

Politechnika Rzeszowska
Wydział matematyki i fizyki stosowanej
Inżynieria i analiza danych, I rok

Jagoda Wolanin
nr indeksu: 166716

Sortowanie gnoma i sortowanie grzebieniowe

Sprawozdanie z projektu nr 2

Rzeszów, 2020

Spis treści

1. Wstęp	2
1.1. Sortowanie gnoma	3
1.2. Sortowanie grzebieniowe	3
2. Program	4
2.1. Treść zadania	4
2.2. Krótki opis programu	4
3. Algorytmy	5
3.1. Algorytm sortowania gnoma	5
3.1.1. Krótka charakterystyka	5
3.1.2. Pseudokod algorytmu	5
3.1.3. Schemat blokowy	6
3.2. Sortowanie grzebieniowe	7
3.2.1. Krótka charakterystyka algorytmu	8
3.2.2. Pseudokod algorytmu	8
3.2.3. Schemat blokowy	9
4. Opis doświadczenia i dokumentacja	10
4.1. Testowanie poprawności działania algorytmów	10
4.2. Działanie programu na liczbach pseudolosowych	11
4.3. Odczyt z pliku	12
4.4. Porównanie szybkości działania algorytmów	13
4.5. Efektywność algorytmów	16
4.5.1. Złożoność czasowa algorytmów	16
5. Podsumowanie i wnioski końcowe	17
6. Źródła	17
7. Spis ilustracji	17

1. Wstęp

1.1. Sortowanie gnomą

Sortowanie gnomą¹ (ang. *gnome sort*) – algorytm sortowania podobny do sortowania przez wstawianie. Różni go sposób przenoszenia danej na właściwe miejsce poprzez wielokrotne zamiany kolejności dwóch sąsiednich elementów, tak jak w sortowaniu bąbelkowym. Nazwa pochodzi od holenderskiego krasnala ogrodowego (niderl. *tuinkabouter*), który zamienia miejscami doniczki w ogrodzie.

Algorytm wyróżnia się prostotą, nie zawiera zagnieżdżonych pętli. Jego złożoność obliczeniowa to $O(n^2)$ w średnim przypadku, jednak zbliża się do $O(n)$, jeśli zbiór wejściowy jest prawie posortowany (tzn. niewielka liczba elementów jest na niewłaściwych miejscach, bądź wszystkie elementy są niedaleko swoich właściwych miejsc). W praktyce algorytm działa równie szybko jak algorytm sortowania przez wstawianie, chociaż wiele zależy od struktury programu i implementacji.

1.2. Sortowanie grzebieniowe

Sortowanie grzebieniowe² (ang. *combsort*) – wynaleziona w 1980 przez Włodzimierza Dobosiewicza, odkryta ponownie i opisana w 1991 roku przez Stephena Lacey'a i Richarda Boxa metoda sortowania tablicowego. Jej główne cechy to:

- oparta na metodzie bubblesort (sortowanie bąbelkowe)
- prawdopodobnie złożoność wynosi $O(n \log n)$, statystycznie gorsza niż quicksort (sortowanie szybkie)
- włączono empirię - współczynnik 1.3 wyznaczony doświadczalnie

wariant podstawowy:

- za rozpiętość przyjmuje się długość tablicy, dzieli się rozpiętość przez 1.3, odrzuca część ułamkową
- bada się kolejno wszystkie pary obiektów odległych o rozpiętość (jeśli są ułożone niemonotonicznie - zamienia się je miejscami)
- wykonuje się powyższe w pętli dzieląc rozpiętość przez 1.3 do czasu, gdy rozpiętość osiągnie wartość 1.

Gdy rozpiętość spadnie do 1 metoda zachowuje się tak jak sortowanie bąbelkowe. Tylko wtedy można określić, czy dane są już posortowane czy nie. W tym celu można użyć zmiennej typu bool, która jest ustawiana po zamianie elementów tablicy miejscami. Przerywane jest wykonywanie algorytmu, gdy podczas przejścia przez całą tablicę nie nastąpiła zamiana.

¹ Źródło: https://pl.wikipedia.org/wiki/Sortowanie_gnomą

² Źródło: https://pl.wikipedia.org/wiki/Sortowanie_grzebieniowe

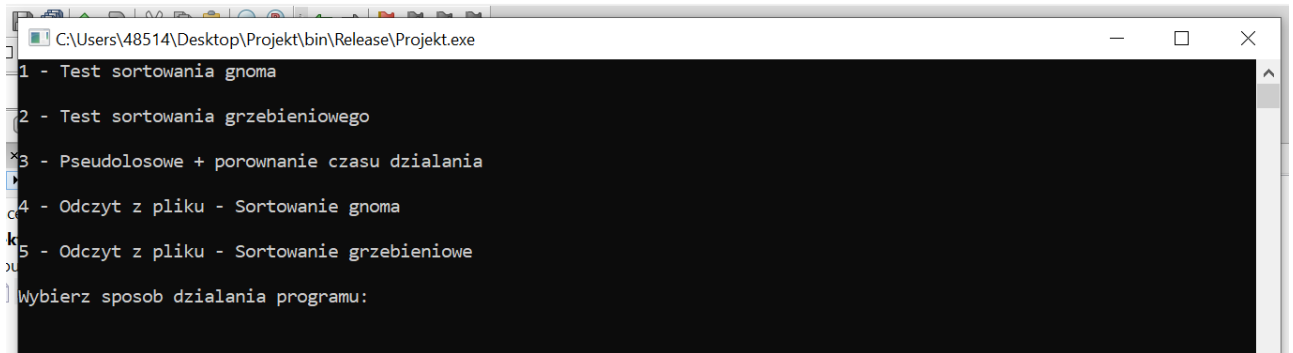
2. Program

2.1. Treść zadania

Zadanie. Zaimplementuj sortowanie gnoma oraz sortowanie grzebieniowe.






2.2. Krótki opis programu

W poniższym programie zostały zawarte dwa algorytmy sortowania – gnoma oraz grzebieniowe. Po uruchomieniu na konsoli pojawia się menu, w którym można wybrać sposób działania. Możliwości wyboru jest 5. Pierwsze dwa to testy algorytmów na wpisanych wcześniej tablicach. Pod numerem 3 znajduje się test na liczbach pseudolosowych. Dodatkowo w trzeciej opcji został dodany pomiar i porównanie czasów działania algorytmów. Można więc przeprowadzić test zarówno na małych tablicach jak i na tych bardzo dużych. Ostatnie dwa sposoby są przeznaczone do odczytu danych z pliku. Algorytmy pobierają dane i sortują podane liczby. Po każdym wyborze tworzy się plik tekstowy z rozwiązaniem o odpowiedniej nazwie do numeru sposobu (przykładowo: sposób 1 – utworzony plik ma nazwę „rozwiązanie1”).



```
C:\Users\48514\Desktop\Projekt\bin\Release\Projekt.exe
1 - Test sortowania gnoma
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnoma
5 - Odczyt z pliku - Sortowanie grzebieniowe
Wybierz sposob dzialania programu:
```

Rysunek 1 Widok konsoli po uruchomieniu programu

 rozwiązanie5	06.01.2021 15:36	Dokument tekstowy
 rozwiązanie4	06.01.2021 15:36	Dokument tekstowy
 rozwiązanie3	06.01.2021 15:18	Dokument tekstowy
 rozwiązanie2	06.01.2021 15:17	Dokument tekstowy
 rozwiązanie1	06.01.2021 15:17	Dokument tekstowy

Rysunek 2 Pliki tekstowe z rozwiązaniami

3. Algorytmy

3.1. Algorytm sortowania gnomu

```
13
14 // Funkcja sortująca algorytmem sortowania gnomu
15 void gnomeSort(int arr[], int n)
16 {
17
18     int index = 0;
19
20
21     while (index < n)
22     {
23         if (index == 0)
24             index++;
25         if (arr[index] >= arr[index - 1])
26             index++;
27         else
28         {
29             swap(arr[index], arr[index - 1]);
30             index--;
31         }
32     }
33     return;
34 }
35
```

Rysunek 3 Algorytm sortowania gnomu

3.1.1. Krótka charakterystyka

Algorytm sortowania gnomu wyróżnia się szczególną prostotą. Nie posiada żadnych zagnieżdżonych pętli. Zawiera funkcję sortującą oraz funkcję użytkową do stworzenia tablicy. Po przeanalizowaniu algorytmu można zaobserwować, że działa on zdecydowanie wolniej od sortowania grzebieniowego.³ W programie zostały zawarte testy poprawności działania. Sprawdzony został on na podanej w kodzie tablicy, danych z pliku oraz liczbach pseudolosowych. Podczas sortowania liczb pseudolosowych przeprowadzono pomiar czasu i porównanie szybkości z algorytmem sortowania grzebieniowego.

3.1.2. Pseudokod algorytmu

void gnomeSort arr[], n

int index ← 0

dopóki index < n

jeżeli index == 0 to

index zwiększamy o 1

jeżeli arr[index] >= arr[index - 1] to

index zwiększamy o 1

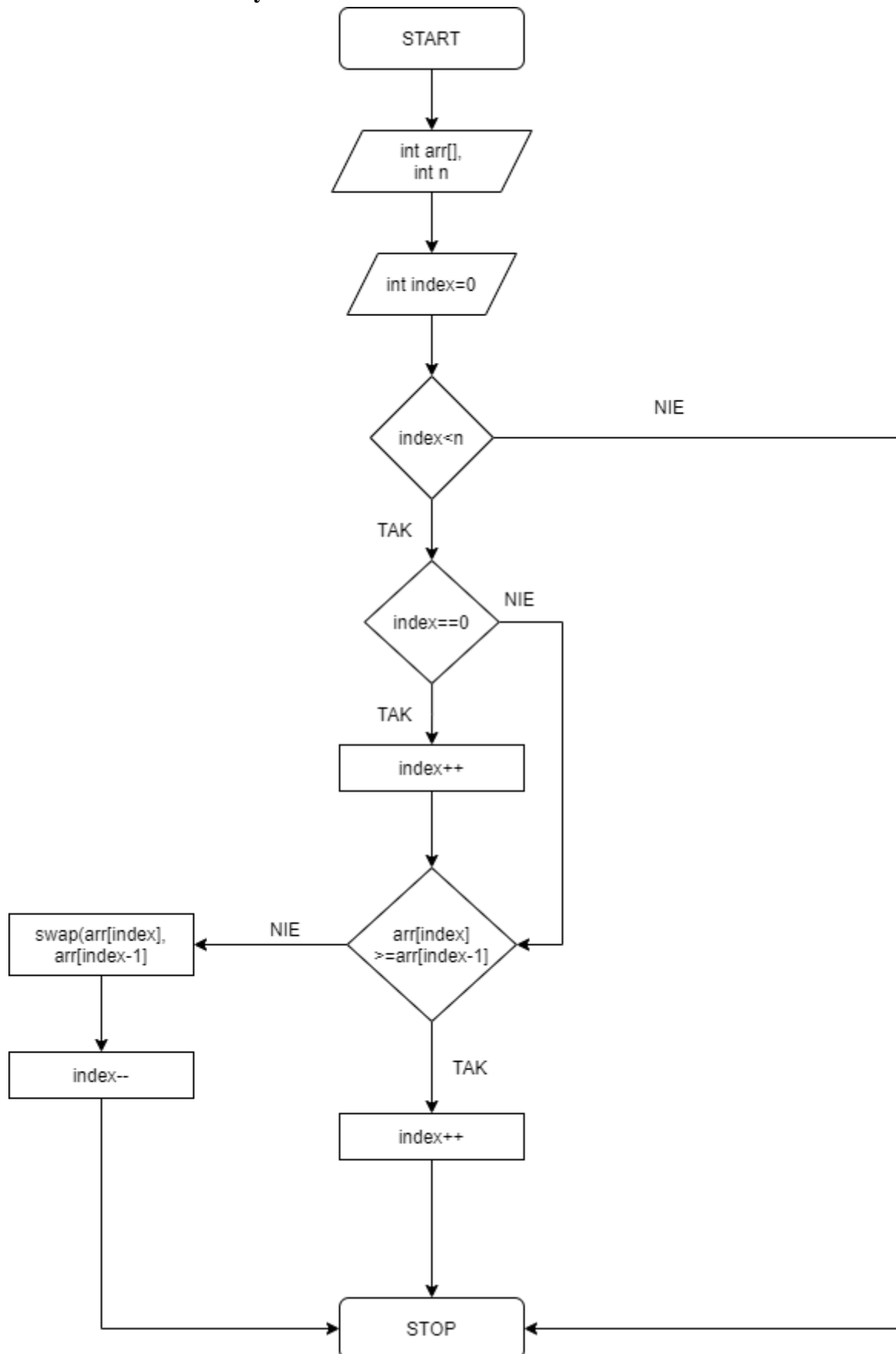
w inny przypadku

³ Porównanie danych w rozdziale 4.4

zamień $\text{arr}[\text{index}]$, $\text{arr}[\text{index}-1]$

index zmniejszamy o 1

3.1.3. Schemat blokowy



3.2. Sortowanie grzebieniowe

```
36 //Szukanie rozpietosci pomiedzy elementami
37 int getNextGap(int gap)
38 {
39     // Podzielenie rozpietosci przez 1.3
40     //co odrzuca czesc ulamkowa
41     gap = (gap*10)/13;
42
43     if (gap < 1)
44         return 1;
45     return gap;
46 }
47
48 // Funkcja sortujaca algorytmem sortowania grzebieniowego
49 void combSort(int a[], int n)
50 {
51     // Inicjowanie rozpietosci
52     int gap = n;
53
54     // Zmienna do upewnienia siê, ze
55     // petla dziala
56     bool swapped = true;
57
```

Rysunek 4 Algorytm sortowania grzebieniowego 1

```
48 // Funkcja sortujaca algorytmem sortowania grzebieniowego
49 void combSort(int a[], int n)
50 {
51     // Inicjowanie rozpietosci
52     int gap = n;
53
54     // Zmienna do upewnienia siê, ze
55     // petla dziala
56     bool swapped = true;
57
58     // w momencie, gdy rozpietosc jest wieksza od 1
59     // iteracja powoduje zamiane elementow
60     while (gap != 1 || swapped == true)
61     {
62
63         gap = getNextGap(gap);
64
65         // Sprawdzenie, czy nastapila zamiana
66         swapped = false;
67
```

Rysunek 5 Algorytm sortowania grzebieniowego 2

```

68 // Porównanie wszystkich elementów z obecną rozpiętością
69 for (int i=0; i<n-gap; i++)
70 {
71     if (a[i] > a[i+gap])
72     {
73         swap(a[i], a[i+gap]);
74         swapped = true;
75     }
76 }
77 }
78 }
79 }
80 }
81

```

Rysunek 6 Algorytm sortowania grzebieniowego 3

3.2.1. Krótka charakterystyka algorytmu

Algorytm sortowania grzebieniowego zaliczany jest do niestabilnych. Charakteryzuje się znacznie większą złożonością od algorytmu sortowania gнома. Na początku ustala się rozpiętość pomiędzy elementami, następnie dzielona jest ona przez 1.3. Odrzucona zostaje część ułamkowa. Powstała w ten sposób rozpiętość ma znaczenie w każdej iteracji. Algorytm bowiem porównuje tylko pary liczb odległe od siebie o daną rozpiętość. Zawarta zostaje również funkcja sortująca. Ważnym elementem programu jest upewnienie się, że pętla działa. Dokonuje się tego za pomocą zmiennej typu bool. W momencie, gdy rozpiętość jest większa od 1, elementy zamieniają się miejscami. Kolejnym rokiem jest sprawdzenie, czy zamiana nastąpiła. Wszystkie elementy porównuje się z obecną rozpiętością. W tym przypadku również zostały przeprowadzone testy poprawności działania algorytmu. Menu programu daje możliwość wyboru testu. Sortowanie grzebieniowe może odbywać się na konkretnej tablicy, liczbach pseudolosowych oraz na danych z pliku. W przypadku drugiej opcji dodany został również pomiar czasu i porównanie z sortowaniem gнома.

3.2.2. Pseudokod algorytmu

getNextGap gap

gap \leftarrow (gap*10)/13

jeżeli gap<1

zwróć 1

zwróć gap

void Combsort a[], n

gap \leftarrow n

bool swapped \leftarrow prawda

dopóki gap!=1 || swapped == true

gap \leftarrow getNextGap(gap)

swapped \leftarrow fałsz

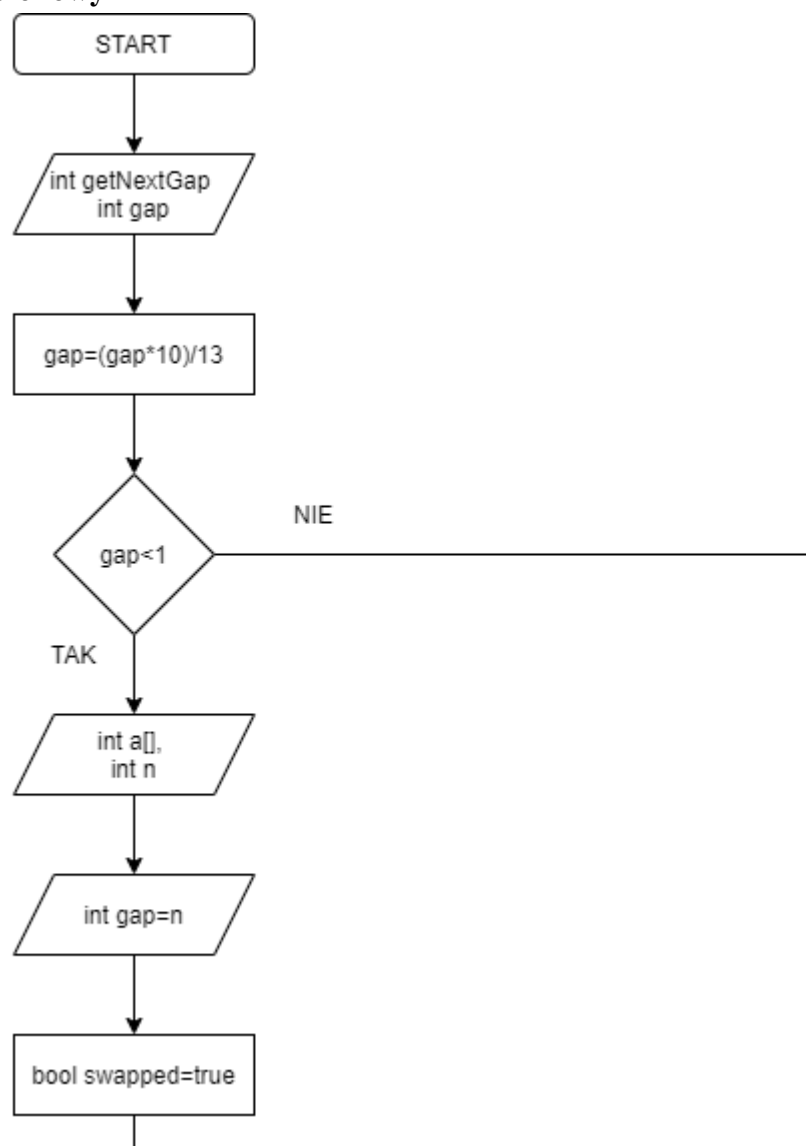
dla i \leftarrow 0; i<n-gap; i++ wykonuj

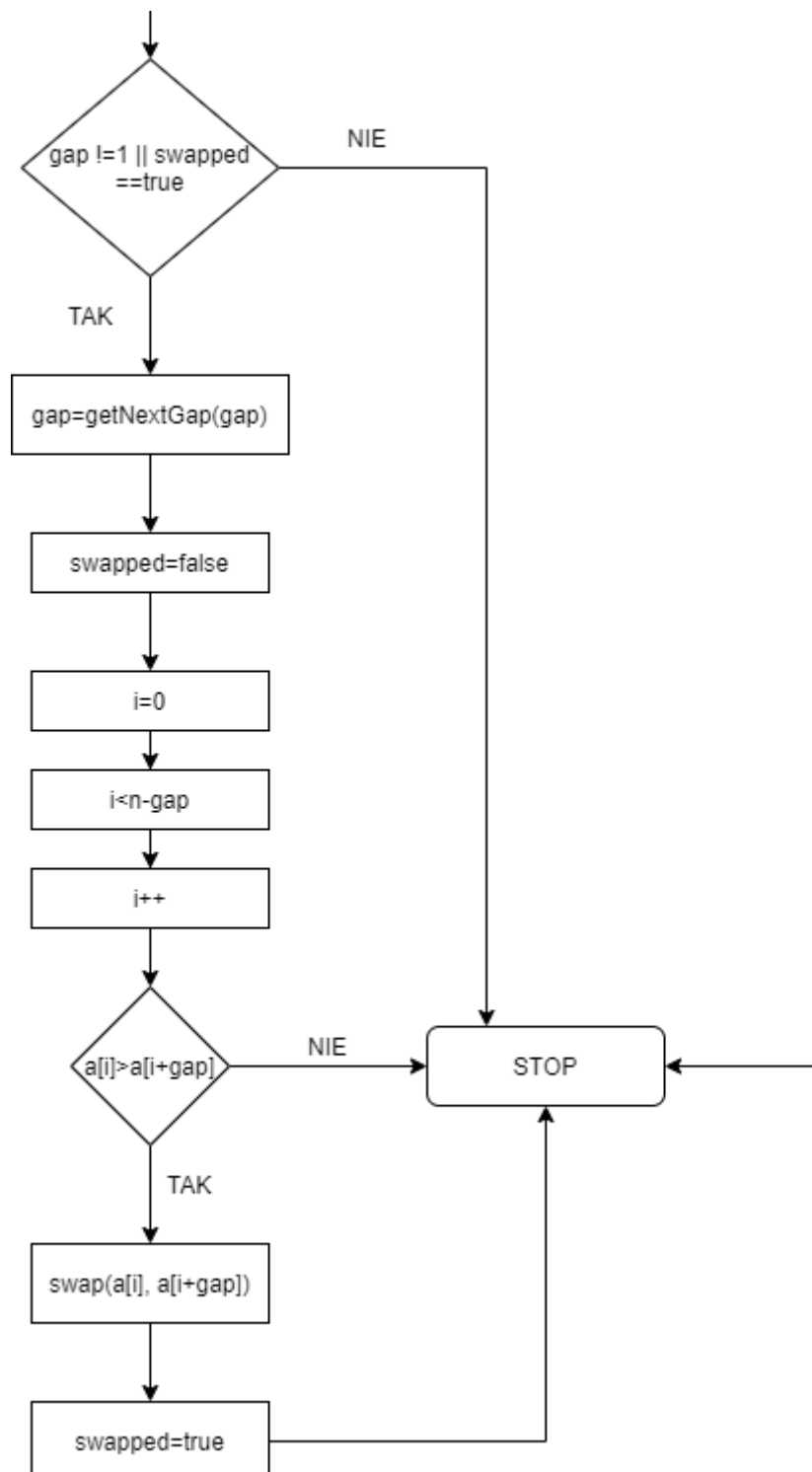
jeżeli a[i]>a[i+gap]

zamień a[i], a[i+gap]

swapped \leftarrow prawda

3.2.3. Schemat blokowy





4. Opis doświadczenia i dokumentacja

4.1. Testowanie poprawności działania algorytmów

Dla upewnienia się, czy algorytmy działają poprawnie pierwsze dwie opcje działania programu są przeznaczone do przetestowania ich. Testy 1 i 2 zostały wykonane na konkretnej tablicy, wpisanej wcześniej w kodzie. Na poniższych rysunkach można zauważyć, że ciąg został posortowany poprawnie w obu przypadkach.

```
"C:\Users\48514\Desktop\PROJEKT NR 2\Sortowanie gnomu i sortowanie grzebieniowe.exe"
1 - Test sortowania gnomu
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnomu
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 1
Posortowane liczby to: 0 1 2 3 3 4 6 7 10 12 21 21 22 23 32 32 33 33 77 88 97 112 123 234 643 811 812 1727 9998

Process returned 0 (0x0)  execution time : 1.163 s
Press any key to continue.
```

Rysunek 7 Test 1 - Sortowanie gnomu

```
"C:\Users\48514\Desktop\PROJEKT NR 2\Sortowanie gnomu i sortowanie grzebieniowe.exe"
1 - Test sortowania gnomu
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnomu
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 2
Posortowane liczby to: 0 1 2 3 3 4 6 7 10 12 21 21 22 23 32 32 33 33 77 88 97 112 123 234 643 811 812 1727 9998

Process returned 0 (0x0)  execution time : 1.183 s
Press any key to continue.
```

Rysunek 8 Test 2 - Sortowanie grzebieniowe


4.2. Działanie programu na liczbach pseudolosowych

W sposobie dzialania nr 3 zostala zaimplementowana funkcja generujaca ciagi pseudolosowe. Po wybraniu opcji trzeciej wyświetla się komunikat o konieczności podania rozmiaru tablicy. Rozmiar musi być liczbą naturalną, w innym przypadku na konsoli pojawi się komunikat o błędzie i prośba o podanie poprawnej liczby (rys.7). Następnie algorytmy rozpoczynają sortowanie, kiedy proces zostaje zakończony, na konsoli wyświetla się czas wykonania pracy każdego z nich.

```
"C:\Users\48514\Desktop\PROJEKT NR 2\Sortowanie gnomu i sortowanie grzebieniowe.exe"
1 - Test sortowania gnomu
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnomu
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 3
Podaj ilosc elementow w tablicy: -12
Bład! Proszę podać poprawna liczbę!
```

Rysunek 9 Komunikat o błędzie



```
"C:\Users\48514\Desktop\PROJEKT NR 2\Sortowanie gnoma i sortowanie grzebieniowe.exe"

1 - Test sortowania gnoma
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnoma
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 3
Podaj ilosc elementow w tablicy: 1000
Prosze czekac! Trwa sortowanie gnoma.

Czas sortowania gnoma: 0.006 s

Prosze czekac! Trwa sortowanie grzebieniowe

Czas sortowania grzebieniowego: 0 s

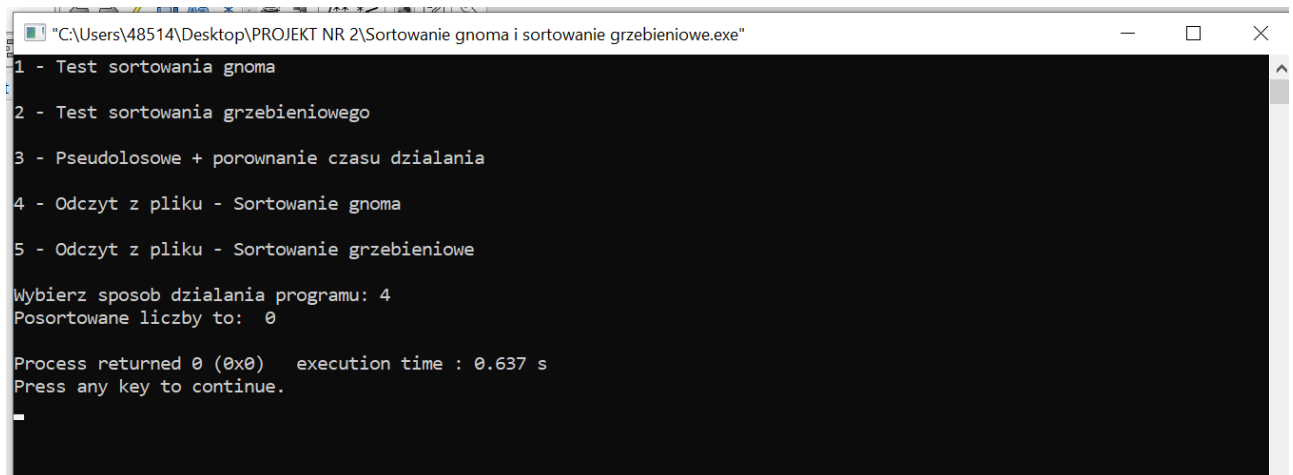
Process returned 0 (0x0)   execution time : 4.293 s
Press any key to continue.
```

4.3. Odczyt z pliku

Opcje 4 i 5 są przeznaczone do odczytu danych z pliku. Program pobiera dane z pliku tekstowego. Jeżeli plik zawiera dane, zostają one posortowane.

[illegible]

W przypadku braku danych w pliku, program wyświetli na ekranie 0.



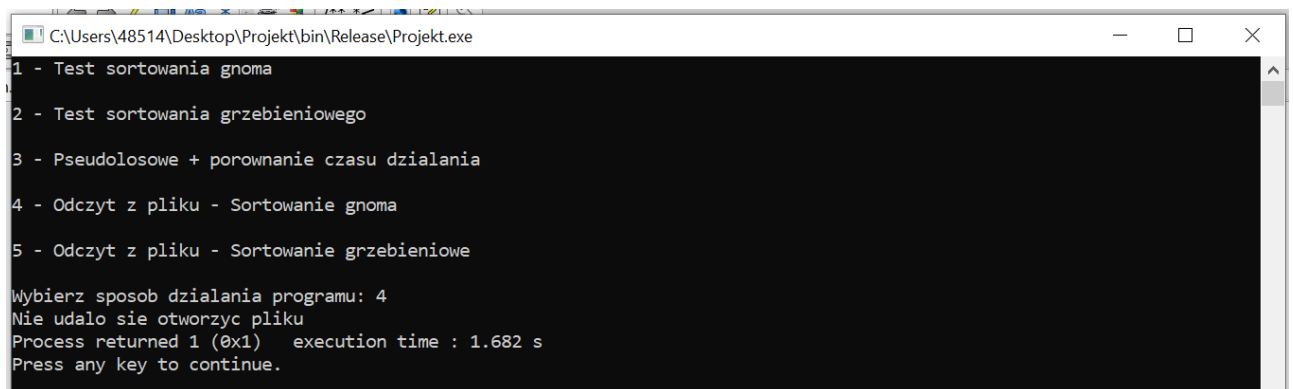
```
"C:\Users\48514\Desktop\PROJEKT NR 2\Sortowanie gnomu i sortowanie grzebieniowe.exe"
1 - Test sortowania gnomu
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnomu
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 4
Posortowane liczby to: 0

Process returned 0 (0x0) execution time : 0.637 s
Press any key to continue.
```

Rysunek 12 Brak danych w pliku

Kiedy program nie będzie mógł odczytać danych z pliku, na konsoli wyświetli się komunikat o błędzie.



```
C:\Users\48514\Desktop\Projekt\bin\Release\Projekt.exe
1 - Test sortowania gnomu
2 - Test sortowania grzebieniowego
3 - Pseudolosowe + porownanie czasu dzialania
4 - Odczyt z pliku - Sortowanie gnomu
5 - Odczyt z pliku - Sortowanie grzebieniowe

Wybierz sposob dzialania programu: 4
Nie udalo sie otworzyc pliku

Process returned 1 (0x1) execution time : 1.682 s
Press any key to continue.
```

Rysunek 13 Błąd odczytu

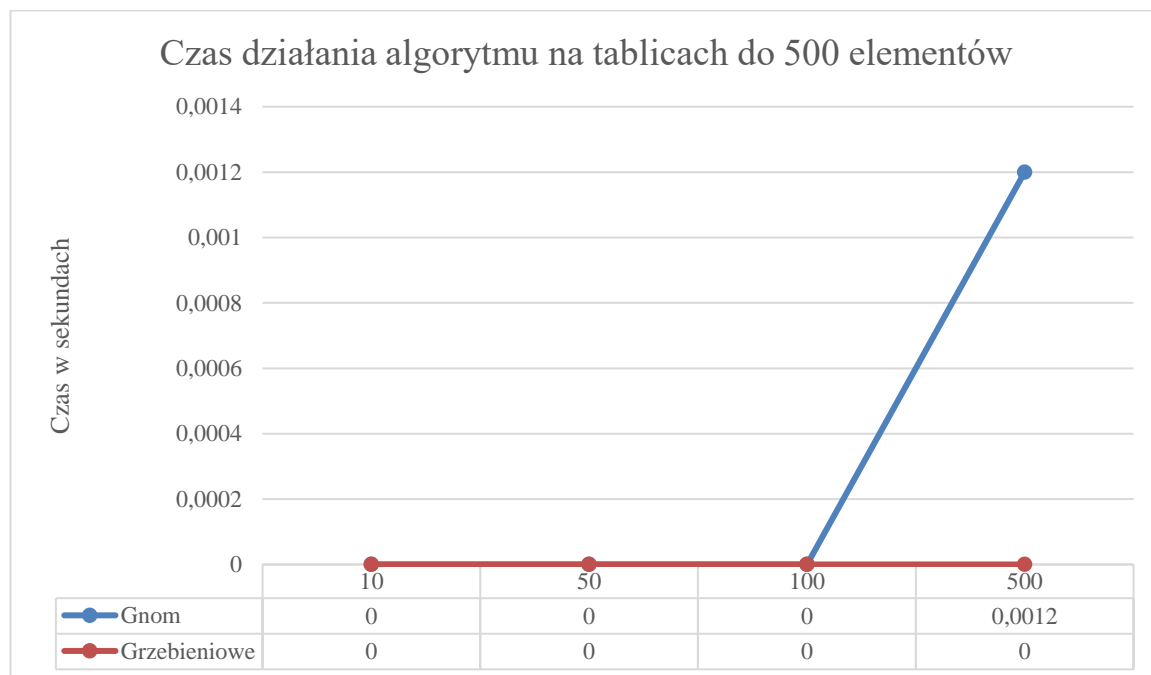
4.4. Porównanie szybkości działania algorytmów

Pomiary zostały wykonane za pomocą funkcji generującej ciągi pseudolosowe. Aby dobrze zobrazować i przeanalizować czas działania poszczególnych algorytmów podzielone są na 3 kategorie. Od „mniejszych” tablic, po te bardzo duże. Dla każdej wielkości tablicy wykonano 5 prób. Dane wpisane do tabeli posłużyły do obliczenia średniej, która została wpisana do wykresów. Poniżej każdej tabeli pomiarów zostały przedstawione wykresy zależności czasu działania od wielkości tablicy.

ROZMIAR TABLICY	10		50		100		500	
Sortowanie:	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort
Próba 1	0	0	0	0	0	0	0,001	0
Próba 2	0	0	0	0	0	0	0,001	0
Próba 3	0	0	0	0	0	0	0,001	0
Próba 4	0	0	0	0	0	0	0,001	0
Próba 5	0	0	0	0	0	0	0,002	0
Średnia prób	0	0	0	0	0	0	0,0012	0

Rysunek 14 Tabela pomiarów 1

Poniższy wykres obrazuje zależność czasu od tablic 10, 50, 100 i 500 elementowych.

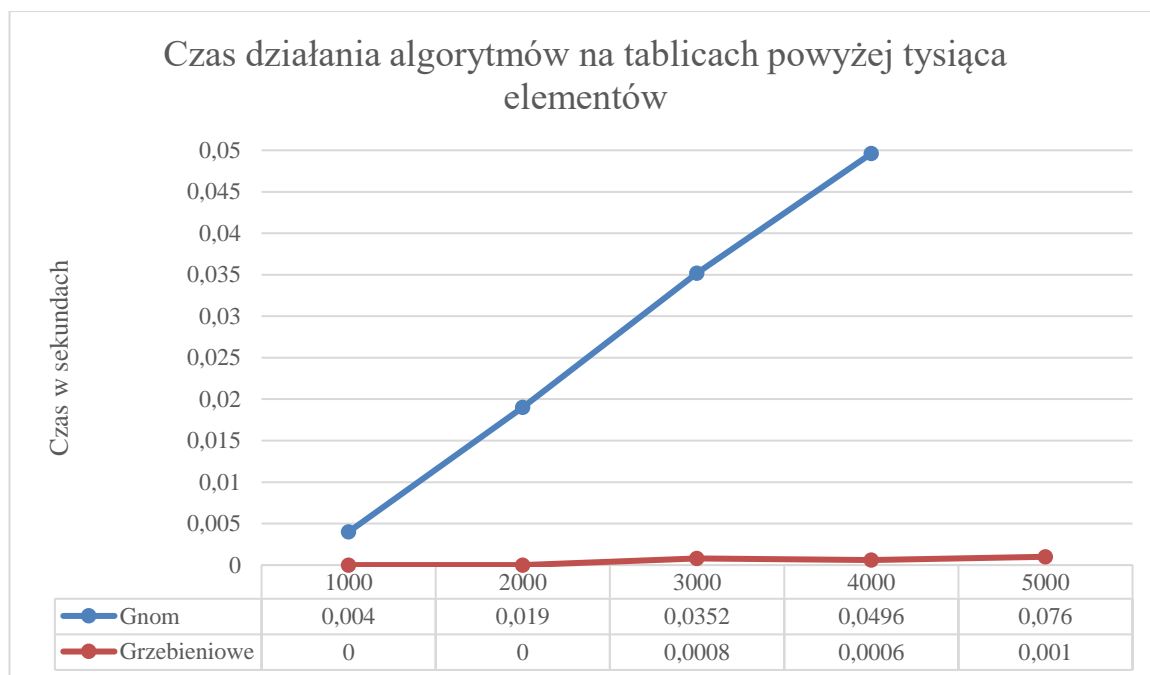


Rysunek 15 Wykres czasu działania na tablicach do 500 elementów

ROZMIAR TABLICY	1000		2000		3000		4000		5000	
Sortowanie:	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort
Próba 1	0,006	0	0,021	0	0,027	0,001	0,043	0,001	0,074	0,001
Próba 2	0,002	0	0,012	0	0,036	0,001	0,052	0	0,08	0,001
Próba 3	0,003	0	0,02	0	0,039	0,001	0,044	0,001	0,077	0,001
Próba 4	0,005	0	0,022	0	0,037	0	0,057	0	0,073	0,001
Próba 5	0,004	0	0,02	0	0,037	0,001	0,052	0,001	0,076	0,001
Średnia prób	0,004	0	0,019	0	0,0352	0,0008	0,0496	0,0006	0,076	0,001

Rysunek 16 Tabela pomiarów 2

Następnie zmierzony został czas na tablicach 1000, 2000, 3000, 4000, oraz 5000 – elementowych.



Rysunek 17 Wykres czasu działania na tablicach powyżej 5000 elementów

ROZMIAR TABLICY	100000		500000		1000000		1500000		2000000	
Sortowanie:	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort	gnomeSort	combSort
Próba 1	21,601	0,02	143,847	0,042	580,182	0,092	1315,34	0,152	2380,16	0,253
Próba 2	21,5	0,019	142	0,045	583,21	0,1	1310,323	0,15	2381,213	0,25
Próba 3	20,7	0,015	143,45	0,05	578,12	0,081	1317,23	0,16	2379,123	0,257
Próba 4	23,01	0,01	142,567	0,053	576,13	0,097	1318,23	0,159	2378,21	0,26
Próba 5	21,603	0,023	140,999	0,049	580,543	0,09	1320,123	0,161	2382,213	0,259
Średnia prób	21,6828	0,0174	142,5726	0,0478	579,637	0,092	1316,2492	0,1564	2380,184	0,2558

Rysunek 18 Tabela pomiarów 3

Na końcu wykonane zostały pomiary dla tablic 100000, 500000, 1000000, 1500000, 2000000 – elementowych. W tym przypadku najlepiej widać różnicę.



Rysunek 19 Wykres czasu działania na bardzo dużych tablicach

Warto zauważyć, że przy tablicach o małym rozmiarze czas sortowania jest minimalny. W przypadku sortowania grzebieniowego są to ułamki sekund. Ze względu na bardzo małe liczby, program zazwyczaj wyświetla czas działania algorytmu jako 0s. Dopiero przy sortowaniu tablic powyżej 100 tys. elementów sortowanie trwa wystarczająco długo, aby czas został odnotowany. Zupełnie inaczej jest w przypadku sortowania gnom. Ten algorytm jest znacznie wolniejszy. Przy małych tablicach sortuje równie szybko jak konkurent, jednak zmianę można zauważyć już przy tablicach 500-elementowych, gdzie w przypadku pierwszego algorytmu pojawia się czas powyżej 0s. Tablica z 2 mln. Elementów najlepiej obrazuje różnice w pracy. Sortowanie za pomocą algorytmu gnom trwało około 40 minut, natomiast algorytm sortowania grzebieniowego poradził sobie z tą tablicą w ciągu 0,25 sekundy.

4.5. Efektywność algorytmów

Efektywności algorytmu dzieli się na dwie złożoności:

- Czasową (jest to zależność między rozmiarem i porządkiem danych wejściowych, a czasem jego wykonania.),
- Pamięciową (jest to zależność pomiędzy rozmiarem i porządkiem danych wejściowych algorytmu, a jego zapotrzebowaniem na pamięć niezbędną do realizacji tegoż algorytmu).

4.5.1. Złożoność czasowa algorytmów

Mówiąc o złożoności czasowej algorytmów, należy zawsze rozpatrzyć 3 przypadki:

- Optymistyczny, gdy tablica wejściowa jest już posortowana lub prawie posortowana,
- Średni (oczekiwany), gdy algorytm pracuje na tablicach pseudolosowych,
- Pesymistyczny, gdy tablica wejściowa nie jest posortowana, a liczby są pomieszane.

W przypadku algorytmu sortowania grzebieniowego złożoność w każdym przypadku wynosi prawdopodobnie $O(n \log n)$, natomiast złożoność sortowania gnomą wynosi $O(n^2)$ w przypadku średnim, jednak zbliża się ona do $O(n)$, w przypadku optymistycznym.

5. Podsumowanie i wnioski końcowe

Aby dokładnie przeanalizować i zebrać dane, algorytmy zostały przetestowane na różnych tablicach. Od tych bardzo małych, po bardzo duże, nawet kilku milionowe. Na mniejszych działają równie dobrze i sprawnie. Zazwyczaj na konsoli wyświetla się liczba 0s. Jest to spowodowane minimalnym czasem wykonania pracy. Jednak w przypadku większych tablic algorytm sortowania gnomą działa wolniej i mniej efektywnie od konkurenta. Sortowanie grzebieniowe zdecydowanie lepiej radzi sobie z „większymi” ciągami. Tablice o małej liczbie elementów nie będą problemem dla żadnego z porównywanych algorytmów, jednak pracując na większych, zdecydowanie efektywniejszą opcją będzie wybór sortowania grzebieniowego.

6. Źródła

https://pl.wikipedia.org/wiki/Sortowanie_grzebieniowe

https://pl.wikipedia.org/wiki/Sortowanie_gnoma

<https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/>

<https://www.geeksforgeeks.org/comb-sort/>

<https://miroslawzelent.pl/kurs-c++/sortowanie-zlozonosc-algorytmow/>

<https://mattomatti.com/pl/fs10>

7. Spis ilustracji

Rysunek 1 Widok konsoli po uruchomieniu programu	4
Rysunek 2 Pliki tekstowe z rozwiązaniami	4
Rysunek 3 Algorytm sortowania gnomą	5
Rysunek 4 Algorytm sortowania grzebieniowego 1	7
Rysunek 5 Algorytm sortowania grzebieniowego 2	7
Rysunek 6 Algorytm sortowania grzebieniowego 3	8
Rysunek 7 Test 1 - Sortowanie gnomą	11
Rysunek 8 Test 2 - Sortowanie grzebieniowe	11
Rysunek 9 Komunikat o błędzie	11
Rysunek 10 Pseudolosowe + porównanie czasów	12
Rysunek 11 Odczyt z pliku	12
Rysunek 12 Brak danych w pliku	13
Rysunek 13 Błąd odczytu	13
Rysunek 14 Tabela pomiarów 1	13
Rysunek 15 Wykres czasu działania na tablicach do 500 elementów	14
Rysunek 16 Tabela pomiarów 2	14
Rysunek 17 Wykres czasu działania na tablicach powyżej 5000 elementów	15
Rysunek 18 Tabela pomiarów 3	15
Rysunek 19 Wykres czasu działania na bardzo dużych tablicach	16

