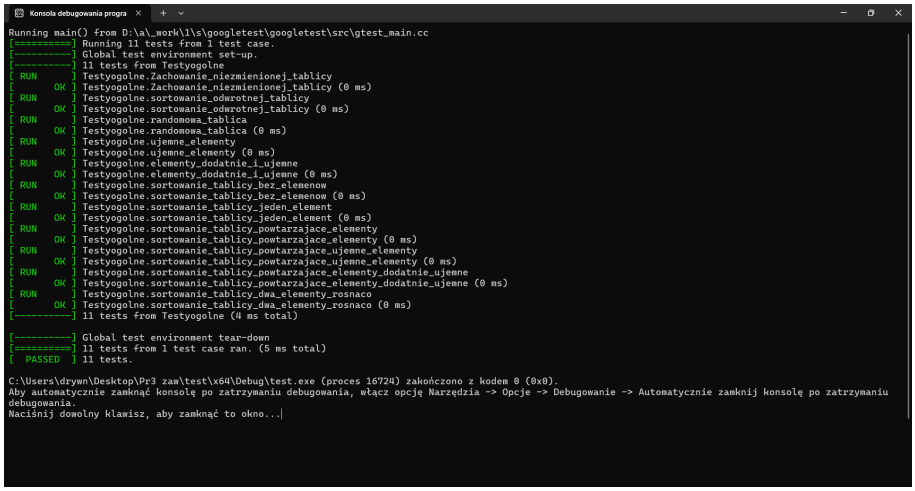
```
70 }
71
72 /// @brief Test7 sprawdza czy posortuje tablicŹ z jednym elementem
73 TEST(Testyogolne, sortowanie_tablicy_jeden_element) {
74     string input = ("7");
75     scalacz a(input);
76     EXPECT_EQ("7", a.zwracajTablice(input));
77 }
78
79 /// @brief Test8 sprawdza czy posortuje tablicŹ z powtarzaj cymi
       si Ź elementami
80 TEST(Testyogolne, sortowanie_tablicy_powtarzajace_elementy) {
81     string input = ("4 4 4 6 6 6 5 5 5 5 5 30 18 18 18");
82     scalacz a(input);
83     EXPECT_EQ("4 4 4 5 5 5 5 5 6 6 6 18 18 18 30", a.zwracajTablice
       (input));
84 }
85
86 /// @brief Test9 sprawdza czy posortuje tablicŹ z powtarzaj cymi
       si Ź ujemnymi elementami
87 TEST(Testyogolne, sortowanie_tablicy_powtarzajace_ujemne_elementy)
       {
88     string input = ("-4 -4 -4 -6 -6 -6 -5 -5 -5 -5 -5 -30 -18 -18
       -18");
89     scalacz a(input);
90     EXPECT_EQ("-30 -18 -18 -18 -6 -6 -6 -5 -5 -5 -5 -5 -4 -4 -4", a
       .zwracajTablice(input));
91 }
92
93 /// @brief Test10 sprawdza czy posortuje tablicŹ z powtarzaj cymi
        si Ź ujemnymi i dodatnimi elementami
94 TEST(Testyogolne,
       sortowanie_tablicy_powtarzajace_elementy_dodatnie_ujemne) {
95     string input = ("-2 -2 2 4 -4 -4 3 3 -3 -3 3 20 15 -15 15");
96     scalacz a(input);
97     EXPECT_EQ("-15 -4 -4 -3 -3 -2 -2 2 3 3 3 4 15 15 20", a.
       zwracajTablice(input));
98 }
99
100 /// @brief Test11 sprawdza czy posortuje tablicŹ z dwoma
       elementami rosn co
101 TEST(Testyogolne, sortowanie_tablicy_dwa_elementy_rosnaco) {
102     string input = ("5 9");
103     scalacz a(input);
104     EXPECT_EQ("5 9", a.zwracajTablice(input));
```

```
105 }
```

Listing 14. Google tests

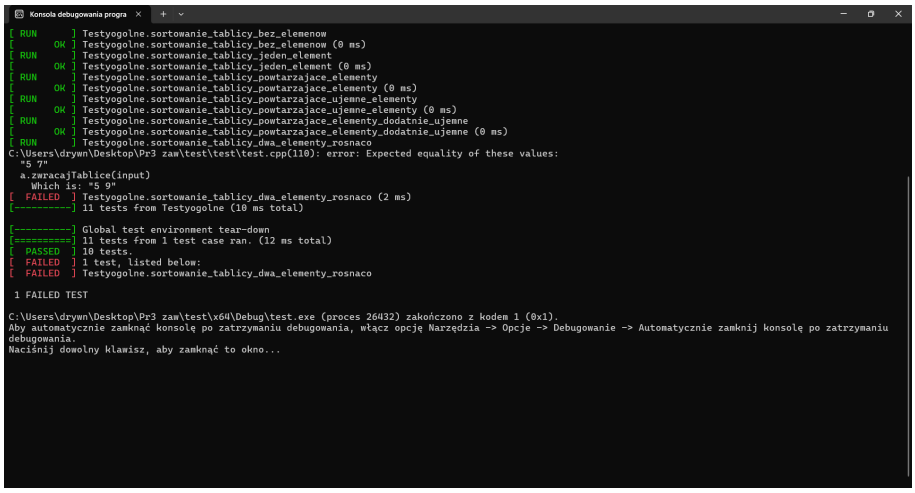
Wynik działania testów:

Prawidłowy: (Rys:4.1)



Rys. 4.1. Wynik prawidłowy

Nieprawdiłowy: (Rys:4.2)



Rys. 4.2. Wynik nie prawidłowy

5. Wnioski

Algorytm Merge Sort został pomyślnie zaimplementowany, a jego działanie spełnia oczekiwania. Praca nad implementacją wymagała zrozumienia zasad działania algorytmu dziel i zwyciężaj, co przyczyniło się do rozwinięcia umiejętności programowania w języku C++ oraz pogłębienia wiedzy o algorytmach sortujących. Rozwiązywanie problemów podczas kodowania pozwoliło na doskonalenie umiejętności analitycznego myślenia i debugowania kodu, szczególnie przy analizie działania rekurencji i łączenia posortowanych podtablic.

Narzędzie Google Test (GTest) okazało się niezwykle przydatne w procesie testowania zaimplementowanego algorytmu. Pozwoliło ono na weryfikację poprawności działania kluczowych funkcji Merge Sorta poprzez testy jednostkowe. Testowanie przy użyciu GTest zwiększyło pewność co do poprawności działania implementacji i dało praktyczne doświadczenie w pracy z narzędziami do testowania, które są istotne w pracy zawodowej.

Bibliografia

[1] *Strona internetowa magre sort*. URL: https://www.geeksforgeeks.org/merge-sort/.

[2] *Strona Repozytorium projektu*. URL: https://github.com/Drywno/Projekt_3.

[3] *Strona internetowa Github*. URL: https://github.com/.

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

[5] *Strona internetowa Visual Studio 2022*. URL: https://learn.microsoft.com/pl-pl/visualstudio/install/install-visual-studio?view=vs-2022.

Spis rysunków

Spis listingów