Rafał Gaweł

Omówienie i porównanie algorytmów Gnoma i przez Zliczanie.

Projekt z Algorytmów i struktur danych

Rzeszów, 2021

Spis treści

1. Wstęp

Porównam dzisiaj 2 algorytmy: Sortowanie Gnoma i Sortowanie przez Zliczanie. Są one jednymi z wielu algorytmów pozwalających nam sortować dane w informatyce. Na podstawie wykresów ocenie który algorytm radzi sobie lepiej zależnie od danych.

2. Opis problemu

Moim zadaniem było zaimplementować oba algorytmy w języku c++, po czym wykonać parę testów na danych różnej wielkości, a także odpowiednio spreparować dane, wiedząc przy jakich danych algorytmy radzą sobie gorzej czy lepiej.

3. Podstawy teoretyczne

3.1. Sortowanie Gnoma:

Sortowanie Gnoma polega na prostej zasadzie, w której przechodzimy po kolei po parach i jeśli jakaś nie jest w odpowiedniej kolejności zamieniamy je miejscami. Oczywiście może wtedy powstać kolejna niepoprawna para więc powtarzamy zamiany aż element trafi na odpowiednie miejsce. Jeśli przejdziemy przez wszystkie elementy i nie ma par niepoprawnych, ciąg jest posortowany.

3.2. Sortowanie przez Zliczanie:

Sortowanie przez Zliczanie tworzy pomocniczą tablicę która zlicza ile razy występuje każdy z elementów tablicy sortowanej, po czym generuje na jej podstawie tablicę posortowaną wstawiając wartości na odpowiednie indeksy. Każdy element ma swój osobny licznik, zaczynamy od największego, sumując liczniki wszystkich poprzednich elementów otrzymujemy indeks na którym powinien być dany element, po czym z jego licznika odejmujemy jeden, powtarzamy ten proces, aż nie wyzerujemy wszystkich liczników, dzięki czemu mamy posortowaną tabele.

4. Szczegóły implementacji problemu

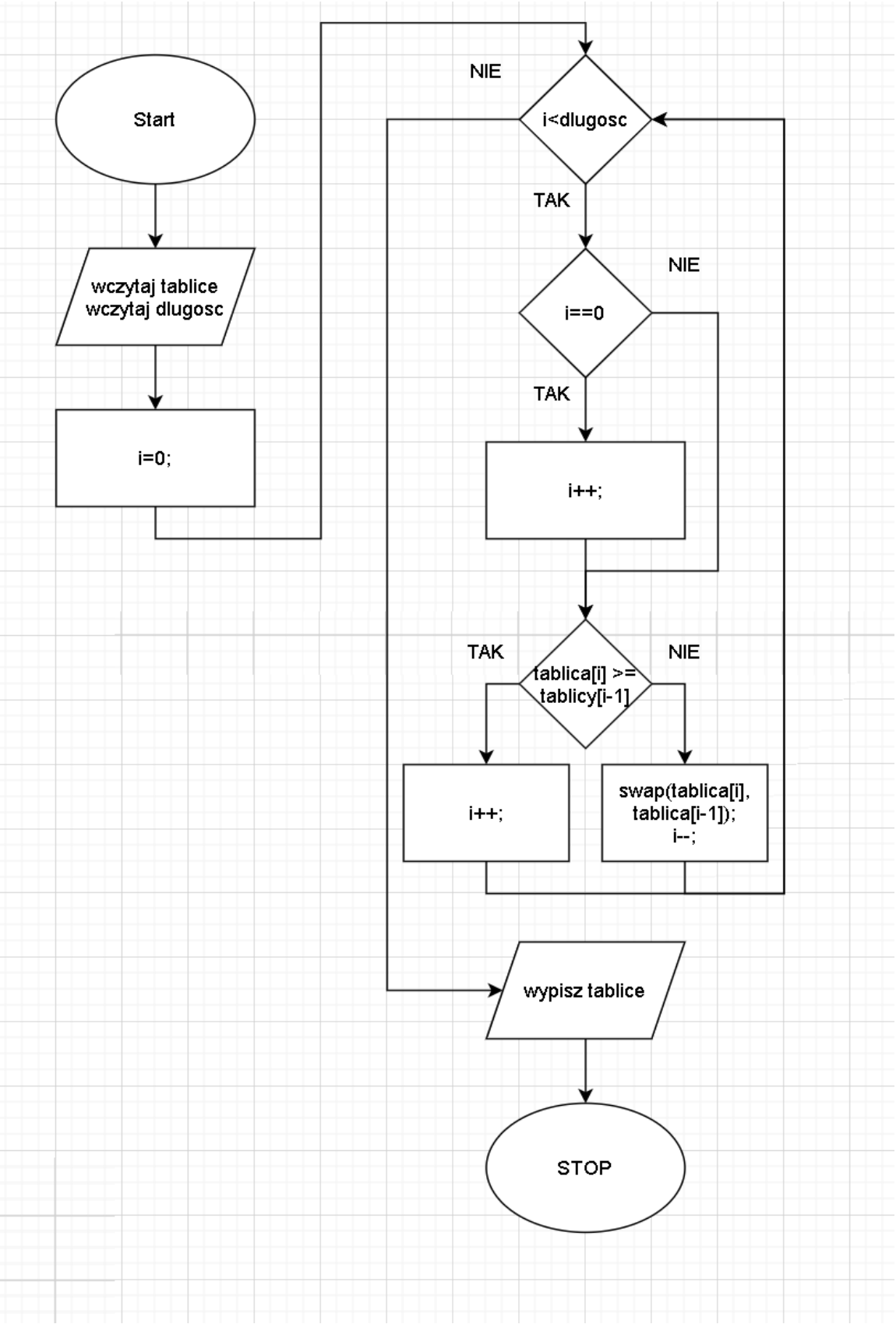
Zaczyna się od najprostszej funkcji wyświetlania tablicy, przestała być użyteczna w momencie w który wszystko zacząłem zapisywać do plików. Napisałem funkcje do generowania tablic, która generuje tablice o wybranej długości i o wyznaczonym możliwym najmniejszym i największym elemencie. Dzięki temu mogę jak chce kontrolować dane, które wygeneruje program. Później funkcja, która kopiuje tablice, używana na początku algorytmów by oba niezależnie sortowały taką samą tablice. Po tym oba algorytmy, które zwracają element posortowanej tablicy o określonym indeksie, by umożliwić łatwe wyświetlanie. W tym miejscu mamy już dwie główne funkcje: Testy i Posortuj tablice. Funkcja testy przyjmuje dużo wejściowych danych, które są tak naprawdę potrzebne do wywoływania funkcji do generowania tabeli, wybieramy od jakiej do jakiej wartości i z jakim skokiem mają się zmieniać długość, najmniejszy i największy element. Trzy fory do których wchodzą te dane odpowiadają za przejście po kolei po wszystkich kombinacjach danych. Generujemy tablice, po czym wywołujemy oba algorytmy mierząc ich czas działania. Zapisujemy wszystko do pliku tak, by łatwo później w excelu zrobić z tego wykresy. Funkcja posortuj tablice służy do wczytania z pliku tablicy o

określonej długości , ją też sortujemy oboma algorytmami i mierzymy ich czas. Wszystko ładnie zapisujemy do pliku. Na końcu mamy main w którym wywołujemy tylko testy z przykładowymi parametrami i posortuj tablice na przykładowym pliku z tablicą.

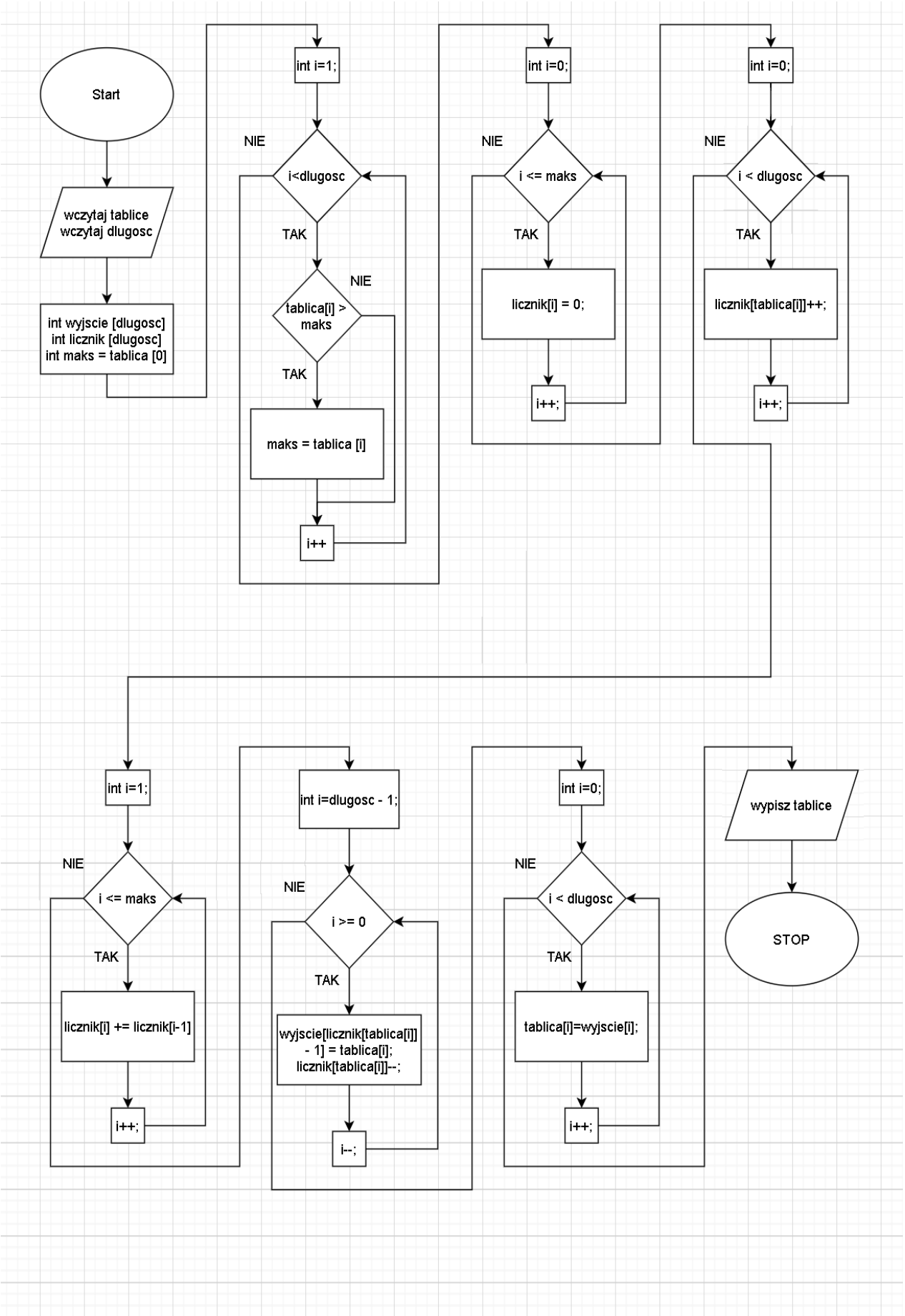
Dane wygenerowane przez program importowałem do excela, w którym generowałem wykresy, które pojawią się później w sprawozdaniu.

5. Schemat blokowy

5.1. Schemat Sortowania Gnoma



5.2. Schemat Sortowania przez Zliczanie



6. Pseudokod

6.1. Sortowanie Gnoma

Wczytujemy tablice i jej długość

Ustawiamy **i** na zero

Dopóki **i** jest mniejsza od długości tabeli to

Jeżeli **i** jest równe zero, zmień jego wartość na jeden

Jeżeli **i**-ty element tablicy jest większy lub równy **i** minus jeden elementowi tablicy to

Zwiększ **i** o jeden

Jeżeli nie to

Zamień **i**-ty element tablicy z **i** minus jeden elementem tablicy

Zmniejsz **i** o jeden

Wypisz tablice

6.2. Sortowanie przez zliczanie

Wczytujemy tablice i jej długość

Tworzymy tablice wyjście o długości tablicy sortowanej

Tworzymy tablice licznik o długości tablicy sortowanej

Maks równe zerowemu elementowi tablicy

i równe jeden, powtarzaj póki **i** mniejsze od długości

Jeżeli **i**-ty element tablicy jest większy od maksa

Maks równa się **i**-temu elementowi tablicy

Zwiększ **i** o jeden

i równe zero, powtarzaj póki **i** mniejsze lub równe maks

i-ty element licznika równy zero

Zwiększ **i** o jeden

i równe zero, powtarzaj póki **i** mniejsze długości

element licznika o indeksie równym **i**-temu elementowi tablicy zwiększamy o jeden

Zwiększ **i** o jeden

i równe jeden, powtarzaj póki **i** mniejsze lub równe maks

i-ty element licznika równa się **i**-temu elementowi licznika dodanemu do **i** minus jeden elementu licznika

Zwiększ **i** o jeden

i równe długości minus jeden, powtarzaj póki **i** większe lub równe zeru

element wyjścia o indeksie równym elementowi licznika o indeksie równym **i**-temu elementowi tablicy minus jeden równy **i**-temu elementowi tablicy

element licznika o indeksie równym **i**-temu elementowi tablicy zmniejszamy o jeden

Zmniejsz **i** o jeden

i równe zero, powtarzaj póki **i** mniejsze od długości

i-ty element tablicy równy **i**-temu elementowi wyjścia

Zwiększ **i** o jeden

Wypisz tablice

7. Złożoność obliczeniowa

7.1. Sortowanie Gnoma

Średnia złożoność obliczeniowa to $O(n^2)$, jest to też pesymistyczna złożoność. Optymistyczna złożoność, czyli kiedy dane już są prawie posortowane to O(n).

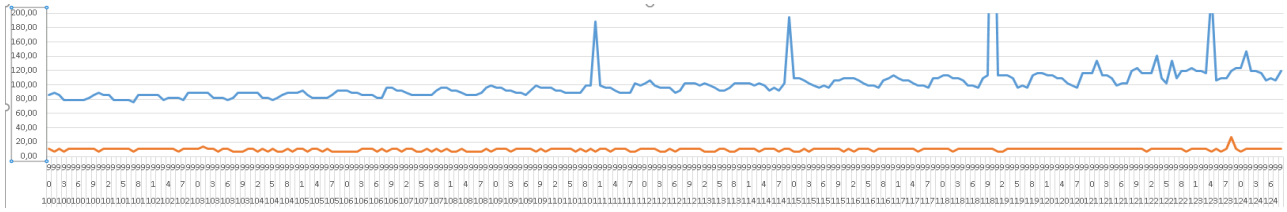
7.2. Sortowanie przez Zliczanie

Złożoność obliczeniowa w wszystkich przypadkach wynosi O(n+k),k oznacza rozpiętość danych, równą powiększonej o 1 różnicy między maksymalną a minimalną wartością. Złożoność się nie zmienia, bo nie ma znaczenia jak ułożone są dane, algorytm zawsze przejdzie po wszystkich tak samo.

8. Testy

8.1. Test pierwszy

Długość od 100 do 125, skok 1; dolna granica od 0 do 9, skok 1; górna granica równa 9, stała;

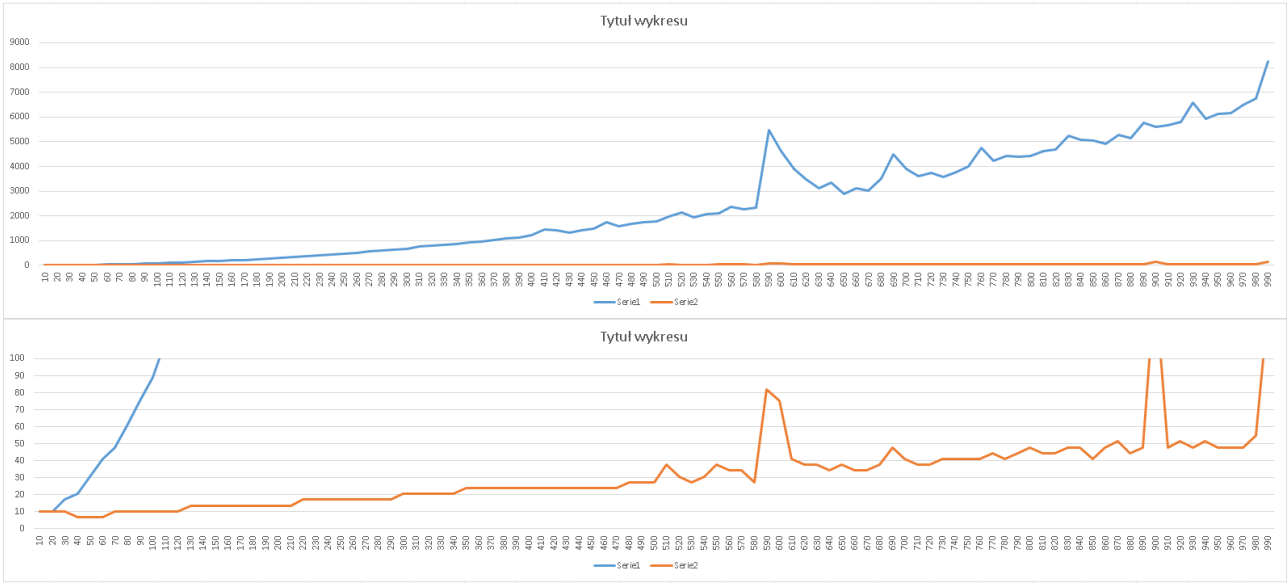


Od razu możemy zauważyć, że sortowanie gnoma zajmuje znacznie dłużej, jego czas działania zmniejsza się za to, gdy rosła dolna granica, co jest zrozumiałe, bo im mniej różnych elementów tym

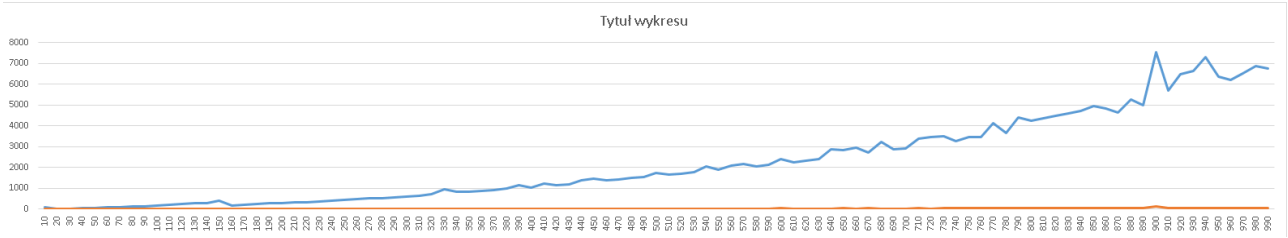
bardziej są ułożone po kolei. Sortowanie przez zliczanie utrzymuje mniej więcej taki samy poziom, ale też zmniejsza się jego czas działania wraz z wzrostem dolnej granicy, no zmniejsza się omówione w złożoności k.

8.2. Test drugi

Długość od 10 do 1000, skok 10; dolna granica 0, stała; górna granica 100, stała;

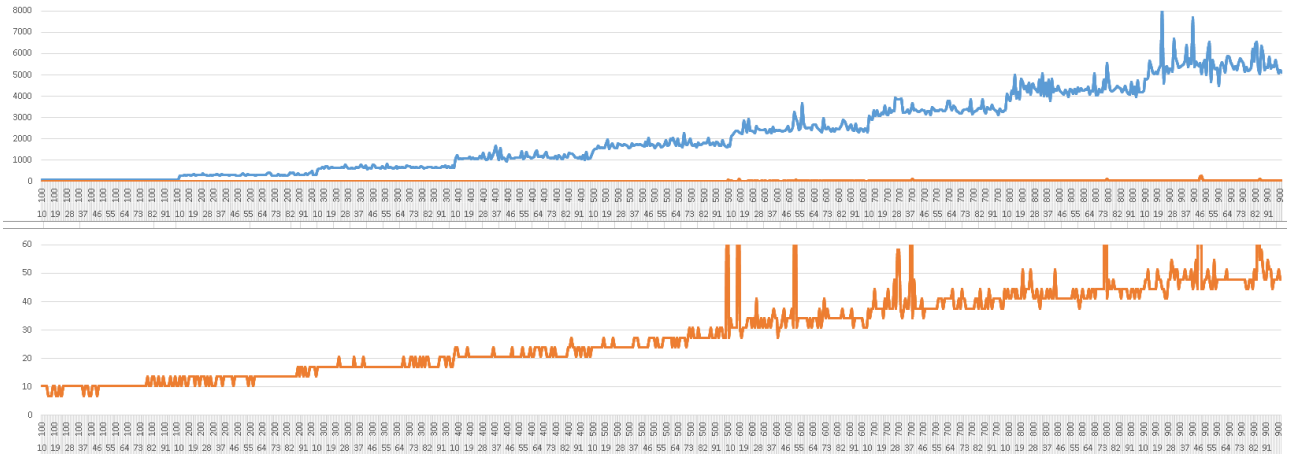


Tym razem zwiększamy tylko N, by zobaczyć jak rośnie czas w obu algorytmach. Sortowanie gnoma rośnie o wiele szybciej i z czasem staje się nawet do 130 razy większe. Z tymi samymi parametrami wykonałem jeszcze kolejny test i wyniki były podobne:



8.3. Test trzeci

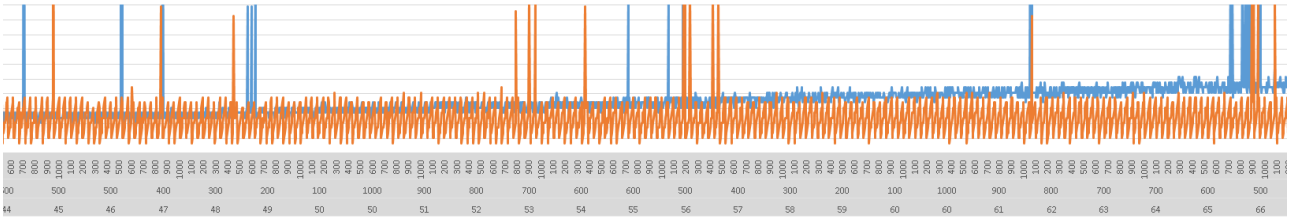
Długość od 100 do 900, skok 100; dolna granica 0, stała; górna granica od 1 do 99, skok 1;



Widać tutaj swego rodzaju „schodki”, ale wynika to tylko z długości tablic, co jest jednak zaskakujące nie widać jasnego wzrostu w czasie w poszczególnych schodkach w przypadku sortowania przez zliczanie na podstawie tego, że rośnie górna granica. Liczby dalej jednak są przypadkowe, wiec zwiększanie górnej granicy nie oznacza jednoznacznie większej rozbieżności liczb.

8.4. Test czwarty

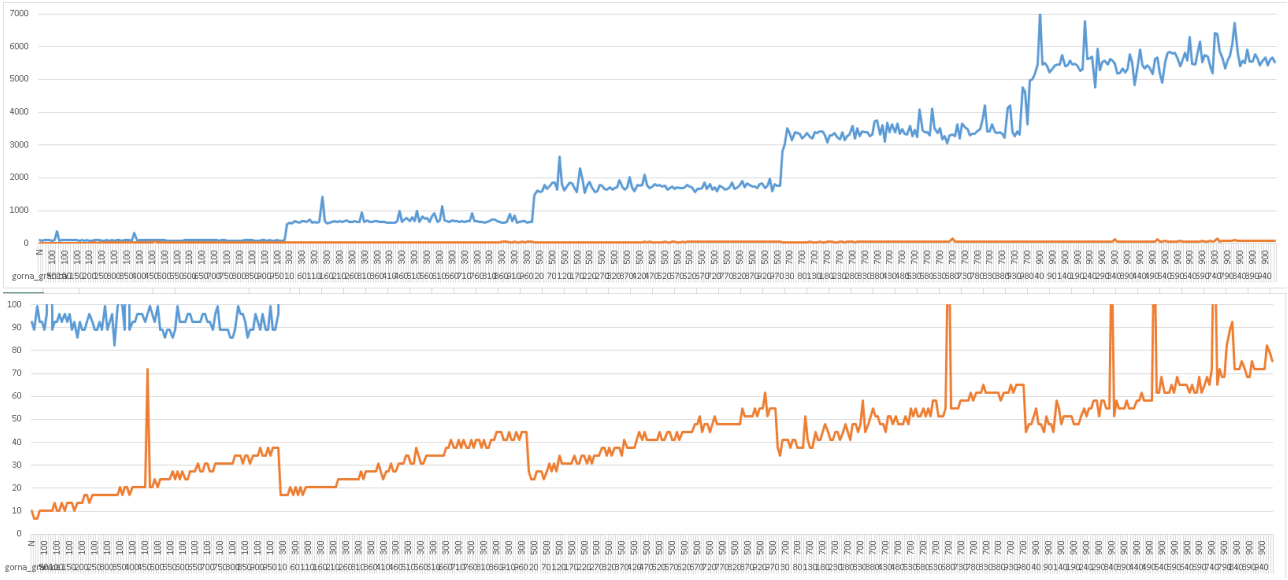
Długość od 10 do 100, skok 1; dolna granica od 100 do 1000, skok 100; górna granica od 100 do 1000, skok 100;



Tutaj tylko część wykresu, bo było tak dużo danych. Chciałem zauważyć tylko, że przy takiej rozbieżności danych na początku dłużej działa sortowanie przez zliczanie, jednak już koło długości 50 sortowanie gnoma zaczyna rosnąć szybciej i szybko znowu dominuje dalszą czas wykresu.

8.5. Test piąty

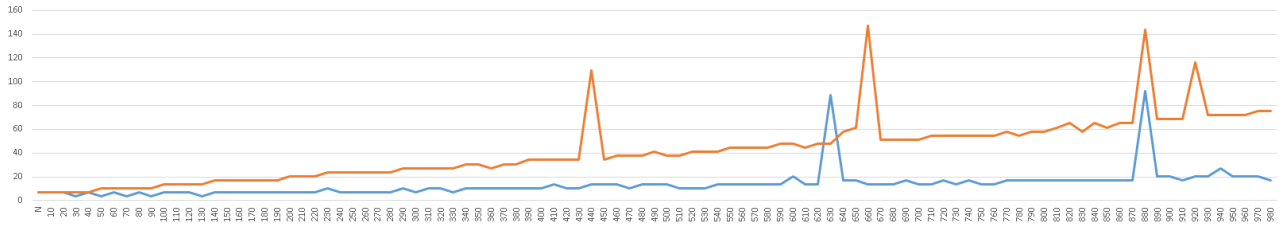
Długość od 100 do 900, skok 200; dolna granica 0, stała; górna granica od 10 do 1000, skok 10;



Schodki znowu wynikają z przeskoku długości o 200, tak samo jak wcześniej sortowanie gnoma jest o wiele dłuższe. Co jest tutaj ciekawe, to to, że sortowanie gnoma tworzy dość równą poziomą linie, sortowanie przez zliczanie jednak tworzy linie ukośną, co zgadzało by się z tym że rośnie ona proporcjonalnie z rozbieżnością danych wynikającą z wzrostu górnej granicy.

8.6. Test optymistyczny dla gnoma/ pesymistyczny dla zliczania

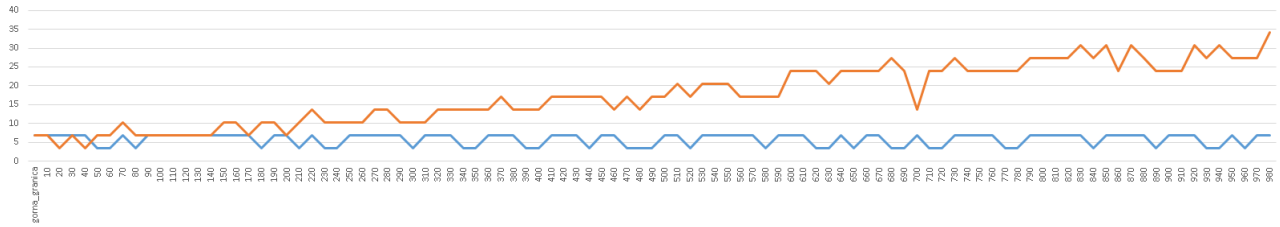
Dane spreparowane, wygenerowane przez inny program. Wypełniłem tablice indeksami elementów, przez co tablica jest zawsze posortowana, a długość i rozbieżność danych rośnie ze stałą prędkością.



Obie funkcje rosną, ale sortowanie przez zliczanie robi to widocznie szybciej, szczególnie jeśli przypomnimy sobie, że na większości poprzednich wykresów był on praktycznie poziomą linią na dole. Oczywiście trudno mówić o sensie sortowania posortowanych danych.

8.7. Test pesymistyczny dla zliczania

Długość 10, stała; dolna granica 0, stała; górna granica od 10 do 1000, skok 10;



Nie jest to test optymistyczny dla gnoma, bo dane nie są posortowane. Wystarczy to jednak, żeby sortowanie przez zliczanie zajęło więcej czasu. Dzieje się tak, bo nie rośnie długość, która jest zawarta w złożoności obliczeniowej sortowania gnoma, ale rośnie rozbieżność danych zawarta w złożoności sortowania przez zliczania.

9. Wnioski i podsumowanie

Dane i wykresy sporządzone na ich podstawie dają nam jasny werdykt. Sortowania gnoma trwa o wiele za długo, by sensownie użyć go do sortowania dużych danych, jeśli jednak dane są wystarczająco małe, lub już wstępnie prawie posortowane, może się on sprawdzić. Sortowanie przez zliczanie radzi sobie lepiej z nawet ogromną ilością całkowicie losowych danych, jeśli jednak mam zbiór wypełniony danymi, które się nie powtarzają, to już lepiej użyć sortowania gnoma. Sortowanie gnoma jest bardzo nieskomplikowane i intuicyjne, zrozumienie sortowania przez zliczanie wymaga jednak chwili zastanowienia i porządnego wyjaśnienia, widać jednak że w praktyce bardziej skompilowany algorytm zadziała lepiej, bo to co potrafi wykonać algorytm gnoma, możemy zrobić sami nawet w głowie czy na kartce, algorytm zliczania natomiast pozwoli nam „przemielić" dużo danych.

Załącznik-kod programu

```
//Program do analizy dwoch algorytmow sortujacych: Sortowanie Gnoma i Sortowanie przez Zliczanie.


#include <iostream>
#include <algorithm>
#include <ctime>
#include <fstream>
#include <chrono>
#include <iomanip>
using namespace std::chrono;
using namespace std;


////////////////////////////////////////////////////////////////////////////////////////////////////


//Wyswietlanie tablicy - funkcja pomocnicza, przydawala sie podaczas testowania manualnego algorytmow
void wyswietl_tablice (int tablica[], int dlugosc){
cout<<"tablica: [ ";
   for (int j=0; j<dlugosc; j++){
     cout<<tablica[j]<<" ";}
   cout<<"]"<<endl;}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////

//Generuje tablice, w ktorej mozna zmieniac zarowno gorna jak i dolna granice
void generuj_tablice(int tablica[], int dlugosc, int dolna_granica, int gorna_granica){
   srand((unsigned) time(0));
   for (int j=0; j<dlugosc; j++){
     tablica[j]=(dolna_granica+rand() % gorna_granica ); if(tablica[j]>gorna_granica){tablica[j]-
=dolna_granica;}}
  //cout<<"Stworzona ";
  //wyswietl_tablice(tablica, dlugosc);
}

////////////////////////////////////////////////////////////////////////////////////////////////////

//Kopiuje tablice, zeby moc dzialac na jednej tablicy pare razy, bedzie urzywana na poczatku algorytmow by
oba dzialaly na tej samej tablicy
void kopiowanie_talbicy(int tablica[], int tablica_kopia[], int dlugosc){
 for(int i=0; i<dlugosc; i++){
  tablica_kopia[i]=tablica[i];
 }
}

////////////////////////////////////////////////////////////////////////////////////////////////////

//Algorytm - sortowanie gnoma
int sortowanie_gnoma (int tablica[], int dlugosc, int h) {

int tablica_kopia[dlugosc];
kopiowanie_talbicy(tablica, tablica_kopia, dlugosc);
int i = 0;
while (i < dlugosc) { //przechodzimy po kolei tablice
   if (i == 0){i++;} //jesli wrocimy sie do 0 przechodzimy znowu tablice
```

```
    if (tablica_kopia[i] >= tablica_kopia[i - 1]){
     i++; //jesli elementy sa w odpowiedniej kolejnosci idziemy dalej
    }else{
     swap(tablica_kopia[i], tablica_kopia[i - 1]); //jesli elementy nie sa w odpowiedniej kolejnosci zamieniamy je i wracamy sie do poprzednich elementow by zobaczyc czy nie sa wieksze od zamienionego
    i--;
    }
}

  //cout<<"Posortowana sortowaniem gnoma ";  //odkomentowac jesli chcemy wyswietlac na ekranie
  //wyswietl_tablice(tablica_kopia, dlugosc);

  return tablica_kopia[h]; //zwracamy element tablicy o okreslonym indeksie, by ulatwic wyswietlanie
}

////////////////////////////////////////////////////////////////////////////////////////////////////

//Algorytm - sortowanie przez zliczanie
int sortowanie_przez_zliczanie(int tablica[], int dlugosc,  int gorna_granica, int h) {

 int tablica_kopia[dlugosc];
 kopiowanie_talbicy(tablica, tablica_kopia, dlugosc);
  int wyjscie[dlugosc]; //tworzymy pomocnicza tablice, ktora posluzy do wpisania danych do oryginalnej tablicy

  int maks = tablica_kopia[0]; //szukamy maksymalnej wartosci w tablicy
  for (int i = 1; i < dlugosc; i++) {
    if (tablica_kopia[i] > maks)
      maks = tablica_kopia[i];
  }

  int licznik[maks+1]; //tablica ktora bedzie zliczac ile razy wystepuje kazdy element
  for (int i = 0; i <= maks; i++) { //wypelniamy ja zerami
```

```
   licznik[i] = 0;

  }


  for (int i = 0; i < dlugosc; i++) { //w tablicy licznik na miejscu o indeksie rownym kazdej wartosci zliczamy ile razy wystepuje ta wartosc

   licznik[tablica_kopia[i]]++;

  }


  for (int i = 1; i <= maks; i++) { //znajdujemy faktyczna pozycje wartosci w tablicy wyjsciowej

   licznik[i] += licznik[i - 1];

  }


  for (int i = dlugosc - 1; i >= 0; i--) {

   wyjscie[licznik[tablica_kopia[i]] - 1] = tablica_kopia[i]; //wpisujemy wartosc w miejsce z indeksem rownym odpowiadajacemu mu numerowi w liczniku minus jeden

   licznik[tablica_kopia[i]]--; //zmniejszamy numer w liczniku, przez co wypiszemy odpowiednia ilosc takich samych wartosci

  }


  for (int i = 0; i < dlugosc; i++) { //zapisujemy wartosci z wyjscia do oryginalnej tablicy, potrzebowalismy jej do tej pory dlatego mielismy talbice pomocnicza wyjscia

   tablica_kopia[i] = wyjscie[i];

  }


  //cout<<"Posortowana sortowaniem przez zliczanie "; //odkomentowac jesli chcemy wyswietlac na ekranie

  //wyswietl_tablice(tablica_kopia, dlugosc);


  return tablica_kopia[h]; //zwracamy element tablicy o okreslonym indeksie, by ulatwic wyswietlanie

}


////////////////////////////////////////////////////////////////////////////////////////////////


//Funkcja do testowania, ma du¿o parametrow by moc dokladnie sprawdzic w jakich warunkach najlepiej i najgorzej dzialaja algorytmy
```

```
void testy(int dlugosc_od, int dlugosc_do, int skok_dlugosci, int dolna_granica_od, int dolna_granica_do, int skok_dolnej,  int gorna_granica_od, int gorna_granica_do, int skok_gornej){


fstream output;

output.open("Testy.txt", ios::out); //otwieramy plik do ktorego bedziemy zapisywac dane


output<<"Gnome Zliczanie dlugosc dolna_granica gorna_granica"<<endl; //nazwy kolum do pliku w excelu w ktorym bede generowal wykresy


//ustalamy jak beda wygladac tabele, by moc stworzyc taki zestaw danych, jaki bedzie nam potrzebny

for (int dlugosc=dlugosc_od; dlugosc<=dlugosc_do; dlugosc+=skok_dlugosci){ //wybieramy od jakiej do jakiej dlugosci beda generowane tablice, oraz o ile beda sie te dlugosci zmieniac

    for (int dolna_granica=dolna_granica_od; dolna_granica<=dolna_granica_do; dolna_granica+=skok_dolnej){ //wybieramy jaka wartosc moze miec najmniejszy element, przydaje sie to jesli chcemy miec wieksze wartosci w tablicach

      for (int gorna_granica=gorna_granica_od; gorna_granica<=gorna_granica_do; gorna_granica+=skok_gornej){ //wymieramy jaka moze byc najwieksza wartosc, tego glownie urzywalem by zwiekszyc roznorodnosc wartosci


int tablica[dlugosc];

generuj_tablice (tablica,dlugosc,dolna_granica,gorna_granica); //tworzymy tablice z odpowiednimi parametrami


steady_clock::time_point start ; //inicjujemy rzeczy potrzebne do liczenia czasu

steady_clock::time_point stop ;

std::chrono::duration<double> czas ;


start = steady_clock::now(); //mierzymy ile zajmuje posortowanie Gnomem

sortowanie_gnoma (tablica,dlugosc,0);

stop = steady_clock::now();

czas = stop-start;

output<<setprecision(10)<<(czas.count())*10000000<<" "; //zapisujemy ten czas


start = steady_clock::now(); //mierzymy ile zajmuje posortowanie przez Zliczanie

sortowanie_przez_zliczanie (tablica,dlugosc,gorna_granica,0);

stop = steady_clock::now();
```

```
czas = stop-start;

output<<setprecision(10)<<(czas.count())*10000000<<" "<<dlugosc<<" "<<dolna_granica<<"
"<<gorna_granica; //zapisujemy czas i parametry tabeli ktora sortowalismy


output<<" tablica: [ "; //zapisujemy jak wygladala tablica przed sortowanie, zeby zobaczy na podstawie
grafu ktore tablice byly problemem dla algorytmow

for (int j=0; j<dlugosc; j++){

output<<tablica[j]<<" ";}

output<<"]"<<endl;

}}}

output.close();

}


////////////////////////////////////////////////////////////////////////////////////////////////////


//Funkcja, ktora wczytuje tablice i sortuje ja oboma algorytmami

void posortuj_tablice(int dlugosc){


fstream input;

input.open("tablica.txt", ios::in); //otwieramy plik z ktorego wczytujemy tablice

fstream output;

output.open("Posortowana tablica.txt", ios::out); //otwieramy plik do ktorego wpiszemy posortowana
tablice


int tablica[dlugosc];

for(int i=0; i<dlugosc; i++){ //wczytujemy tablice z pliku

   input>>tablica[i];

}


output<<"Podana tablica: [ "; //zapisujemy tablice z pliku, by byc pewnym ze program dziala

for(int i=0; i<dlugosc; i++){

   output<<tablica[i]<<" ";

}

output<<"]"<<endl<<endl;
```

```
steady_clock::time_point start ; //inicjujemy rzeczy potrzebne do liczenia czasu

steady_clock::time_point stop ;

std::chrono::duration<double> czas ;


start = steady_clock::now(); //mierzymy ile zajmuje posortowanie Gnomem

sortowanie_gnoma (tablica,dlugosc,0);

stop = steady_clock::now();

czas = stop-start;

output<<"Sortowanie Gnoma:"<<endl //zapisujemy czas dzialania algorytmu i posortowana przez niego
tablice

<<"Czas dzialania: " <<setprecision(10)<<(czas.count())*10000000<<endl<<"Posortowana tablica: [ ";

for(int h=0; h<dlugosc; h++){

   output<<sortowanie_gnoma (tablica,dlugosc,h)<<" ";

}

output<<"]"<<endl<<endl;


start = steady_clock::now(); //mierzymy ile zajmuje posortowanie przez Zliczanie

sortowanie_przez_zliczanie (tablica,dlugosc,dlugosc,0);

stop = steady_clock::now();

czas = stop-start;

output<<"Sortowanie przez Zliczanie:"<<endl //zapisujemy czas dzialania algorytmu i posortowana przez
niego tablice

<<"Czas dzialania: "<<setprecision(10)<<(czas.count())*10000000<<endl<<"Posortowana tablica: [ ";

for(int h=0; h<dlugosc; h++){

   output<<sortowanie_przez_zliczanie (tablica,dlugosc,dlugosc,h)<<" ";

}

output<<"]"<<endl<<endl;


input.close();

output.close();

}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////

int main (){

testy(10,100,10,0,0,1,10,100,10); //wywolujemy testy z wybranymi parametrami

posortuj_tablice(20); //sortujemy tablice z pliku znajac jej dlugosc

return 0;
}
```