Merge Sort

3

Wygenerowano za pomocą Doxygen 1.12.0

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdujÄ siÄ klasy, struktury, unie i interfejsy wraz z ich krÃ^{3}tkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików wraz z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy MergeSort

Klasa implementuj¹ca algorytm MergeSort.

```
#include <sort.h>
```

Statyczne metody publiczne

- static void Sort (std::vector< int > &tab)

  Funkcja sortuj¹ca tablicê przy u¿yciu algorytmu MergeSort.
- static void Divide (std::vector< int > &tab, int l, int r)

  Funkcja dziel¹ca tablicê na dwie czêci do dalszego sortowania.
- static void Merge (std::vector< int > &tab, int l, int mid, int r)

  Funkcja scala dwie posortowane czêci tablicy w jedn¹ posortowan¹ ca³oœæ.

3.1.1 Opis szczegÃ³Åowy

Klasa implementuj¹ca algorytm MergeSort.

Definicja w linii 7 pliku sort.h.

3.1.2 Dokumentacja funkcji skÅadowych

3.1.2.1 Divide()

```
void MergeSort::Divide (
            std::vector< int > & tab,
            int l,
            int r)  [static]
```

Funkcja dziel¹ca tablicê na dwie czêci do dalszego sortowania.

Funkcja ta dzieli tablicê na dwie czêci na podstawie indeksów lewego (l) i prawego (r) granicy. Nastêpnie wywo³uje rekurencyjnie sortowanie dla obu czêci.

Parametry

tab	Referencja do tablicy, która ma zostaæ podzielona.
l	Indeks pocz¹tkowy lewej czêci tablicy.
r	Indeks koñcowy prawej czêci tablicy.

Definicja w linii 8 pliku Sort.cpp.

3.1.2.2 Merge()

```
void MergeSort::Merge (
            std::vector< int > & tab,
            int l,
            int mid,
            int r)  [static]
```

Funkcja scala dwie posortowane czêci tablicy w jedn¹ posortowan¹ ca^{3}oæ.

Ta funkcja ^{3}¹czy dwie posortowane czêci tablicy w jeden ci¹g w sposób wydajny, zachowuj¹c porz¹dek rosn¹cy. Operacja scalania jest kluczow¹ czêci¹ algorytmu MergeSort.

Parametry

tab	Referencja do tablicy, w której odbywa siê scalanie.
l	Indeks pocz¹tkowy lewej czêci tablicy.
mid	Indeks rodkowy, który dzieli tablicê na dwie czêci.
r	Indeks koñcowy prawej czêci tablicy.

Definicja w linii 17 pliku Sort.cpp.

3.1.2.3 Sort()

```
void MergeSort::Sort (
            std::vector< int > & tab)  [static]
```

Funkcja sortuj¹ca tablicê przy u¿yciu algorytmu MergeSort.

Ta funkcja jest wywo^{3}ywana w celu rozpoczêcia procesu sortowania tablicy. U¿ywa algorytmu MergeSort, który dzieli tablicê na mniejsze czêci i scala je w posortowan¹ ca^{3}oæ.

Parametry

tab	Referencja do tablicy, która ma zostaæ posortowana.

Definicja w linii 3 pliku Sort.cpp.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ^{3}w:

- MergeSort/sort.h
- MergeSort/Sort.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku MergeSort/main.cpp

```
#include <iostream>
#include "sort.h"
```

Funkcje

- void PrintVector (const std::vector< int > &tab)
- int main ()

4.1.1 Dokumentacja funkcji

4.1.1.1 main()

```
int main ()
```

Definicja w linii 10 pliku main.cpp.

4.1.1.2 PrintVector()

```
void PrintVector (
            const std::vector< int > & tab)
```

Definicja w linii 4 pliku main.cpp.

4.2 main.cpp

Idź do dokumentacji tego pliku.

```
00001 #include <iostream>
00002 #include "sort.h"
00003
00004 void PrintVector(const std::vector<int>& tab) { //<-- Funkcja wypisująca vektor
00005     for (int i = 0; i < tab.size(); ++i) {
00006         std::cout « tab[i] « " ";
00007     }
00008     std::cout « std::endl;
00009 }
00010 int main()
00011 {
00012     std::vector<int> test1 = { 38, 27, 43, 3, 9, 82, 10 };
00013     std::vector<int> test2 = { 13, 2, 33, 42, 5 };
00014
00015
00016     std::cout « "Test 1 - Przed sortowaniem: ";
00017     PrintVector(test1);
00018     MergeSort::Sort(test1);
00019     std::cout « "Test 1 - Po sortowaniu: ";
00020     PrintVector(test1);
00021     std::cout « "Test 2 - Przed sortowaniem: ";
00022     PrintVector(test2);
00023     MergeSort::Sort(test2);
00024     std::cout « "Test 2 - Po sortowaniu: ";
00025     PrintVector(test2);
00026
00027
00028 }
00029
```

4.3 Dokumentacja pliku MergeSort/Sort.cpp

```
#include "sort.h"
```

4.4 Sort.cpp

IdÅº do dokumentacji tego pliku.

```
00001 #include "sort.h"
00002
00003 void MergeSort::Sort(std::vector<int>& tab) {
00004     if (tab.size() <= 1) return; // <-- Jeli tablica ma jeden lub mniej elementów, nie ma potrzeby
      sortowaæ
00005     Divide(tab, 0, tab.size() - 1); // <-- Rozpoczynamy dzielenie tablicy od indeksu 0 do ostatniego
      elementu
00006 }
00007
00008 void MergeSort::Divide(std::vector<int>& tab, int l, int r) { // <-- Funkcja jest odpowiedzialana za
      dzielenie tablicy na mniejsze podtablice
00009     if (l >= r) return; // <-- Jeli lewy indeks jest wiêkszy lub równy prawemu, oznacza to, ¿e nie ma
      ju¿ co dzieliæ (tablica ma tylko jeden element)
00010
00011     int mid = l + (r - l) / 2; // <-- Obliczamy indeks rodkowy
00012     Divide(tab, l, mid);       // <-- Rekurencyjnie dzielimy lew¹ czêæ tablicy
00013     Divide(tab, mid + 1, r);   // <-- Rekurencyjnie dzielimy praw¹ czêæ tablicy
00014     Merge(tab, l, mid, r);     // <-- Scalanie obu posortowanych czêci tablicy
00015 }
00016
00017 void MergeSort::Merge(std::vector<int>& tab, int l, int mid, int r) {
00018     std::vector<int> left(tab.begin() + l, tab.begin() + mid + 1); // <-- Tworzenie tablicy która
      zawiera elementy lewa do rodka
00019     std::vector<int> right(tab.begin() + mid + 1, tab.begin() + r + 1);// <-- Tworzenie tablicy która
      zawiera elementy rodka do koñca
00020
00021     int i = 0; // <-- indeks dla lewej podtablicy
00022     int j = 0; // <-- indeks dla lewej prawej
00023     int k = l; // <-- indeks dla lewej g3ównej
00024
```

```
00025     while (i < left.size() && j < right.size()) {// <-- Scalanie obu podtablic to tablicy g3ównej
      dopóki s1 elementy w obu podtablicach
00026         if (left[i] <= right[j]) {// <-- Jeli element w lewej podtablicy jest mniejszy lub równy
      elementowi w prawej
00027             tab[k++] = left[i++];// <-- Wstawiamy element z lewej podtablicy do g3ównej tablicy
00028         }
00029         else {
00030             tab[k++] = right[j++]; // <-- Wstawiamy element z prawej podtablicy do g3ównej tablicy
00031         }
00032     }
00033
00034     while (i < left.size()) {// <-- Je¿eli pozosta3y elementy w lewej podtablicy, wstawiamy je do
      g3ównej tablicy
00035         tab[k++] = left[i++]; // <-- Przenosimy pozosta3e elementy z lewej podtablicy
00036     }
00037
00038     while (j < right.size()) {// <-- Jeli pozosta3y elementy w prawej podtablicy, wstawiamy je do
      g3ównej tablicy
00039         tab[k++] = right[j++]; // <-- Przenosimy pozosta3e elementy z prawej podtablicy
00040     }
00041 }
```

4.5 Dokumentacja pliku MergeSort/sort.h

```
#include <vector>
```

Komponenty

- class MergeSort

  Klasa implementuj1ca algorytm MergeSort.

4.6 sort.h

Idź do dokumentacji tego pliku.

```
00001 #pragma once
00002 #include <vector>
00007 class MergeSort {
00008 public:
00017     static void Sort(std::vector<int>& tab);
00028     static void Divide(std::vector<int>& tab, int l, int r);
00040     static void Merge(std::vector<int>& tab, int l, int mid, int r);
00041 };
```

4.7 Dokumentacja pliku Test1/test.cpp

```
#include "../MergeSort/Sort.cpp"
#include "../MergeSort/sort.h"
#include <gtest/gtest.h>
```

Funkcje

- TEST (Tests, alreadydone)

  Test sortowania dla tablicy juz posortowanej.
- TEST (Tests, reverseOrder)

  Test sortowania dla tablicy juz posortowanej odwrotnie.
- TEST (Tests, randomOrder)

  Test sortowania dla tablicy losowej.
- TEST (Tests, negnumb)

  Test sortowania dla tablicy z negetywnymi numerami.
- TEST (Tests, negposnumb)

  Test sortowania dla tablicy z mieszanymi liczbami.
- TEST (Tests, empty)

  Test sortowania dla tablicy pustej.
- TEST (Tests, onenumb)

  Test sortowania dla tablicy jednoliczbowej.
- TEST (Tests, duplicates)

  Test sortowania dla tablicy z powtarzajacymi sie numerami.
- TEST (Tests, negativeDuplicates)

  Test sortowania dla tablicy z powtarzajacymi sie numerami ale negatynej.
- TEST (Tests, twonumb)

  Test sortowania dla tablicy z dwoma numerami.
- TEST (Tests, hundred)

  Test sortowania dla bardzo du¿ej tablicy .
- TEST (Tests, hundredmix)

  Test sortowania dla bardzo du¿ej tablicy a ujemnymi liczbami i duplikatami.

4.7.1 Dokumentacja funkcji

4.7.1.1 TEST() [1/12]

```
TEST (
            Tests ,
            alreadydone )
```

Test sortowania dla tablicy juz posortowanej.

Definicja w linii 5 pliku test.cpp.

4.7.1.2 TEST() [2/12]

```
TEST (
            Tests ,
            duplicates )
```

Test sortowania dla tablicy z powtarzajacymi sie numerami.

Definicja w linii 54 pliku test.cpp.

4.7.1.3 TEST() [3/12]

```
TEST (
            Tests ,
            empty )
```

Test sortowania dla tablicy pustej.

Definicja w linii 40 pliku test.cpp.

4.7.1.4 TEST() [4/12]

```
TEST (
            Tests ,
            hundred )
```

Test sortowania dla bardzo du¿ej tablicy .

Definicja w linii 75 pliku test.cpp.

4.7.1.5 TEST() [5/12]

```
TEST (
            Tests ,
            hundredmix )
```

Test sortowania dla bardzo du¿ej tablicy a ujemnymi liczbami i duplikatami.

Definicja w linii 90 pliku test.cpp.

4.7.1.6 TEST() [6/12]

```
TEST (
            Tests ,
            negativeDuplicates )
```

Test sortowania dla tablicy z powtarzajacymi sie numerami ale negatynej.

Definicja w linii 61 pliku test.cpp.

4.7.1.7 TEST() [7/12]

```
TEST (
            Tests ,
            negnumb )
```

Test sortowania dla tablicy z negetywnymi numerami.

Definicja w linii 26 pliku test.cpp.

4.7.1.8 TEST() [8/12]

```
TEST (
            Tests ,
            negposnumb )
```

Test sortowania dla tablicy z mieszanymi liczbami.

Definicja w linii 33 pliku test.cpp.

4.7.1.9 TEST() [9/12]

```
TEST (
            Tests ,
            onenumb )
```

Test sortowania dla tablicy jednoliczbowej.

Definicja w linii 47 pliku test.cpp.

4.7.1.10 TEST() [10/12]

```
TEST (
            Tests ,
            randomOrder )
```

Test sortowania dla tablicy losowej.

Definicja w linii 19 pliku test.cpp.

4.7.1.11 TEST() [11/12]

```
TEST (
            Tests ,
            reverseOrder )
```

Test sortowania dla tablicy juz posortowanej odwrotnie.

Definicja w linii 12 pliku test.cpp.

4.7.1.12 TEST() [12/12]

```
TEST (
            Tests ,
            twonumb )
```

Test sortowania dla tablicy z dwoma numerami.

Definicja w linii 68 pliku test.cpp.

4.8 test.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "../MergeSort/Sort.cpp"
00002 #include "../MergeSort/sort.h"
00003 #include <gtest/gtest.h>
00005 TEST(Tests, alreadydone) {
00006     std::vector<int> t = { 1, 2, 3, 4, 5 };
00007     MergeSort::Sort(t);
00008     std::vector<int> expected = { 1, 2, 3, 4, 5 };
00009     ASSERT_EQ(t, expected);
00010 }
00012 TEST(Tests, reverseOrder) {
00013     std::vector<int> t = { 10, 6, 5, 2, 1 };
00014     MergeSort::Sort(t);
00015     std::vector<int> expected = { 1, 2, 5, 6, 10 };
00016     ASSERT_EQ(t, expected);
00017 }
00019 TEST(Tests, randomOrder) {
00020     std::vector<int> t = { 3, 1, 4, 5, 2 };
00021     MergeSort::Sort(t);
00022     std::vector<int> expected = { 1, 2, 3, 4, 5 };
00023     ASSERT_EQ(t, expected);
00024 }
00026 TEST(Tests, negnumb) {
00027     std::vector<int> t = { -12, -8, -4, -16, -2 };
00028     MergeSort::Sort(t);
00029     std::vector<int> expected = { -16, -12, -8, -4, -2 };
00030     ASSERT_EQ(t, expected);
00031 }
00033 TEST(Tests, negposnumb) {
00034     std::vector<int> t = { -53, 20, -14, 5, 2 };
00035     MergeSort::Sort(t);
00036     std::vector<int> expected = { -53, -14, 2, 5, 20 };
00037     ASSERT_EQ(t, expected);
00038 }
00040 TEST(Tests, empty) {
00041     std::vector<int> t = {};
00042     MergeSort::Sort(t);
00043     std::vector<int> expected = {};
00044     ASSERT_EQ(t, expected);
00045 }
00047 TEST(Tests, onenumb) {
00048     std::vector<int> t = { 42 };
00049     MergeSort::Sort(t);
00050     std::vector<int> expected = { 42 };
00051     ASSERT_EQ(t, expected);
00052 }
00054 TEST(Tests, duplicates) {
00055     std::vector<int> t = { 2, 1, 2, 8, 1 ,8 };
00056     MergeSort::Sort(t);
00057     std::vector<int> expected = { 1, 1, 2, 2, 8 ,8 };
00058     ASSERT_EQ(t, expected);
00059 }
00061 TEST(Tests, negativeDuplicates) {
00062     std::vector<int> t = { -2, -1, -2, -8, -1 ,-8 };
00063     MergeSort::Sort(t);
00064     std::vector<int> expected = { -8, -8, -2, -2, -1, -1 };
00065     ASSERT_EQ(t, expected);
00066 }
00068 TEST(Tests, twonumb) {
00069     std::vector<int> t = {1,2 };
00070     MergeSort::Sort(t);
00071     std::vector<int> expected = { 1,2 };
00072     ASSERT_EQ(t, expected);
00073 }
00075 TEST(Tests, hundred) {
00076     std::vector<int> t(101);
00077     std::vector<int> expected(101);
00078     for (int i = 0;i < 101;i++) {
00079         expected[i] = i*2;
00080     }
00081     for (int i = 0;i < 101;i++) {
00082         t[i] = i*2;
00083     }
00084     t[0] = 2;
00085     t[1] = 0;
00086     MergeSort::Sort(t);
00087     ASSERT_EQ(t, expected);
00088 }
00090 TEST(Tests, hundredmix) {
00091     std::vector<int> t(201);
00092     std::vector<int> expected(201);
00093     for (int i = -100;i < 101;i++) {
00094         int index = i + 100;
```

```
00095         expected[index] = i * 2;
00096         t[index] = i * 2;
00097     }
00098     t[20] = 16;
00099     expected[20] = 16;
00100     t[20] = 6;
00101     expected[20] = 6;
00102     std::sort(expected.begin(), expected.end());
00103     MergeSort::Sort(t);
00104     ASSERT_EQ(t, expected);
00105 }
```

Skorowidz