P O L I T E C H N I K A R Z E S Z O W S K A im.
Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

ANDRII KOTOVYCH
173163

ALGORYTMY I STRUKTURY DANYCH

Projekt

kierunek studiów: Inżynieria i analiza danych

Opiekun pracy:

Prof. Mariusz Borkowski

Rzeszów 2022

Spis treści

1. Wstęp 3
2. Algorytm 3
3. Schemat blokowy 3
4. Pseudokod 5
5. Rezultaty testów 6
6. Wykresy złożoności czasowej oraz obliczeniowej 9
7. Wnioski oraz podsumowanie 11
8. Kod 12

1. Wstęp

Zaimplementuj algorytm sortowania przez scalanie oraz algorytm sortowania przez wybieranie.

2. Algorytm

SCALANIE

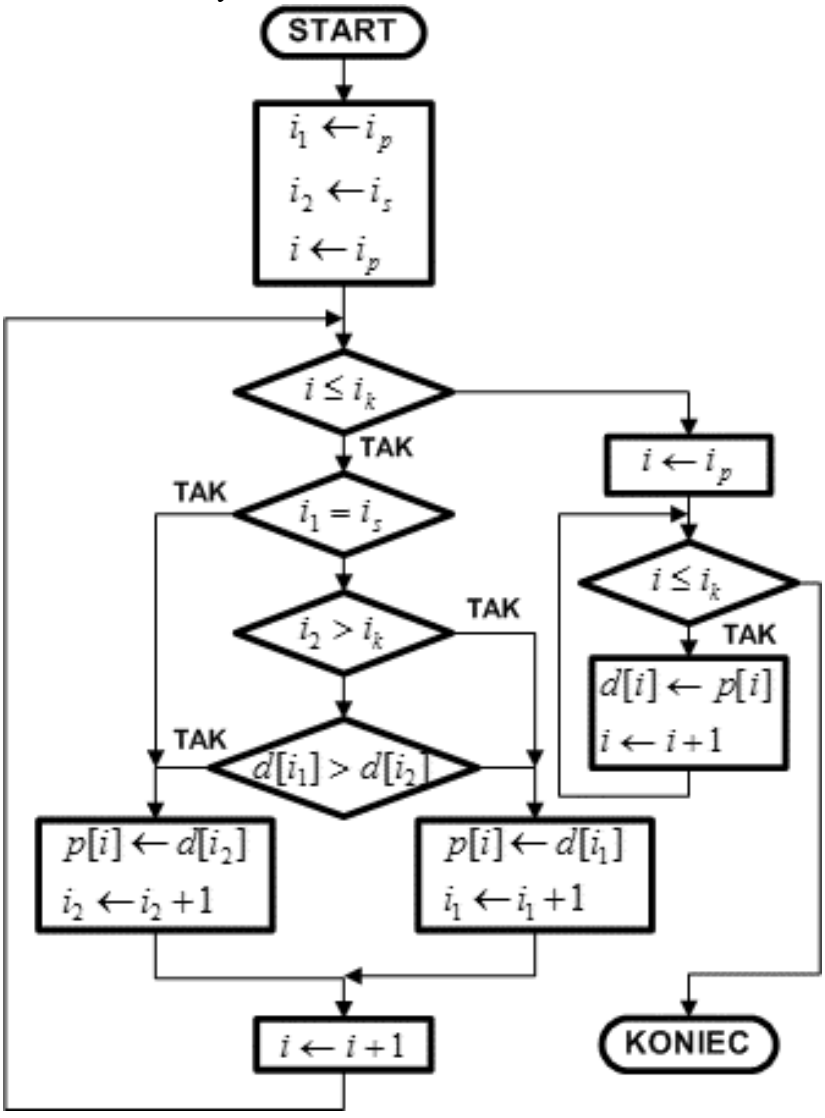
1. Dane dzielimy na dwie równe lub prawie równe części. Dopóki uzyskane dwie części nie są posortowanymi (czyli jednoelementowymi) fragmentami ponownie dzielimy osobno każdą z powstałych części tą samą metodą;
2. Kiedy mamy doczynienia z dwoma posortowanymi częściami , scalamy je w jeden dłuższy posortowany fragment i zwracamy go do scalenia z jego drugą połówką.;

WYBIERANIE

1. Tworzymy zbiór wartości posortowanych (na początku pusty);
2. Szukamy w naszej tablicy najmniejszej wartości, zabieramy ją i odkładamy na koniec zbioru posortowanego;

3. Schemat blokowy

Schemat blokowy scalanie



*Scalaj(***i_p, i_s, i_k***)*

Dane wejściowe

d[ ] - scalany zbiór

i_p - indeks pierwszego elementu w młodszym podzbiorze, $i_p \in N$

i_s - indeks pierwszego elementu w starszym podzbiorze, $i_s \in N$

i_k - indeks ostatniego elementu w starszym podzbiorze, $i_k \in N$

Dane wyjściowe

d[ ] - scalony zbiór

Zmienne pomocnicze

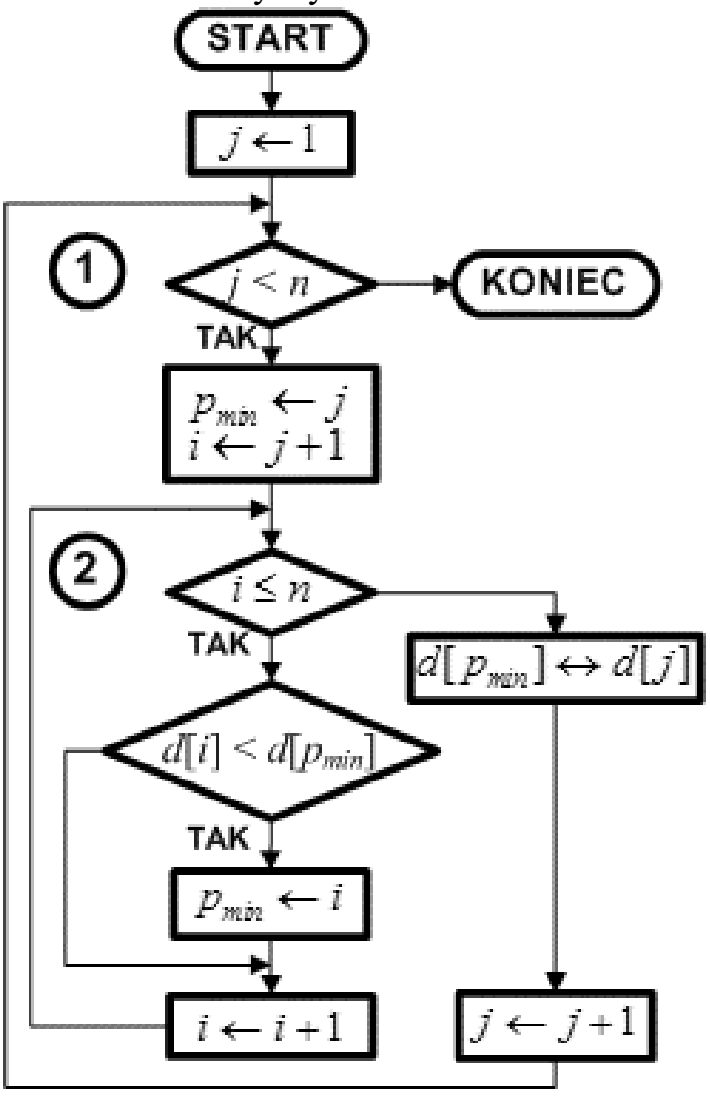
p[ ] - zbiór pomocniczy, który zawiera tyle samo elementów, co zbiór d[ ].

i_1 - indeks elementów w młodszej połówce zbioru d[ ], $i_1 \in N$

i_2 - indeks elementów w starszej połówce zbioru d[ ], $i_2 \in N$

i - indeks elementów w zbiorze pomocniczym p[ ], $i \in N$

Schemat blokowy wybieranie



Dane wejściowe

n - liczba elementów w sortowanym zbiorze, $n \in N$

d[ ] - zbiór n-elementowy, który będzie sortowany. Elementy zbioru mają indeksy od 1 do n.

Dane wyjściowe

d[ ] - posortowany zbiór n-elementowy. Elementy zbioru mają indeksy od 1 do n.

Zmienne pomocnicze

i, j - zmienne sterujące pętli, $i, j \in N$

p_{min} - pozycja elementu minimalnego w zbiorze d[ ], $p_{min} \in N$

4. Pseudokod

SCALANIE

K01: $i_1 \leftarrow i_p$; $i_2 \leftarrow i_s$; $i \leftarrow i_p$

K02: **Dla** $i = i_p, i_p + 1, ..., i_k$:
jeśli $(i_1 = i_s) \vee (i_2 \leq i_k$ i $d[i_1] > d[i_2])$,
to $p[i] \leftarrow d[i_2]$; $i_2 \leftarrow i_2 + 1$
inaczej $p[i] \leftarrow d[i_1]$; $i_1 \leftarrow i_1 + 1$

K03: **Dla** $i = i_p, i_p + 1,...,i_k$:
$d[i] \leftarrow p[i]$

K04: **Zakończ**

WYBIERANIE

K01: **Dla** $j = 1, 2, ..., n - 1$:
wykonuj kroki K02...K04

K02: $p_{min} \leftarrow j$

K03: **Dla** $i = j + 1, j + 2, ..., n$:
jeśli $d[i] < d[p_{min}]$,
to $p_{min} \leftarrow i$

K04: $d[j] \leftrightarrow d[p_{min}]$

K05: **Zakończ**

5. Rezultaty testów

SCALANIE

```
 Sortowanie przez scalanie
--------------------------
  2022 Andrii Kotovych

Przed sortowaniem:

  96   1  68  47  96  76  30  53  80  33
Po sortowaniu:

   1  30  33  47  53  68  76  80  96  96

Process finished with exit code 0
```

```
 Sortowanie przez scalanie
--------------------------
  2022 Andrii Kotovych

Przed sortowaniem:

   7  98  58  99  11  13  49  98  18  75  86  91  86  94  20
Po sortowaniu:

   7  11  13  18  20  49  58  75  86  86  91  94  98  98  99

Process finished with exit code 0
```

```
 Sortowanie przez scalanie
--------------------------
  2022 Andrii Kotovych

Przed sortowaniem:

   3  75  22  52  92  81  12   5  56  58  52  36  55  88  80  29  30  38  88  70
Po sortowaniu:

   3   5  12  22  29  30  36  38  52  52  55  56  58  70  75  80  81  88  88  92

Process finished with exit code 0
```

```
 Sortowanie przez scalanie
--------------------------
  2022 Andrii Kotovych

Przed sortowaniem:

  70  10  97  50  80  85  49  48  58  66  76  28  41  95  87  63  90  50  46  65  32  57  77   8  97
Po sortowaniu:

   8  10  28  32  41  46  48  49  50  50  57  58  63  65  66  70  76  77  80  85  87  90  95  97  97

Process finished with exit code 0
```

```
  2022 Andrii Kotovych

Przed sortowaniem:

  50  40   8  55  99  89  81  74  14  23  18  31  81  90  53  31  26  46  64  91  95  13  65  83  34  28  77  65   0  63

Po sortowaniu:

   0   8  13  14  18  23  26  28  31  31  34  40  46  50  53  55  63  64  65  65  74  77  81  81  83  89  90  91  95  99


Process finished with exit code 0
```

WYBIERANIE

```
Przed sortowaniem:

  34  66  31  12  91  10  38  63  81  88
Po sortowaniu:

  10  12  31  34  38  63  66  81  88  91

Process finished with exit code 0
```

```
Przed sortowaniem:

  71  47  62  31  51  92  88  89  72  13  52   1  57  71   7
Po sortowaniu:

   1   7  13  31  47  51  52  57  62  71  71  72  88  89  92

Process finished with exit code 0
```

```
Przed sortowaniem:

  11  56  56  79  22  77  70  30   3  73  77  29  80  19  69  13  97  88  70  77
Po sortowaniu:

   3  11  13  19  22  29  30  56  56  69  70  70  73  77  77  77  79  80  88  97

Process finished with exit code 0
```

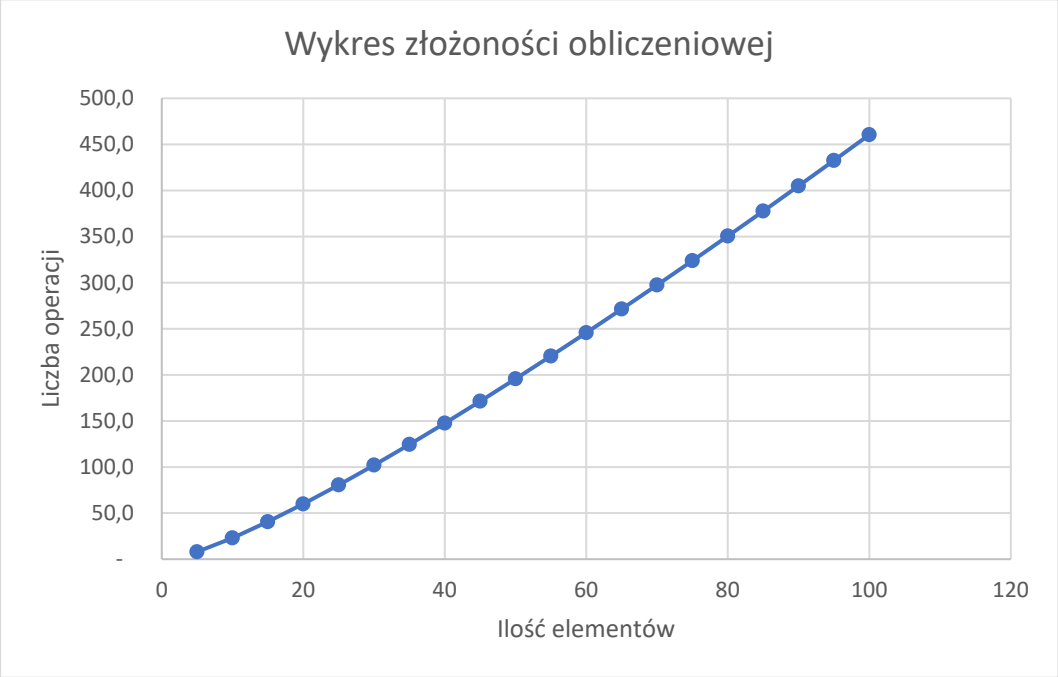
```
Przed sortowaniem:

  11  56  56  79  22  77  70  30   3  73  77  29  80  19  69  13  97  88  70  77
Po sortowaniu:

   3  11  13  19  22  29  30  56  56  69  70  70  73  77  77  77  79  80  88  97

Process finished with exit code 0
```

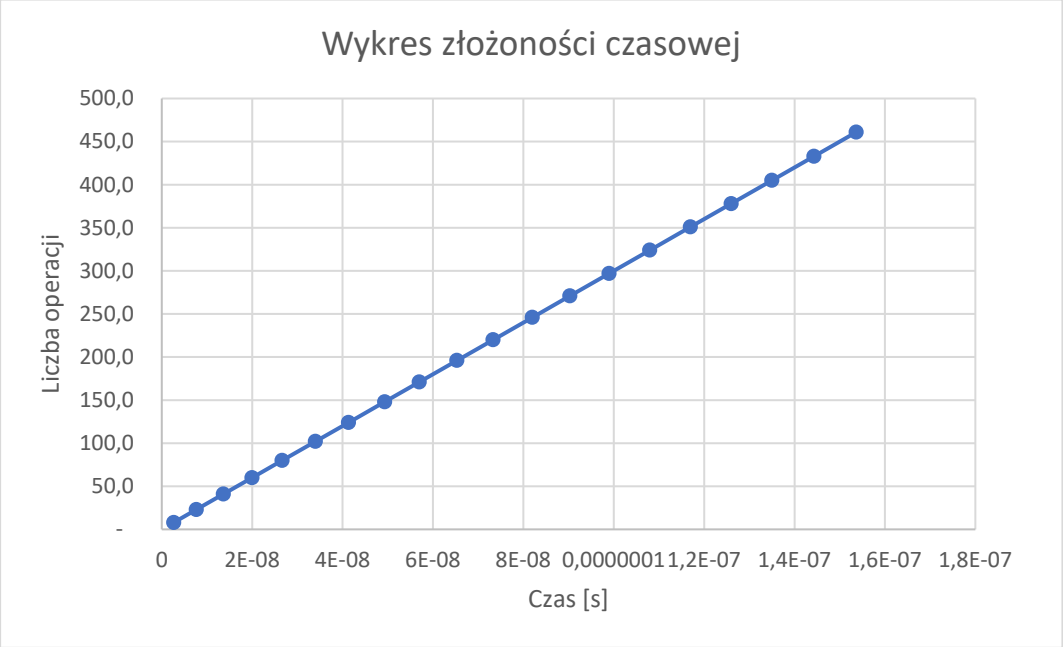
```
Przed sortowaniem:

  52   6  85  24  68  43  15  88  45  68   0  89  56  63  89  79  80  99  72  98  52  60  58  56  81  22  37  90   8  46
Po sortowaniu:

   0   6   8  15  22  24  37  43  45  46  52  52  56  56  58  60  63  68  68  72  79  80  81  85  88  89  89  90  98  99

Process finished with exit code 0
```

6. Wykresy złożoności czasowej oraz obliczeniowej

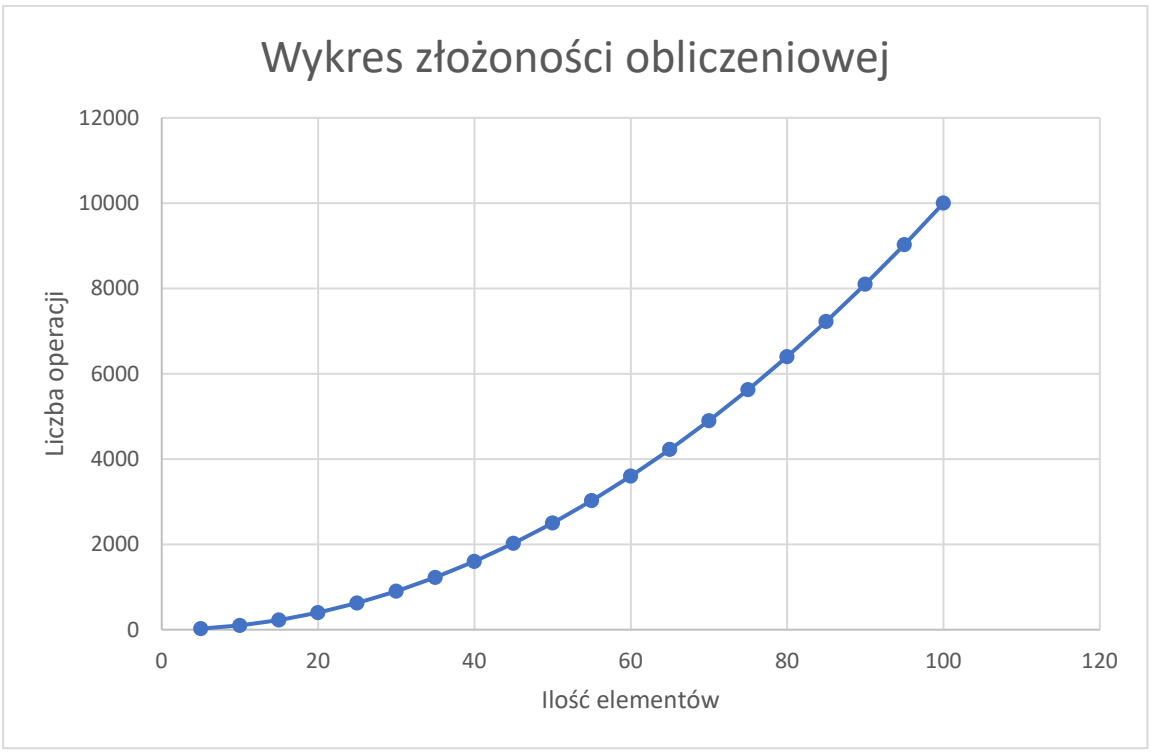
Moc obliczeniowa 3Ghz (około 3000000000 operacji/sekunda)

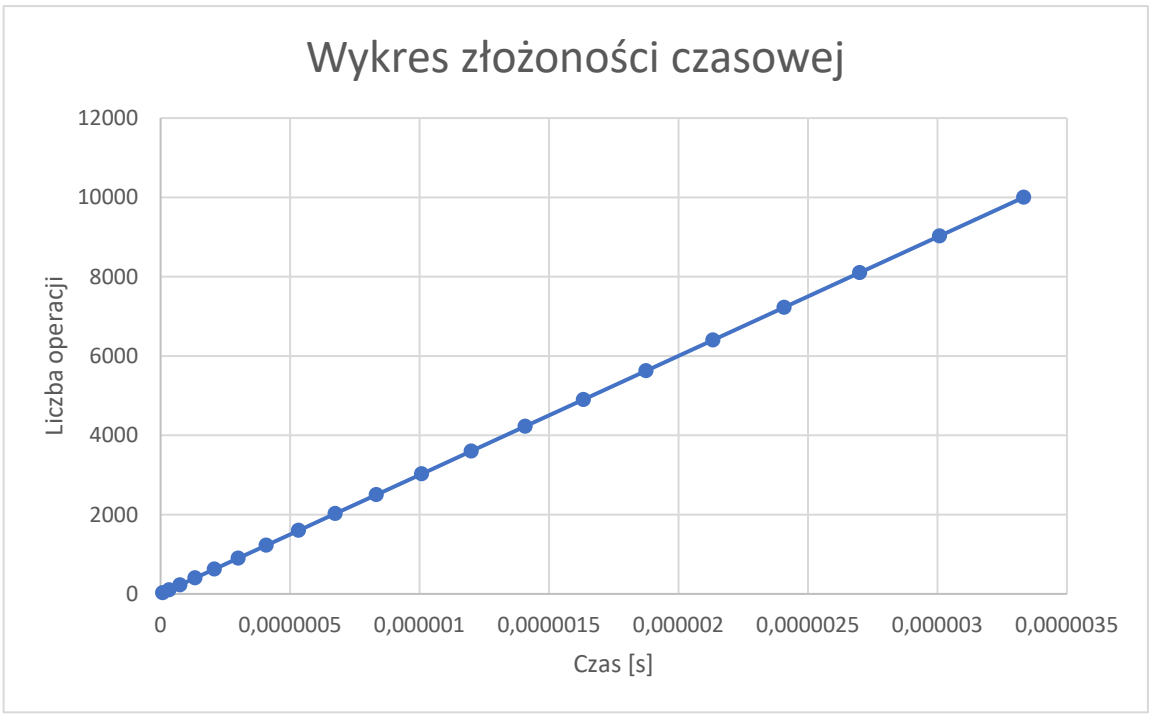
SCALANIE





WYBIERANIE


Wykres złożoności obliczeniowej
Liczba operacji
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
Ilość elementów


Wykres złożoności czasowej
Liczba operacji
12000
10000
8000
6000
4000
2000
0
0
0,0000005
0,000001
0,0000015
0,000002
0,0000025
0,000003
0,0000035
Czas [s]

7. Wnioski oraz podsumowanie

SCALANIE

Dobry i szybki algorytm względem złożoności czasowej, gorzej jest ze złożonością pamięciową bo potrzebuje jeszcze tyle samo pamięci dla tworzenia dodatkowej tablicy.

Złożoność O(n log n), tego typu algorytmy są nieco wolniejsze od algorytmów o złożoności liniowej, ale nadal bardzo wydajne.

WYBIERANIE

Prosty algorytm, wystarczy szukać minimum w przedziale, ale jest on mało wydajny i ilość operacji wiodących jest niezależna od układu danych na wejściu – porównuje nawet jeśli nie musi.

Złóżoność obliczeniowa- Złóżoność kwadratowa O(n^2), dwie zagnieżdżone pętli ze złożonością liniową każdej pętli.

8. Kod

WYBIERANIE

```cpp
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

// Funkcja do sortowania przez wybieranie
void selectionSort(vector<int>& v) {
  // Iterujemy przez wszystkie elementy w wektorze
  for (int i = 0; i < v.size(); i++) {
    // Szukamy najmniejszego elementu w reszcie wektora
    int minIndex = i;
    for (int j = i + 1; j < v.size(); j++) {
      if (v[j] < v[minIndex]) {
        minIndex = j;
      }
    }
    // Zamieniamy element na pozycji i z najmniejszym znalezionym elementem
    swap(v[i], v[minIndex]);
  }
}

int main() {
  // Otwieramy plik wejściowy
  ifstream inputFile("in.txt");
  if (!inputFile.is_open()) {
    cout << "Błąd otwarcia pliku wejściowego!" << endl;
    return 1;
  }

  // Wczytujemy dane z pliku do wektora
  vector<int> data;
  int x;
  while (inputFile >> x) {
    data.push_back(x);
  }

  // Sortujemy dane przy użyciu funkcji selectionSort
  selectionSort(data);

  // Otwieramy plik wyjściowy
  ofstream outputFile("out.txt");
  if (!outputFile.is_open()) {
    cout << "Błąd otwarcia pliku wyjściowego!" << endl;
    return 1;
  }

  // Zapisujemy posortowane dane do pliku
  for (int x : data) {
    outputFile << x << " ";
  }

  return 0;
}
```

SCALANIE

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <time.h>
#include <fstream>

using namespace std;

const int N = 20; // Liczebność zbioru.

int d[N],p[N];

// Procedura sortująca
//--------------------

void MergeSort(int i_p, int i_k)
{
  int i_s,i1,i2,i;

  i_s = (i_p + i_k + 1) / 2;
  if(i_s - i_p > 1) MergeSort(i_p, i_s - 1);
  if(i_k - i_s > 0) MergeSort(i_s, i_k);
  i1 = i_p; i2 = i_s;
  for(i = i_p; i <= i_k; i++)
    p[i] = ((i1 == i_s) || ((i2 <= i_k) && (d[i1] > d[i2]))) ? d[i2++] : d[i1++];
  for(i = i_p; i <= i_k; i++) d[i] = p[i];
}

// Program główny
//---------------

int main()
{
  int i;

  cout << " Sortowanie przez scalanie\n";

//Otwórz pliki dla odczytu i zapisu
  fstream plik,plik2;
  plik.open("xxx.txt");
  plik2.open("yyy.txt",ios::out);
  for(i = 0; i < N; i++) plik>>d[i];
  for(i = 0; i < N; i++) cout << setw(4) << d[i];
  cout << endl;

// Sortujemy

  MergeSort(0,N-1);

// Wyświetlamy wynik sortowania i zapisujemy do pliku

  cout << "Po sortowaniu:\n\n";
  for(i = 0; i < N; i++) cout << setw(4) << d[i];
  cout << endl;
  for(int i=0;i<N;i++) plik2<<d[i]<<endl;
  return 0;
}
```