,,,,

AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Sortowanie przez scalanie oraz wykorzystanie google test

Autor:
Doktor Dawid

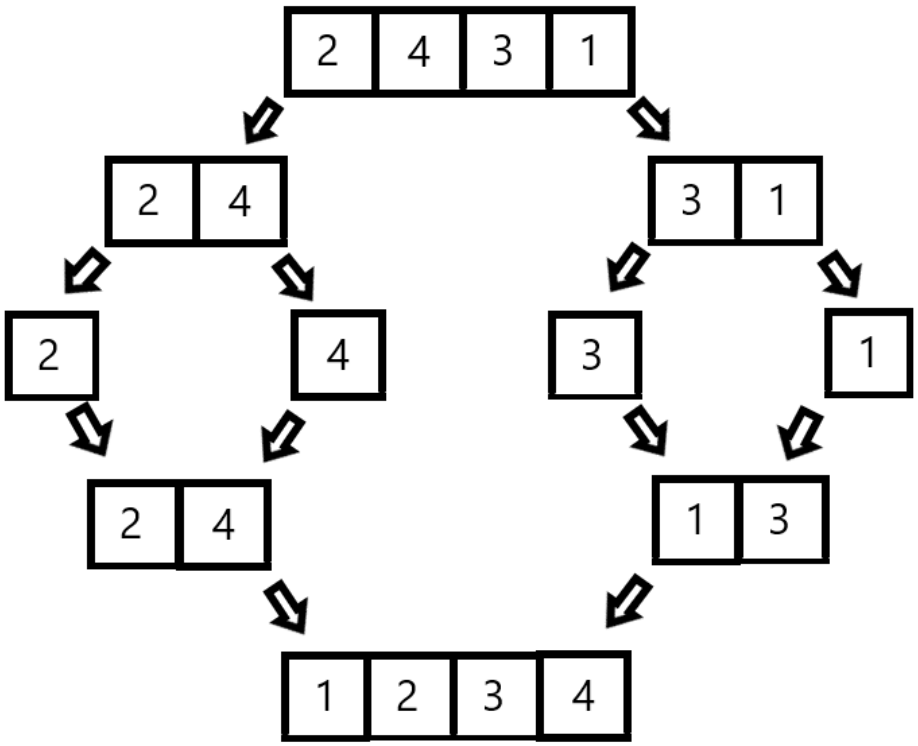
Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Głównym zadaniem projektu jest stworzenie kodu robiącego sortowanie poprzez scalanie. Sortowanie przez scalanie to algorytm dzielący tablicę na coraz mniejsze części, aż do pojedynczych elementów, a następnie łączący je w posortowaną całość. Działa na zasadzie dziel i zwyciężaj, z gwarantowaną złożonością O(nlogn).



Rys. 1.1. Przykład sortowania przez scalanie

Testowanie za pomocą Google Test w Visual Studio polega na weryfikacji poprawności działania algorytmów poprzez tworzenie zestawu testów, które sprawdzają różne scenariusze działania kodu. Google Test umożliwia pisanie testów w języku C++ i zapewnia funkcje do porównywania wyników oczekiwanych z uzyskanymi. Testy są organizowane w zestawy i uruchamiane w sposób automatyczny, a wyniki prezentowane w formie raportu, wskazując, które przypadki przeszły, a które nie. Dzięki temu można szybko zidentyfikować błędy w kodzie i upewnić się, że zmiany wprowadzone do algorytmu nie naruszają jego wcześniejszej funkcjonalności.

2. Analiza problemu

2.1. Czym jest sortowanie przez scalanie

Sortowanie przez scalanie to algorytm sortowania oparty na zasadzie dziel i zwyciężaj. Polega na podziale zbioru danych na coraz mniejsze części, aż do pojedynczych elementów, które są następnie scalane w większe, już posortowane fragmenty. Proces kończy się po połączeniu wszystkich elementów w jedną, posortowaną całość. Algorytm ma złożoność O(nlogn) i jest stabilny, ale wymaga dodatkowej pamięci na czas scalania.

2.2. Zastosowania sortowania przez scalanie

Sortowanie przez scalanie to algorytm sortowania oparty na zasadzie dziel i zwyciężaj. Polega na podziale zbioru danych na coraz mniejsze części, aż do pojedynczych elementów, które są następnie scalane w większe, już posortowane fragmenty. Proces kończy się po połączeniu wszystkich elementów w jedną, posortowaną całość. Algorytm ma złożoność O(nlogn) i jest stabilny, ale wymaga dodatkowej pamięci na czas scalania. Zastosowania sortowania przez scalanie obejmują sortowanie dużych zbiorów danych, porządkowanie danych na zewnętrznych nośnikach, scalanie i sortowanie strumieni danych w czasie rzeczywistym oraz algorytmy przetwarzania równoległego.

2.3. Działanie list dwukierunkowych

Sortowanie przez scalanie działa w trzech krokach:
-Podział: Zbiór [38,27,43,3,9,82,10] [38,27,43,3,9,82,10] dzielimy na dwie części: [38,27,43][38,27,43] i [3,9,82,10] [3,9,82,10], a następnie każdą z nich dzielimy dalej, aż uzyskamy pojedyncze elementy.
-Scalanie: Najpierw scalamy [27][27] i [43][43] w [27,43][27,43], [3][3] i [9][9] w [3,9][3,9], [82][82] i [10][10] w [10,82][10,82], a następnie scalamy większe części: [38][38] i [27,43][27,43] w [27,38,43][27,38,43], [3,9][3,9] i [10,82][10,82] w [3,9,10,82][3,9,10,82].
-Połączenie: Na końcu scalamy [27,38,43][27,38,43] i [3,9,10,82][3,9,10,82] w [3,9,10,27,38,43,82] [3,9,10,27,38,43,82], uzyskując posortowany zbiór.

2.4. Czym jest Google test

Google Test to biblioteka do pisania testów jednostkowych w C++, która umożliwia sprawdzanie poprawności kodu za pomocą asercji i organizowania testów w zestawy. Jest szeroko stosowana w projektach do automatycznego wykrywania błędów i regresji.

2.5. Zastosowania Google test

Google Test jest używane do testowania jednostkowego, automatycznego wykrywania błędów, weryfikowania algorytmów w różnych scenariuszach, testowania interfejsów API, mierzenia wydajności funkcji, a także do integracji z procesem CI/CD, co zapewnia wysoką jakość kodu w projekcie. Pomaga w szybkim wychwytywaniu regresji, poprawie jakości kodu oraz zapewnia stabilność systemu w miarę rozwoju projektu.

2.6. Działanie Google test

Google Test umożliwia pisanie testów jednostkowych, które sprawdzają poprawność funkcji. Testy są organizowane w zestawy, a asercje porównują wyniki rzeczywiste z oczekiwanymi. Testy są uruchamiane automatycznie, a wyniki raportowane, wskazując, które testy zakończyły się sukcesem, a które nie. Dzięki temu Google Test wspiera zapewnienie jakości i stabilności kodu.

3. Projektowanie

3.1. C++

3.1.1. Czym jest C++

Rys. 3.1. Logo C++

C++ jest językiem programistycznym, który jest rozwinięciem języka C. Dodaje on wiele zaawansowanych funkcji oraz wiele nowych mechanizmów dla ułatwienia prazy oraz skuteczności.

3.1.2. Zastosowania

Główne zastosowania C++ obejmują:
-Systemy operacyjne – rdzenie systemów, takie jak Windows, korzystają z C++ ze względu na jego wydajność.
-Gry komputerowe – wysoka szybkość i kontrola nad pamięcią sprawiają, że C++ jest popularny w przemyśle gier.
-Aplikacje o dużej wydajności – np. silniki baz danych, systemy handlu na giełdach.
-Oprogramowanie wbudowane – np. oprogramowanie do urządzeń IoT
-Symulacje i analizy naukowe – stosowany w aplikacjach wymagających intensywnych obliczeń

3.1.3. Działanie

C++ działa w ten sposób, że piszesz kod, który następnie kompilator przekształca w program wykonywalny, czyli plik, który komputer może uruchomić. Dzięki temu C++ jest szybki i wydajny. Możesz w nim tworzyć zarówno proste funkcje, jak i złożone struktury (jak klasy i obiekty), które pomagają organizować kod. Programista ma pełną kontrolę nad pamięcią i zasobami, co jest ważne przy tworzeniu wydajnych aplikacji.

3.2. Visual Studio

Rys. 3.2. Logo Visual Studio

3.2.1. Czym jest Visual Studio

Visual Studio to zaawansowane środowisko programistyczne (IDE) stworzone przez Microsoft, które umożliwia pisanie, debugowanie i testowanie kodu w różnych językach, takich jak C++, C Sharp, Python, czy JavaScript. Visual Studio oferuje funkcje ułatwiające pracę, jak autouzupełnianie kodu, zarządzanie projektami, ma również wbudowany debugger i integrację z systemami kontroli wersji.

3.2.2. Zastosowania Visual Studio

Visual Studio służy do tworzenia aplikacji desktopowych, mobilnych, internetowych oraz gier. Ułatwia pisanie i testowanie kodu, wspiera programowanie zespołowe i zarządzanie projektami. Jest często używany do tworzenia aplikacji dla systemu Windows, ale wspiera też inne platformy, np. Android, iOS oraz aplikacje chmurowe.

3.2.3. Spsób działania Visual Studio

Visual Studio działa jako zintegrowane środowisko, w którym można pisać, kompilować, testować i debugować kod. Programista tworzy projekt, wybiera język programowania, a Visual Studio automatycznie organizuje pliki i foldery projektu. Podczas pisania kodu IDE oferuje podpowiedzi i autouzupełnianie, co przyspiesza pracę. Kiedy kod jest gotowy, Visual Studio korzysta z wbudowanego kompilatora, by przekształcić go w program wykonywalny. W razie błędów umożliwia ich szybkie wykrycie i poprawienie dzięki wbudowanemu debuggerowi.

3.3. Git

Rys. 3.3. Logo Git

3.3.1. Czym jest Git

Git to system kontroli wersji, który umożliwia śledzenie zmian w kodzie źródłowym i współpracę nad projektami. Dzięki Gitowi programiści mogą tworzyć wersje kodu, co pozwala na łatwe przywracanie wcześniejszych zmian, zarządzanie różnymi wersjami projektu oraz pracę zespołową, w której każdy może niezależnie wprowadzać modyfikacje, a potem je łączyć.

3.3.2. Zastosowania Git

Git jest systemem kontroli wersji, który pozwala na śledzenie zmian w plikach i zarządzanie historią wersji w projektach programistycznych. Dzięki niemu programiści mogą pracować nad kodem w sposób zorganizowany, tworzyć różne gałęzie (branches) projektu, eksperymentować z nowymi funkcjami, a potem łączyć zmiany. Git jest szeroko stosowany do pracy zespołowej, umożliwiając wielu osobom równoczesną edycję kodu i integrację ich zmian, bez ryzyka nadpisania pracy innych. Platformy takie jak GitHub, GitLab czy Bitbucket opierają się na Gicie, oferując dodatkowe funkcje, takie jak hosting repozytoriów i współpracę online.

3.3.3. Sposób działania Git

Git działa poprzez tworzenie "repozytorium"— bazy, która przechowuje pełną historię zmian w projekcie. Każda zmiana w projekcie jest zapisywana jako "commit", który zawiera informacje o modyfikacjach i autorze. Git pozwala na tworzenie różnych gałęzi (branches), dzięki czemu programiści mogą pracować nad różnymi funkcjami niezależnie, a później łączyć (merge) te zmiany w główną wersję projektu.
Podstawowe operacje w Gicie to:
-Clone — sklonowanie repozytorium zdalnego na lokalny komputer.
-Commit — zapisanie zmian w lokalnym repozytorium.
-Push — wysłanie lokalnych zmian do zdalnego repozytorium.
-Pull — pobranie najnowszych zmian z repozytorium zdalnego.
-Branch — tworzenie gałęzi, które pozwalają pracować nad różnymi funkcjami równolegle.

3.4. Doxygen

3.5. Git

Rys. 3.4. Logo Doxygen

3.5.1. Czym jest Doxygen

Doxygen to narzędzie do generowania dokumentacji z komentarzy zawartych w kodzie źródłowym. Umożliwia automatyczne tworzenie dokumentacji w różnych formatach (HTML, PDF, LaTeX itp.) na podstawie specjalnie sformatowanych komentarzy, które programista dodaje do kodu. Doxygen wspiera wiele języków programowania, w tym C++, C, Java, Python i inne, i jest często wykorzystywane w projektach o dużej skali, gdzie ważne jest utrzymanie aktualnej i przejrzystej dokumentacji.

3.5.2. Zastosowanie Doxygen'a

Doxygen jest narzędziem do automatycznego generowania dokumentacji z komentarzy zawartych w kodzie źródłowym. Umożliwia programistom tworzenie szczegółowych, łatwych do zrozumienia dokumentów technicznych, które opisują struktury danych, funkcje, klasy, metody i interfejsy w projekcie. Doxygen przetwarza specjalnie sformatowane komentarze w kodzie (np. w języku C++, C, Java, Python) i na ich podstawie tworzy dokumentację w różnych formatach, takich jak HTML, PDF, LaTeX, RTF czy man page.
Jest szczególnie przydatne w dużych i złożonych projektach, gdzie manualne pisanie i aktualizowanie dokumentacji byłoby czasochłonne i łatwe do pominięcia. Dzięki Doxygen programiści mogą utrzymywać dokumentację na bieżąco, minimalizując ryzyko niezgodności między kodem a opisami. Narzędzie jest szeroko stosowane w inżynierii oprogramowania, tworzeniu bibliotek, API, a także w projektach open-source, gdzie współpracuje wiele osób.

3.5.3. Sposób działania Doxygen'a

Doxygen działa, analizując specjalne komentarze w kodzie źródłowym i na ich podstawie generuje dokumentację. Programista dodaje w kodzie opisujące komentarze (np. o funkcjach, klasach, parametrach), a Doxygen przetwarza je na dokumenty w formatach takich jak HTML, PDF czy LaTeX. Dzięki temu dokumentacja jest zawsze aktualna i łatwa do uzyskania, bez potrzeby ręcznego pisania opisów.

4. Implementacja

4.1. Listing kodów

```
1  #pragma once
2
3  #include "gtest/gtest.h"
4  #include <string>
5  #include <iostream>
6  #include <algorithm>
7
8  using namespace std;
9  /// @brief Klasa scalanie oraz deklaracje jej metod
10 class scalanie {
11 private:
12   string* tab;///< wska nik do Zmiennej zawieraj cej
      nieposortowanieowan  tablic Ź typu string
13   int* tabs;///< wska nik do tablicy zawieraj cej
      posortowanieowan  tablic Ź tab
14 public:
15
16   scalanie(string a);
17   ~scalanie();
18   int sprawdzanie(int i, string a);
19   int dlugosc(string a);
20   void convert(int a);
21   void sortowanie(int a);
22   string zwrottabs(string a);
23   int* wsaztabs();
24 };
```

Listing 1. Kod pch.h

Kod zawiera w sobie klasę scalanie znajdującą się w 10 wierszu oraz jej metody zaczynające się od 16 wiersza.

```
1
2  #include "pch.h"
3  #include <iostream>
4  #include <algorithm>
5
6  using namespace std;
7  /// @brief konstruktor, automatycznie sortuje tablicŹ
8  /// @param a - tablica do posortowania
9  scalanie::scalanie(string a) {
10     if (a != "") {//je eli string == "" to przechodzi do else w
       kt rym ustawia tablicŹ na 0
11         int i = 0;
12         int tabtab = 0;
13         int spaces = dlugosc(a);
14         tab = new string[spaces + 1];
15         tabs = new int[spaces + 1];
16
17         while (i < a.length()) {
18             if (a[i] == ' ') {
19                 i += 1;
20                 continue;
21             }
22
23             int wordLength = sprawdzanie(i, a);
24
25             tab[tabtab] = a.substr(i, wordLength);
26             tabtab += 1;
27
28             i += wordLength;
29         }
30         convert(spaces);
31         sortowanie(spaces);
32     }
33     else {
34         tab = new string[1];
35         tab[0] = "";
36         tabs = new int[0];
37     }
38
39 }
40 /// @brief Destruktor, usuwa zmienne dynamiczne tab i tabs
41 scalanie::~scalanie() {
42     delete[] tab;
43     delete[] tabs;
44 };
```

```
45
46 /// @brief metoda sprawdzanie, zwraca d Ćugo Ż  danego elementu
       tablicy (np. 3 element to 25 czyli ma d Ćugo Ż  2)
47 /// @param i - miejece elementu w string a
48 /// @param a - nieposortowanieowana tablica tab
49 int scalanie::sprawdzanie(int i, string a) {
50     int length = 0;
51     while (i + length < a.length() && a[i + length] != ' ') {
52         length += 1;
53     }
54     return length;
55 }
56
57 /// @brief metoda dlugosc zwraca ilo Ż  spacji w stringu
       powi Źkszonej o 1
58 /// @param a - nieposortowanieowana tablica tab
59 int scalanie::dlugosc(string a) {
60     int spaces = 0;
61     for (int j = 0; j < a.length(); j++) {
62         if (a[j] == ' ') {
63             spaces += 1;
64         }
65     }
66     return spaces + 1;
67 }
68 /// @brief metoda zwrottab, konwertuje string tab do tablicy int
       tabs
69 /// @param spaces - ilo Ż  element w tablicy tabs
70 void scalanie::convert(int spaces) {
71     int i = 0;
72     while (i < (spaces)) {
73         tabs[i] = stoi(tab[i]);
74         i++;
75     }
76 }
77 /// @brief metoda sortowanie, sortuje tablic Ź tabs
78 /// @param spaces - ilo Ż  element w tablicy tabs
79 void scalanie::sortowanie(int spaces) {
80     int mnoznik = 1;
81
82     while (mnoznik < spaces) {
83         for (int i = 0; i < spaces; i += (2 * mnoznik)) {
84             int mid = min(i + mnoznik, spaces);
85             int end = min(i + 2 * mnoznik, spaces);
86
```

```
87
88              int* temp = new int[spaces];
89              int left = i, right = mid, k = i;
90
91
92              while (left < mid && right < end) {
93                  if (tabs[left] <= tabs[right]) {
94                      temp[k++] = tabs[left++];
95                  }
96                  else {
97                      temp[k++] = tabs[right++];
98                  }
99              }
100
101
102             while (left < mid) {
103                 temp[k++] = tabs[left++];
104             }
105
106
107             while (right < end) {
108                 temp[k++] = tabs[right++];
109             }
110
111
112             for (int j = i; j < end; j++) {
113                 tabs[j] = temp[j];
114             }
115
116             delete[] temp;
117         }
118
119         mnoznik *= 2;
120     }
121 }
122
123
124
125 /// @brief metoda zwrottabs,zwraca tablicŹ w stringu
126 /// @param a - nieposortowanieowana tablica tab
127 string scalanie::zwrottabs(string a) {
128     if (tab[0] == "") {
129         return "";
130     }
131     else {
```

```
132         int i = 0;
133         string r;
134         while (i < dlugosc(a)) {
135             r += to_string(tabs[i]);
136             if (i != (dlugosc(a) - 1)) {
137                 r += " ";
138             }
139             i++;
140         }
141         return  r;
142     }
143 }
144 /// @brief metoda wsaztabs, zwraca wska nik na tabs
145 int* scalanie::wsaztabs() {
146     return tabs;
147 }
```

Listing 2. Kod pch.cpp

W tym kodzie wywołane są wszystkie metody klasy scalanie.

```
1  /// @file test.cpp Plik g Ć wny
2
3  #include "pch.h"
4  #include <cstdlib>
5  #include <ctime>
6
7
8  /// @brief Test1, sprawdza czy tablica zostanie niezmieniona
       je eli wpiszemy ju  posortowanieowan  tablic Ź
9  TEST(Testyogolne, Zachowanie_niezmienionej_tablicy) {
10   string input = ("1 2 3 4 5 6 7");
11   scalanie a(input);
12   EXPECT_EQ(input, a.zwrottabs(input));
13 }
14 /// @brief Test2, Sprawdza czy dobrze posortuje odwr con
       tablic Ź
15 TEST(Testyogolne, sortowanieowanie_odwrotnej_tablicy) {
16   string input = ("10 9 8 7 6 5 4 3 2 1 0");
17   scalanie a(input);
18   EXPECT_EQ("0 1 2 3 4 5 6 7 8 9 10", a.zwrottabs(input));
19 }
20 /// @brief Test3, sprawdza czy posortuje randomow  tablic Ź
21 TEST(Testyogolne, randomowa_tablica) {
22   srand(static_cast<unsigned int>(time(0)));
23   int i = 0;
24   string r;
25   int t = rand() % 100 + 1;
26   while (i < t) {
27     r += to_string(rand() % 10000 - 5000);
28     if (i != (t - 1)) {
29       r += " ";
30     }
31     i++;
32   }
33
34   scalanie a(r);
35
36   int spaces = 0;
37   for (int j = 0; j < r.length(); j++) {
38     if (r[j] == ' ') {
39       spaces += 1;
40     }
41   }
42
43
```

```
44    int j = 0;
45    int* tab = a.wsaztabs();
46    while (j < spaces - 1) {
47      if (tab[j] == tab[j + 1]) {
48        EXPECT_EQ(tab[j], tab[j + 1]);
49      }
50      else {
51        EXPECT_LT(tab[j], tab[j + 1]);
52      }
53      j += 1;
54    }
55  }
56  /// @brief Test4 sprawdza czy posortuje tablicŹ z ujemnymi
        elementami
57  TEST(Testyogolne, ujemne_elementy) {
58    string input = ("-1, -2 -54 -67 -213 -678 -12 -9 -78 -3 -123342
        -1456");
59    scalanie a(input);
60    EXPECT_EQ("-123342 -1456 -678 -213 -78 -67 -54 -12 -9 -3 -2 -1",
       a.zwrottabs(input));
61  }
62  /// @brief Test5 sprawdza czy posortuje tablicŹ z ujemnymi i
        dodatnimi elementami
63  TEST(Testyogolne, elementy_dodatnie_i_ujemne) {
64    string input = ("5 1 7 2 4 3 6 8 0 -2 -1");
65    scalanie a(input);
66    EXPECT_EQ("-2 -1 0 1 2 3 4 5 6 7 8", a.zwrottabs(input));
67  }
68  /// @brief Test6 sprawdza czy wyskoczy b Ć d przy tablicy bez
        element w
69  TEST(Testyogolne, sortowanieowanie_tablicy_bez_elemenow) {
70    string input = ("");
71    scalanie a(input);
72    EXPECT_EQ("", a.zwrottabs(input));
73  }
74  /// @brief Test7 sprawdza czy posortuje tablicŹ z jednym elementem
75  TEST(Testyogolne, sortowanieowanie_tablicy_jeden_element) {
76    string input = ("1");
77    scalanie a(input);
78    EXPECT_EQ("1", a.zwrottabs(input));
79  }
80  /// @brief Test8 sprawdza czy posortuje tablicŹ z powtarzaj cymi
        si Ź elementami
81  TEST(Testyogolne, sortowanieowanie_tablicy_powtarzajace_elementy) {
82    string input = ("1 1 1 3 3 3 2 2 2 2 2 25 12 12 12");
```

```
83    scalanie a(input);
84    EXPECT_EQ("1 1 1 2 2 2 2 2 3 3 3 12 12 12 25", a.zwrottabs(input)
       );
85  }
86  /// @brief Test9 sprawdza czy posortuje tablicŹ z powtarzajcymi
       siŹ ujemnymi elementami
87  TEST(Testyogolne,
       sortowanieowanie_tablicy_powtarzajace_ujemne_elementy) {
88    string input = ("-1 -1 -1 -3 -3 -3 -2 -2 -2 -2 -2 -25 -12 -12 -12
       ");
89    scalanie a(input);
90    EXPECT_EQ("-25 -12 -12 -12 -3 -3 -3 -2 -2 -2 -2 -2 -1 -1 -1", a.
       zwrottabs(input));
91  }
92  /// @brief Test10 sprawdza czy posortuje tablicŹ z powtarzajcymi
       siŹ ujemnymi i dodatnimi elementami
93  TEST(Testyogolne,
       sortowanieowanie_tablicy_powtarzajace_elementy_dodatnie_ujemne)
       {
94    string input = ("-1 -1 1 3 -3 -3 2 2 -2 -2 2 25 12 -12 12");
95    scalanie a(input);
96    EXPECT_EQ("-12 -3 -3 -2 -2 -1 -1 1 2 2 2 3 12 12 25", a.zwrottabs
       (input));
97  }
98  /// @brief Test11 sprawdza czy posortuje tablicŹ z dwoma
       elementami rosnco
99  TEST(Testyogolne, sortowanieowanie_tablicy_dwa_elementy_rosnaco) {
100   string input = ("1 3");
101   scalanie a(input);
102   EXPECT_EQ("1 3", a.zwrottabs(input));
103 }
104 /// @brief Test12 sprawdza czy posortuje tablicŹ wiŹksz  ni
       100
105 TEST(Testyogolne, randomowa_tablica_wiŹksza_niz_100){
106   srand(static_cast<unsigned int>(time(0)));
107   int i = 0;
108   string r;
109   int t = rand() % 500 + 100;
110   while (i < t) {
111     r += to_string(rand() % 10000 - 2000);
112     if (i != (t - 1)) {
113       r += " ";
114     }
115     i++;
116   }
```

```
117
118   scalanie a(r);
119
120   int spaces = 0;
121   for (int j = 0; j < r.length(); j++) {
122     if (r[j] == ' ') {
123       spaces += 1;
124     }
125   }
126
127
128   int j = 0;
129   int* tab = a.wsaztabs();
130   while (j < spaces - 1) {
131     if (tab[j] == tab[j + 1]) {
132       EXPECT_EQ(tab[j], tab[j + 1]);
133     }
134     else {
135       EXPECT_LT(tab[j], tab[j + 1]);
136     }
137     j += 1;
138   }
139 }
140 /// @brief Test13 sprawdza czy posortuje tablicŹ wi Źksz  ni
      100 z ujemnymi, dodatnimi elementami i duplikatami
141 TEST(Testyogolne, randomowa_tablica_wieksza_niz_100_dod_uj_dup) {
142   srand(static_cast<unsigned int>(time(0)));
143   int i = 0;
144   string r;
145   int t = rand() % 500 + 100;
146   while (i < t) {
147     int y = rand() % 4000 - 2001;
148     r += to_string(y);
149     r += " ";
150     r += to_string(y);
151     if (i != (t - 1)) {
152       r += " ";
153     }
154     i++;
155   }
156
157   scalanie a(r);
158
159   int spaces = 0;
160   for (int j = 0; j < r.length(); j++) {
```

```
161     if (r[j] == ' ') {
162       spaces += 1;
163     }
164   }
165
166
167   int j = 0;
168   int* tab = a.wsaztabs();
169   while (j < spaces - 1) {
170     if (tab[j] == tab[j + 1]) {
171       EXPECT_EQ(tab[j], tab[j + 1]);
172     }
173     else {
174       EXPECT_LT(tab[j], tab[j + 1]);
175     }
176     j += 1;
177   }
178 }
```

Listing 3. Kod Test Google

W tym kodzie google test sprawdza działanie kodu sortowanie przez scalanie i ocenia czy dziala on poprawnie.

4.2. Wynik działania kodu

```
[ RUN      ] Testyogolne.sortowanieowanie_odwrotnej_tablicy
[       OK ] Testyogolne.sortowanieowanie_odwrotnej_tablicy (0 ms)
[ RUN      ] Testyogolne.randomowa_tablica
[       OK ] Testyogolne.randomowa_tablica (0 ms)
[ RUN      ] Testyogolne.ujemne_elementy
[       OK ] Testyogolne.ujemne_elementy (0 ms)
[ RUN      ] Testyogolne.elementy_dodatnie_i_ujemne
[       OK ] Testyogolne.elementy_dodatnie_i_ujemne (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_bez_elemenow
[       OK ] Testyogolne.sortowanieowanie_tablicy_bez_elemenow (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_jeden_element
[       OK ] Testyogolne.sortowanieowanie_tablicy_jeden_element (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_elementy
[       OK ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_elementy (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_ujemne_elementy
[       OK ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_ujemne_elementy (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_elementy_dodatnie_ujemne
[       OK ] Testyogolne.sortowanieowanie_tablicy_powtarzajace_elementy_dodatnie_ujemne (0 ms)
[ RUN      ] Testyogolne.sortowanieowanie_tablicy_dwa_elementy_rosnaco
[       OK ] Testyogolne.sortowanieowanie_tablicy_dwa_elementy_rosnaco (0 ms)
[ RUN      ] Testyogolne.randomowa_tablica_wi-Öksza_niz_100
[       OK ] Testyogolne.randomowa_tablica_wi-Öksza_niz_100 (0 ms)
[ RUN      ] Testyogolne.randomowa_tablica_wieksza_niz_100_dod_uj_dup
[       OK ] Testyogolne.randomowa_tablica_wieksza_niz_100_dod_uj_dup (1 ms)
[----------] 13 tests from Testyogolne (4 ms total)

[----------] Global test environment tear-down
[==========] 13 tests from 1 test case ran. (4 ms total)
[  PASSED  ] 13 tests.
```

Rys. 4.1. Działanie Google Test

Google test rozpoczyna testowanie kodu poprzez run, jeśli wynik jest poprawny przy wyniku pojawia się OK i na zakończeniu robi Tak jak przedstawione jest w zdjęciu wyżej test przeszedł poprawnie.

5. Wnioski

Sortowanie przez scalanie jest jednym z najwydajniejszych algorytmów sortowania o złożoności O(nlogn), szczególnie przydatnym w przypadku dużych zbiorów danych, gdzie inne algorytmy mogą być mniej efektywne. Jego stabilność, czyli zachowanie kolejności elementów o tej samej wartości, oraz wydajność sprawiają, że jest szeroko stosowany w wielu dziedzinach, takich jak przetwarzanie danych w bazach czy sortowanie plików na dyskach. Choć wymaga dodatkowej pamięci, jego zdolność do efektywnego przetwarzania danych i niezawodność czynią go solidnym wyborem w wielu zastosowaniach, gdzie czas i przestrzeń są kluczowe. Według mnie jest on przydany w użyciu i pomaga pominąć tworzenia osobnych obliczeń.

Google Test to potężne narzędzie do pisania testów jednostkowych, które pozwala programistom na weryfikację poprawności kodu i algorytmów, takich jak sortowanie przez scalanie, w sposób systematyczny i zautomatyzowany. Dzięki szerokiemu zestawowi asercji i możliwości organizowania testów w zestawy, Google Test ułatwia wykrywanie błędów, regresji i zapewnia wysoką jakość kodu. Jego integracja w procesie ciągłej integracji (CI) pozwala na szybkie i efektywne testowanie dużych projektów, zapewniając stabilność aplikacji podczas dalszego rozwoju. W połączeniu z algorytmami, takimi jak sortowanie przez scalanie, Google Test umożliwia pełną kontrolę nad jakością i poprawnością implementacji. Sądze że jest on bardzo pomocny w odnajdywaniu błedów w swoim działaniu kodu i przyspiesza prace mówiąc w której cześci wystąpił błąd.

Spis rysunków

Spis listingów