Jan Danowski

Politechnika Rzeszowska

Algorytmy i struktury danych

Sprawozdanie

Projekt 2

Opis problemu

Zaimplementuj sortowanie Quicksort oraz sortowanie Gnome sort

Opis podstaw teoretycznych

1. Quicksort - Algorytm wykorzystuje technikę "dziel i zwyciężaj". Według ustalonego schematu wybierany jest jeden element w sortowanej tablicy, który będziemy nazywać pivot. Pivot może być elementem środkowym, pierwszym, ostatnim, losowym lub wybranym według jakiegoś innego schematu dostosowanego do zbioru danych. Następnie ustawiamy elementy nie większe na lewo tej wartości, natomiast nie mniejsze na prawo. W ten sposób powstaną nam dwie części tablicy (niekoniecznie równe), gdzie w pierwszej części znajdują się elementy nie większe od drugiej. Następnie każdą z tych podtablic sortujemy osobno według tego samego schematu.

   Zalety:

   • działa w miejscu, czyli ” in situ” (używa tylko niewielkiego stosu pomocniczego)

   • do posortowania n elementów wymaga średnio czasu proporcjonalnego do n ∗ logn

   • ma wyjątkowo skromną pętlę wewnętrzą

   Wady:

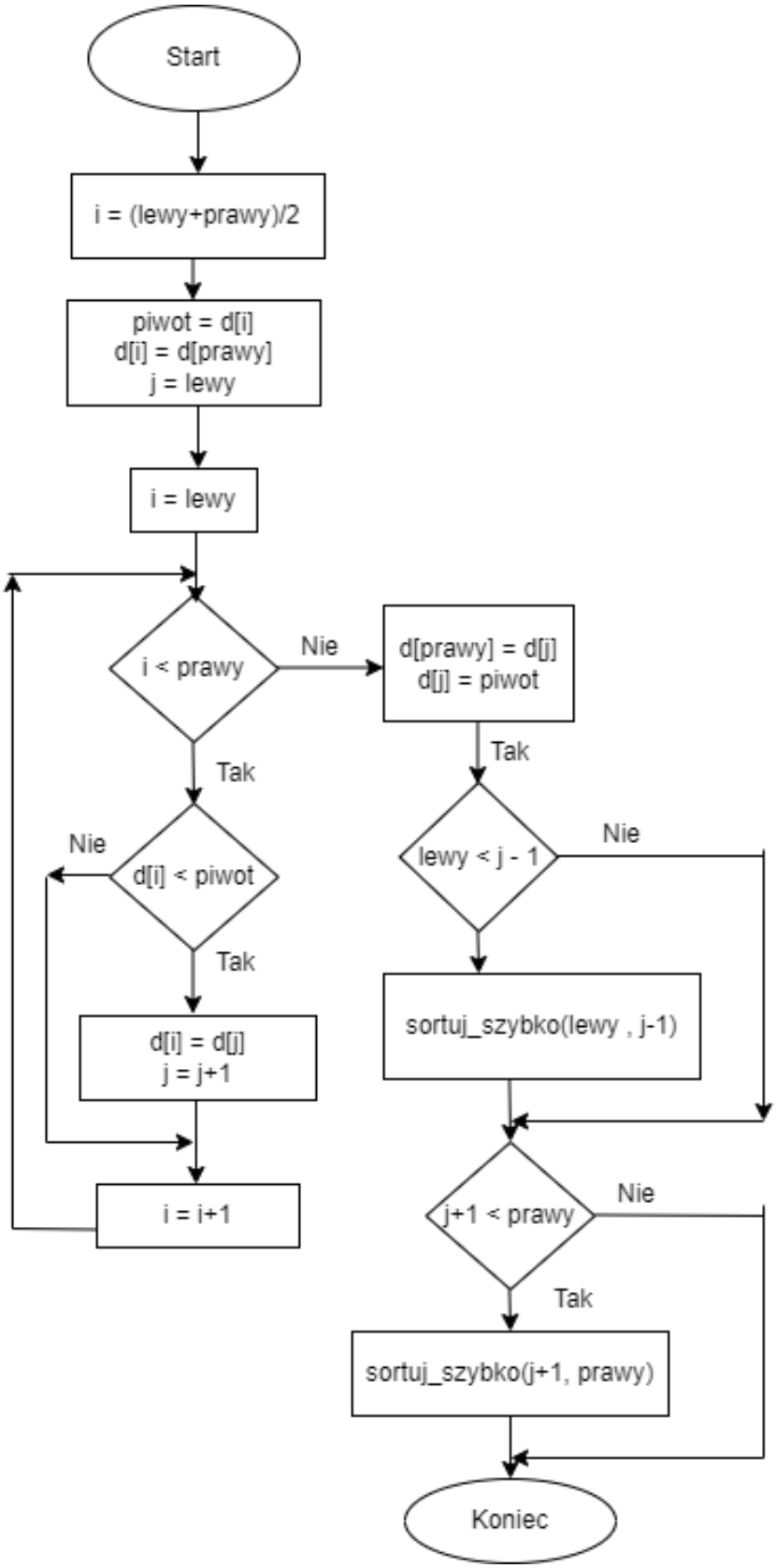
   • jest niestabilny

   • zabiera około n^2 operacji w najgorszym przypadku

   • jest wrażliwy (prosty niezauważony błąd w implementacji może powodować niewłaściwe działanie w przypadku niektórych danych)

2. Gnome sort - Sortowanie Gnoma to metoda sortowania danych, która wymaga jedynie jednej pętli. Implementacja częściowo opiera się na pomyśle z sortowania bąbelkowego. Algorytm wymaga tylko zmiennej do zapamiętania aktualnej pozycji sortowania. Sortowanie rozpoczyna się od pierwszego elementu na liście. Jeśli aktualnie rozpatrywany element jest pierwszym elementem na liście, albo spełnia warunki posortowania to należy zwiększyć numer indeksu. W przeciwnym wypadku należy zamienić aktualny element z poprzednim i zmniejszyć indeks o 1.

Schemat blokowy quicksort



Pseudokod quicksort

K01: i ← (lewy + prawy)/2

K02: piwot ← d[i]; d[i] ← d[prawy]; j ← lewy

K03: Dla i = lewy, lewy + 1, ..., prawy - 1:

wykonuj kroki K04...K05

K04: Jeśli d[i] ⩾ piwot,

to wykonaj kolejny obieg pętli K03

K05: d[i] ↔ d[j]; j ← j + 1

K06: d[prawy] ← d[j]; d[j] ← piwot

K07: Jeśli lewy < j - 1,

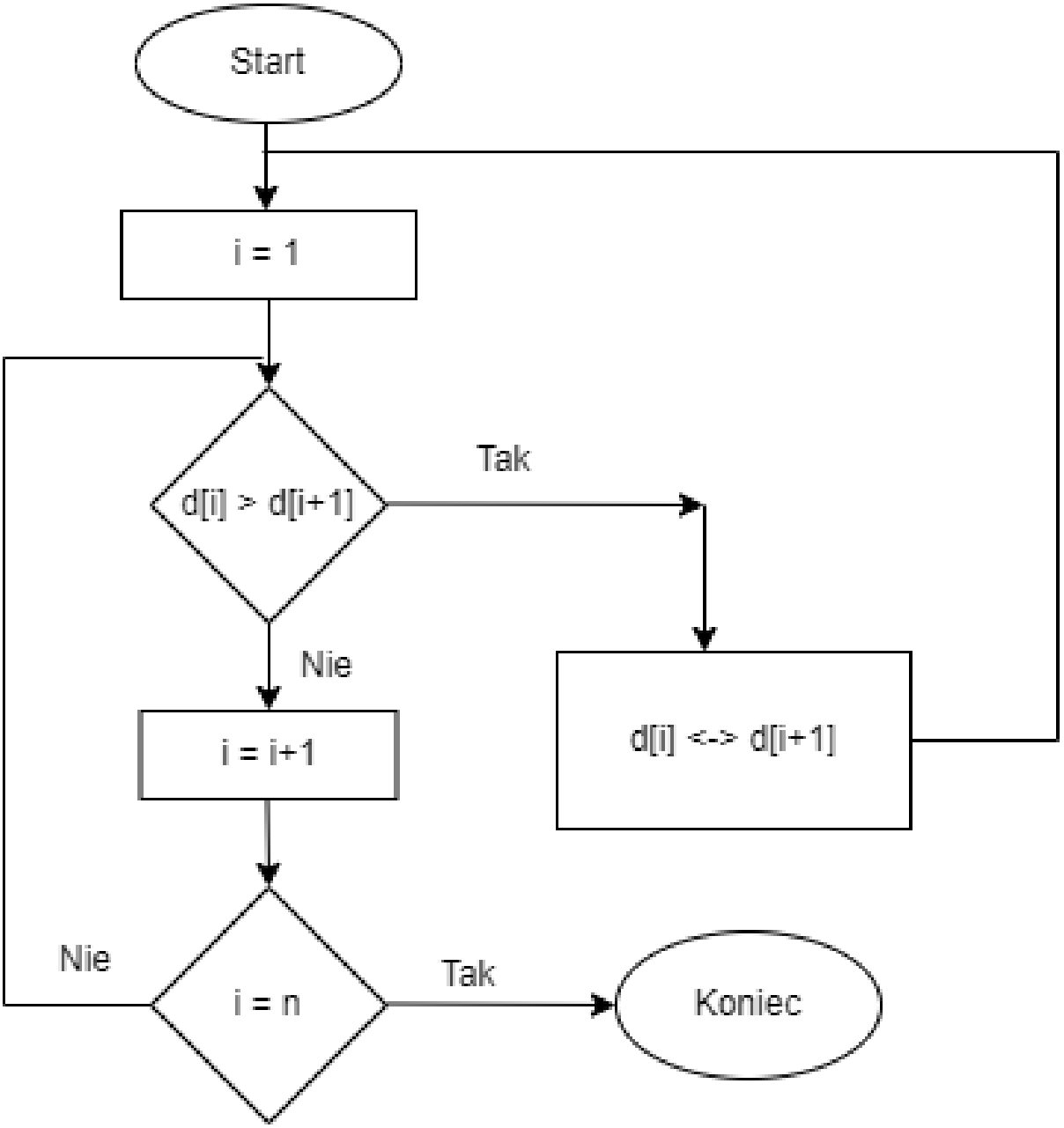
to Sortuj_szybko(lewy, j - 1)

K08: Jeśli j + 1 < prawy,

to Sortuj_szybko(j + 1, prawy)

K09: Koniec

Schemat blokowy gnome sort



Pseudokod gnome sort

K01: Dla i = 1,2,...,n - 1:

Wykonuj kroki K02...K04

K02: Jeśli d[i] ⩽ d[i + 1],

to wykonaj następny obieg K01

K03: d[i] ↔ d[i + 1]

K04: Idź do kroku K01

K05: Koniec

Złożoność obliczeniowa i wykresy quicksort w przypadku optymistycznym i pesymistycznym

O(n log n) w przypadku optymistycznym, w przypadku pesymistycznym O(n2)

1. **Przypadek optymistyczny, również średni gdyż różnica między nimi jest nieznaczna**

```
  26  47  15   0  97  75  11   8  14   1  74  34  42  28  58   9  54  58  24  71
Po sortowaniu:

   0   1   8   9  11  14  15  24  26  28  34  42  47  54  58  58  71  74  75  97
Czas w mikrosekundach: 5996
```

Rys. 1 Wynik programu dla 20 elementów

```
  98  98  99  99  99  99  99  99  99  99
Czas w mikrosekundach: 242712
```

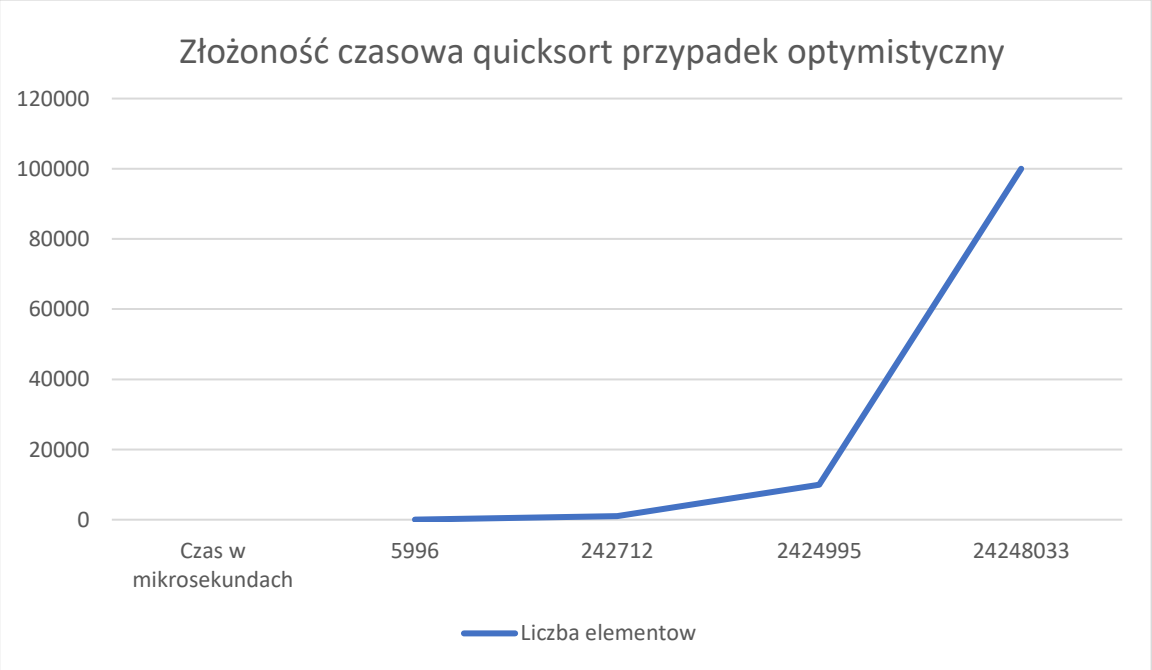
Rys. 2 Wynik programu dla 1000 elementów

```
  99  99  99  99  99  99  99  99  99  99
  99  99  99  99  99  99  99  99  99  99
Czas w mikrosekundach: 2424995
```

Rys. 3 Wynik programu dla 10000 elementów

```
  99  99  99  99  99  99  99  99  99  99
Czas w mikrosekundach: 24248033
```

Rys. 4 Wynik dla 100 000 elementów



2. **Przypadek pesymistyczny**

```
   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
Po sortowaniu:

   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
Czas w mikrosekundach: 6000
```

Rys. 5 Pesymistyczny dla 20 elementów

```
Czas w mikrosekundach: 215787
```

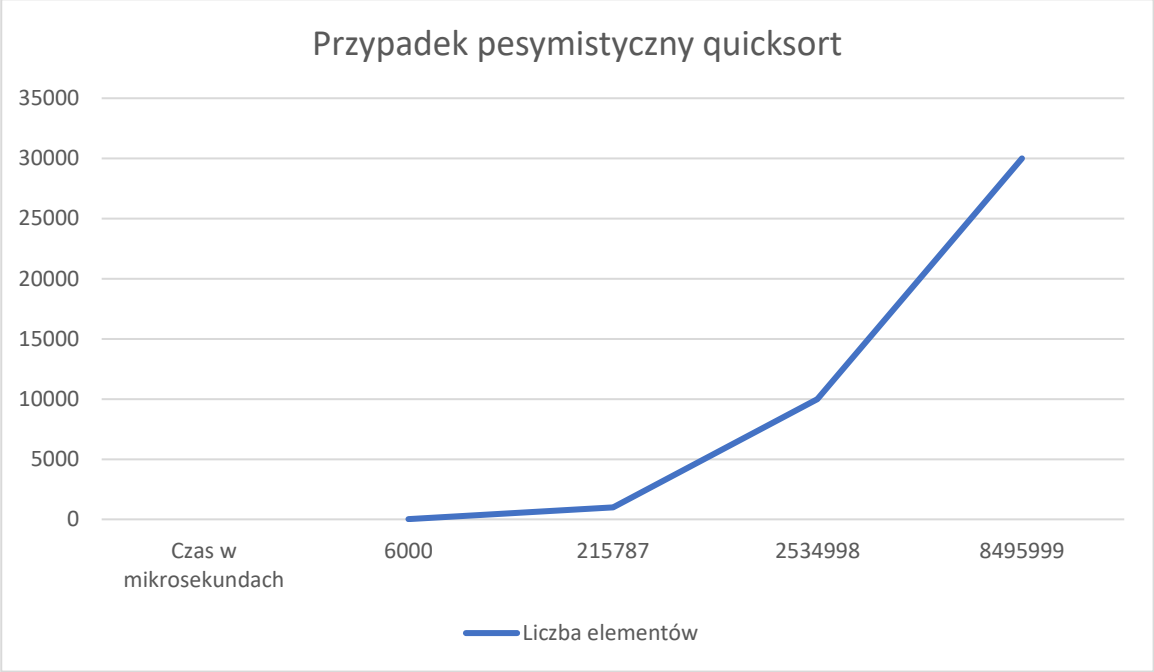
Rys. 6 Pesymistyczny 1000 elementów

```
Czas w mikrosekundach: 2534998
```

Rys. 7 Pesymistyczny dla 10000 elementów

```
Czas w mikrosekundach: 8495999
```

Rys. 8 Pesymistyczny dla 30000 elementów



Złożoność obliczeniowa i wykresy gnome sort w przypadku optymistycznym i pesymistycznym

Złożoność w przypadku optymistycznym wynosi O(n^2), w pesymistycznym O(n^3)

1. **Przypadek optymistyczny**

```
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
Po sortowaniu:

  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
Czas w mikrosekundach: 4997
```

Rys. 9 Dla 20 elementów

```
Czas w mikrosekundach: 222995
```

Rys. 10 Dla 1000 elementów

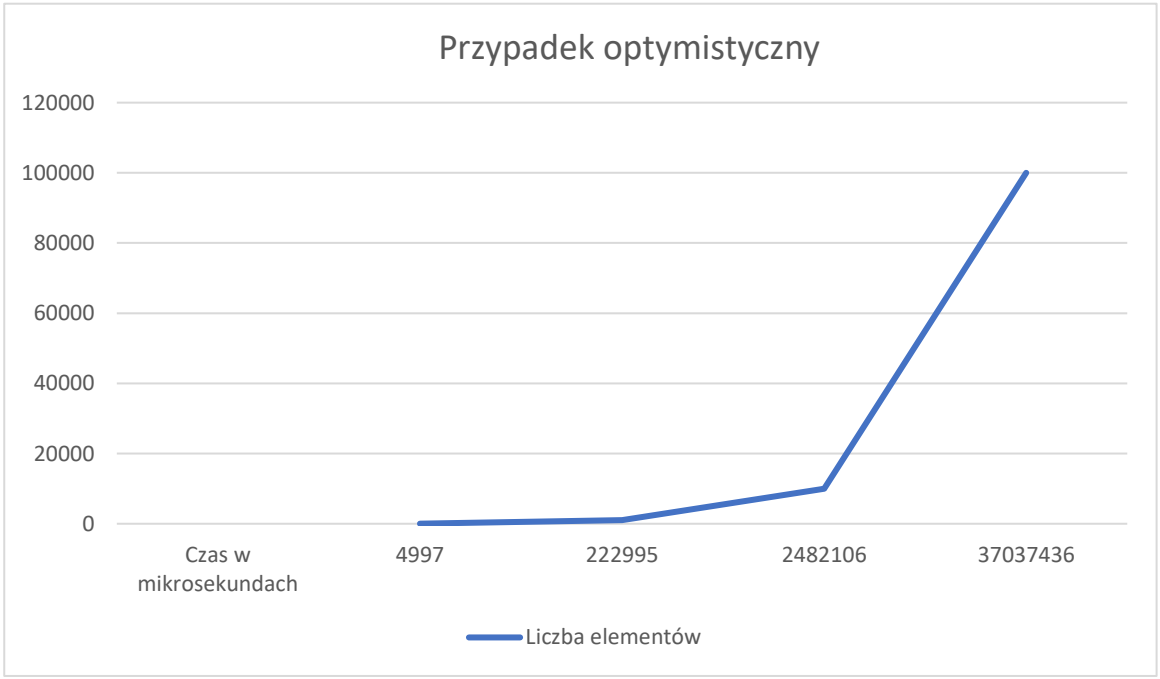
```
Czas w mikrosekundach: 2482106
```

Rys. 11 Dla 10000 elementów

```
Czas w mikrosekundach: 37037436
```

Rys. 12 Dla 100 000 elementów

1.1. Wykres dla przypadku optymistycznego – tablicy z góry posortowana



2. **Przypadek pesymistyczny**

```
 80  52  80  56  13  75   6  70  67  44  62   5  42  35  44  23  75  35  36  31
Po sortowaniu:

  5   6  13  23  31  35  35  36  42  44  44  52  56  62  67  70  75  75  80  80
Czas w mikrosekundach: 4997
```

Rys. 13 Dla 20 elementów

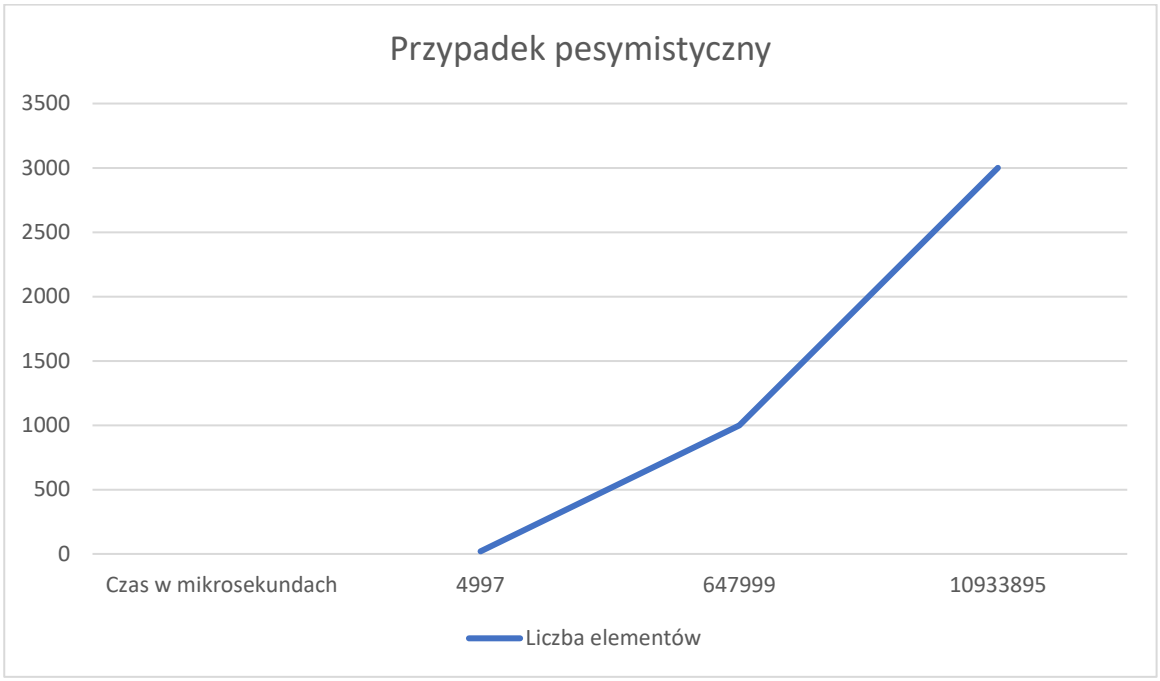
```
Czas w mikrosekundach: 647999
```

Rys. 14 Dla 1000 elementów

```
Czas w mikrosekundach: 10933895
```

Rys. 15 Dla 3000 elementów

2.1. Wykres dla przypadku pesymistycznego – przy ilości większej niż 3000 elementów program przestaje działać



Kod źródłowy sortowania quicksort i gnome

Quicksort :

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <time.h>
#include <chrono>
#include <string>

#include <fstream>
using namespace std;
using namespace chrono;



const int N = 20; // Liczebność zbioru.

int d[N];

void saveResultsToFile(string file_name, int d[], int rozmiar) {
  fstream plik;
  plik.open(file_name + ".txt", ios:: in | ios::out); //otworzenie pliku
  if (plik.good() == true) {
    cout << "Po sortowaniu:\n\n";
    for (int i = 0; i < rozmiar; i++) {
      plik << d[i] << endl; // wpisujemy posortowane dane do pliku
      cout << setw(4) << d[i]; // wypisujemy posortowane dane
    }
    plik.close(); // zamknięcie pliku
```

```
  }
}
// Procedura sortowania szybkiego
//------------------------------

void Sortuj_szybko(int lewy, int prawy) //funkcja do quicksorta
{
  int i, j, piwot; //deklaracja zmiennych

  i = (lewy + prawy) / 2;
  piwot = d[i];
  d[i] = d[prawy];
  for (j = i = lewy; i < prawy; i++)
    if (d[i] < piwot) //warunek do podmiany
  {
    swap(d[i], d[j]); //podmiana elementow
    j++;
  }
  d[prawy] = d[j];
  d[j] = piwot;
  if (lewy < j - 1) Sortuj_szybko(lewy, j - 1);
  if (j + 1 < prawy) Sortuj_szybko(j + 1, prawy);
}

// Program główny
//---------------

int main() {
  int i;
  auto start = high_resolution_clock::now();
  srand((unsigned) time(NULL));
```

```
  // Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
  // a następnie wyświetlamy jej zawartość

  for (i = 0; i < N; i++) d[i] = rand() % 100;
  for (i = 0; i < N; i++) cout << setw(4) << d[i];
  cout << endl;

  // Sortujemy

  Sortuj_szybko(0, N - 1);

  // Wyświetlamy wynik sortowania

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

  saveResultsToFile("zapis", d, N);
  cout<<endl;

  auto stop = high_resolution_clock::now();
  auto duration = duration_cast < microseconds > (stop - start);

  cout << "Czas w mikrosekundach: " << duration.count() << endl;

  return 0;
}
```

Gnome:

```
#include <cmath>
#include <iostream>
```

```
#include <iomanip>
#include <cstdlib>
#include <time.h>
#include <string>
#include <chrono>
#include<fstream>
using namespace std;
using namespace chrono;
const int N = 20; // Liczebność zbioru.

void saveResultsToFile(string file_name, int d[], int rozmiar) {
  fstream plik;
  plik.open(file_name + ".txt", ios:: in | ios::out); //otworzenie pliku
  if (plik.good() == true) {
    cout << "Po sortowaniu:\n\n";
    for (int i = 0; i < rozmiar; i++) {
      plik << d[i] << endl; // wpisujemy posortowane dane do pliku
      cout << setw(4) << d[i]; // wypisujemy posortowane dane
    }
    plik.close(); // zamknięcie pliku
  }
}
// Program główny
//---------------

int main() {
  auto start = high_resolution_clock::now();
  int d[N], i;

  // Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
  // a następnie wyświetlamy jej zawartość
```

```
srand((unsigned) time(NULL));

for (i = 0; i < N; i++) d[i] = rand() % 100;
for (i = 0; i < N; i++) cout << setw(4) << d[i];
cout << endl;

// Sortujemy

i = 0;
do {
  if (d[i] > d[i + 1]) // Porządek rosnący
  {
    swap(d[i], d[i + 1]);
    i = 0;
    continue;
  }
  i++;
} while (i < N - 1);

// Wyświetlamy wynik sortowania

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

saveResultsToFile("zapis", d, N);
cout << endl;

auto stop = high_resolution_clock::now();
auto duration = duration_cast < microseconds > (stop - start);
```

```
  cout << "Czas w mikrosekundach: " << duration.count() << endl;
  return 0;
}
```

Link do githuba

https://github.com/Dzbanowsky/Projekt2