

Projekt

Algorytmy i struktury danych

Treść zadania:

„Zaimplementuj sortowanie przez scalanie oraz sortowanie metodą quicksort.”

Autor: Jakub Grasa

Inżynieria i analiza danych, semestr 2020/2021

Grupa projektowa: P02

Numer albumu: 166726

Prowadzący: dr. inż. prof. PRz Mariusz Borkowski

Spis treści

1. Zadanie	3
2. Sortowanie metodą quicksort	3
2.1. Informacje ogólne	3
2.2. Złożoność obliczeniowa	3
2.3. Schemat blokowy	4
2.4. Pseudokod	5
2.5. Implementacja C++	5
3. Sortowanie przez scalanie	6
3.1. Informacje ogólne	6
3.2. Złożoność obliczeniowa	6
3.3. Schemat blokowy	7
3.4. Pseudokod	8
3.5. Implementacja C++	10
4. Efektywność algorytmów	11
4.1. Przypadek pesymistyczny	11
4.2. Przypadek typowy (oczekiwany)	11
4.3. Przypadek optymistyczny	12
4.4. Przypadek mocno pesymistyczny dla algorytmu quickSort	12
5. Specyfikacja programu	13
6. Dokumentacja z doświadczeń	14
6.1. Niepoprawne dokonanie wyboru w menu początkowym:	14
6.2. Wprowadzanie danych z klawiatury:	14
6.2.1. Poprawne dane:	14
6.2.2. Niepoprawna długość tablicy, wprowadzenie znaku innego niż liczba:	15
6.2.3. Próba wprowadzenia innego elementu niż liczba do tablicy:	15
6.3. Wczytywanie danych z pliku	16
6.3.1. Poprawne dane w pliku:	16
6.3.2. Plik nie zawiera danych (plik pusty):	16
6.3.3. Próba otworzenia pliku, który nie istnieje:	17
6.4. Liczby pseudolosowe	17
6.4.1. Podanie poprawnego rozmiaru i zakresu liczb (N):	17
6.4.2. Podanie rozmiaru tablicy, który nie jest liczbą:	18
6.5. Generowanie ciągów liczb losowych do pliku	18
6.5.1. Podanie poprawnego rozmiaru i zakresu liczb do losowania:	18
6.5.2. Podanie rozmiaru tablicy, nie będącego liczbą:	19
7. Wnioski	19
Źródła	19

1. Zadanie

Realizowanym zadaniem jest implementacja i omówienie algorytmów sortowania przez scalanie (**mergeSort**) oraz sortowania metodą **quicksort**.

2. Sortowanie metodą quicksort

2.1. Informacje ogólne

Algorytm sortowania „**quickSort**” opiera się na strategii „dziel i zwyciężaj”.

Strategia „dziel i zwyciężaj”:

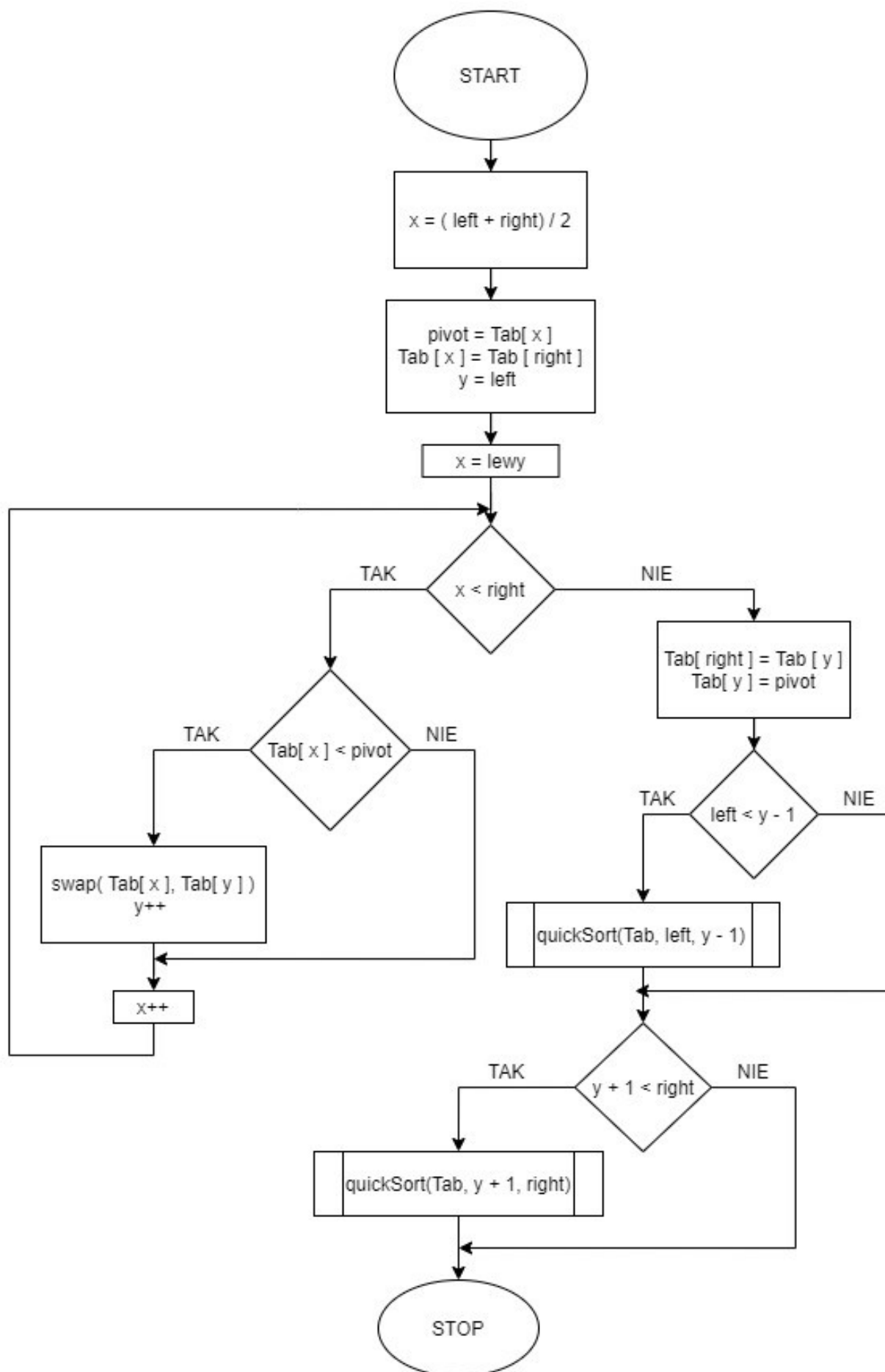
1. **Dziel** – problem główny dzielony jest na mniejsze podproblemy
2. **Zwyciężaj** – znajdowane jest rozwiązanie podproblemów
3. **Połącz** – następuje połączenie znalezionych rozwiązań podproblemów w rozwiązanie problemu głównego

Sortowanie szybkie zostało wynalezione przez angielskiego informatyka, profesora Tony’ego Hoare’a w latach 60. W trakcie sortowania **quicksort** z tablicy wybiera się element rozdzielający „**pivot**”, a następnie tablica jest dzielona na dwie części, do jednej przenoszone są elementy mniejsze od naszego **pivota**, a do drugiej elementy większe od niego. Wybrany **pivot** wymieniany jest z ostatnim elementem zbioru i w trakcie podziału jest tam bezpiecznie przechowywany. Później oba te fragmenty są sortowane, wywołując algorytm rekurencyjnie.

2.2. Złożoność obliczeniowa

W przypadku optymistycznym i typowym złożoność algorytmu quicksort jest równa $O(n \cdot \log n)$, dzieje się tak gdy za każdym razem w czasie sortowania wybierana jest mediana z sortowanego fragmentu tablicy. Równocześnie wtedy otrzymujemy minimalne zagnieżdżenie rekursji (głębokość stosu), a co za tym idzie złożoność pamięciową równą $\log_2 n$. W przypadku niekorzystnego wyboru pivotu złożoność obliczeniowa algorytmu może się zdegradować nawet do $O(n^2)$, a ilość wywołań rekurencyjnych może spowodować przepełnienie stosu i zatrzymać działanie programu. Dlatego algorytmu sortowania quicksort nie można stosować bezmyślnie dla każdego przypadku, tylko ze względu na to że uważany jest za najszybszy algorytm sortujący. Sortowanie quickSort ponadto nie jest algorytmem stabilnym, czyli elementy o tych samych wartościach nie pojawią się w tej samej kolejności w posortowanym zbiorze.

2.3. Schemat blokowy



2.4. Pseudokod

quickSort (int *Tab, int left, int right)

1. deklarujemy zmienne **x**, **y** oraz **pivot** (element podziałowy)
2. do zmiennej **x** przypisujemy wartość (**left + right**) / 2
3. do zmiennej **pivot** przypisujemy wartość zawartą w tablicy pod indeksem **x**
4. do komórki tablicy pod indeksem **x** przypisujemy wartość spod indeksu **right**
5. **for** y = x = left **to** x < right **do**
6. **if** Tab[x] < pivot **then**
7. **swap**(Tab[x] **and** Tab[y])
8. zwiększamy wartość **y** o 1
9. komórce tablicy pod indeksem **right** przypisujemy wartość spod indeksu **y**
10. komórce tablicy pod indeksem **y** przypisujemy wartość zmiennej **pivot**
11. **if** left < y - 1 **then**
12. wywołujemy rekurencyjnie funkcję: quickSort(Tab, left , y - 1)
13. **if** y + 1 < right **then**
14. wywołujemy rekurencyjnie funkcję: quickSort(Tab, y + 1 , right)
15. **koniec działania funkcji**

2.5. Implementacja C++

```
void quickSort(int Tab[], int left, int right) // funkcja realizująca sortowanie quickSort
{
    int x; // deklaracja zmiennej x
    int y; // deklaracja zmiennej y
    int pivot; // deklaracja zmiennej pivot, bedacej elementem podzialowym
    x = (left + right)/2; // przypisanie wartosci do zmiennej x
    pivot = Tab[x]; // przypisanie do zmiennej pivot wartosci zawartej w tablicy pod indeksem x
    Tab[x] = Tab[right]; // przypisanie do komorki tablicy pod indeksem x, wartosci zawartej pod indeksem right
    for(y=x;left; x < right; x++)
    {
        if(Tab[x] < pivot)
        {
            swap(Tab[x], Tab[y]); // zamiana miejscami wartosci tablicy zawartych pod indeksami x i y
            y++; // inkrementacja zmiennej y
        }
    };
    Tab[right] = Tab[y]; // przypisanie do komorki tablicy zawartej pod indeksem right, wartosci z komorki zawartej pod indeksem y
    Tab[y] = pivot; // przypisanie do komorki tablicy pod indeksem y, wartosci zawartej pod indeksem pivot
    if(left < y - 1)
    {
        quickSort(Tab, left, y - 1); // rekurencyjne wywołanie algorytmu sortujacego "quickSort" dla lewej podtablicy
    };
    if(y + 1 < right)
    {
        quickSort(Tab, y + 1, right); // rekurencyjne wywołanie algorytmu sortujacego "quickSort" dla prawej podtablicy
    };
}
```

3. Sortowanie przez scalanie

3.1. Informacje ogólne

Sortowanie przez scalanie(**mergeSort**) jest to rekurencyjny algorytm sortowania danych, podobnie jak **quickSort** opiera się na metodzie „dziel i zwyciężaj”. Jego odkrycie przypisuje się Johnowi von Neumannowi. Wyróżnić można trzy podstawowe kroki algorytmu:

1. Podzielenie zbioru elementów, które chcemy posortować na dwie równe części.
2. Zastosowanie sortowania przez scalanie dla każdej z części z osobna.
3. Połączenie posortowanych części w całość, tworząc posortowany ciąg.

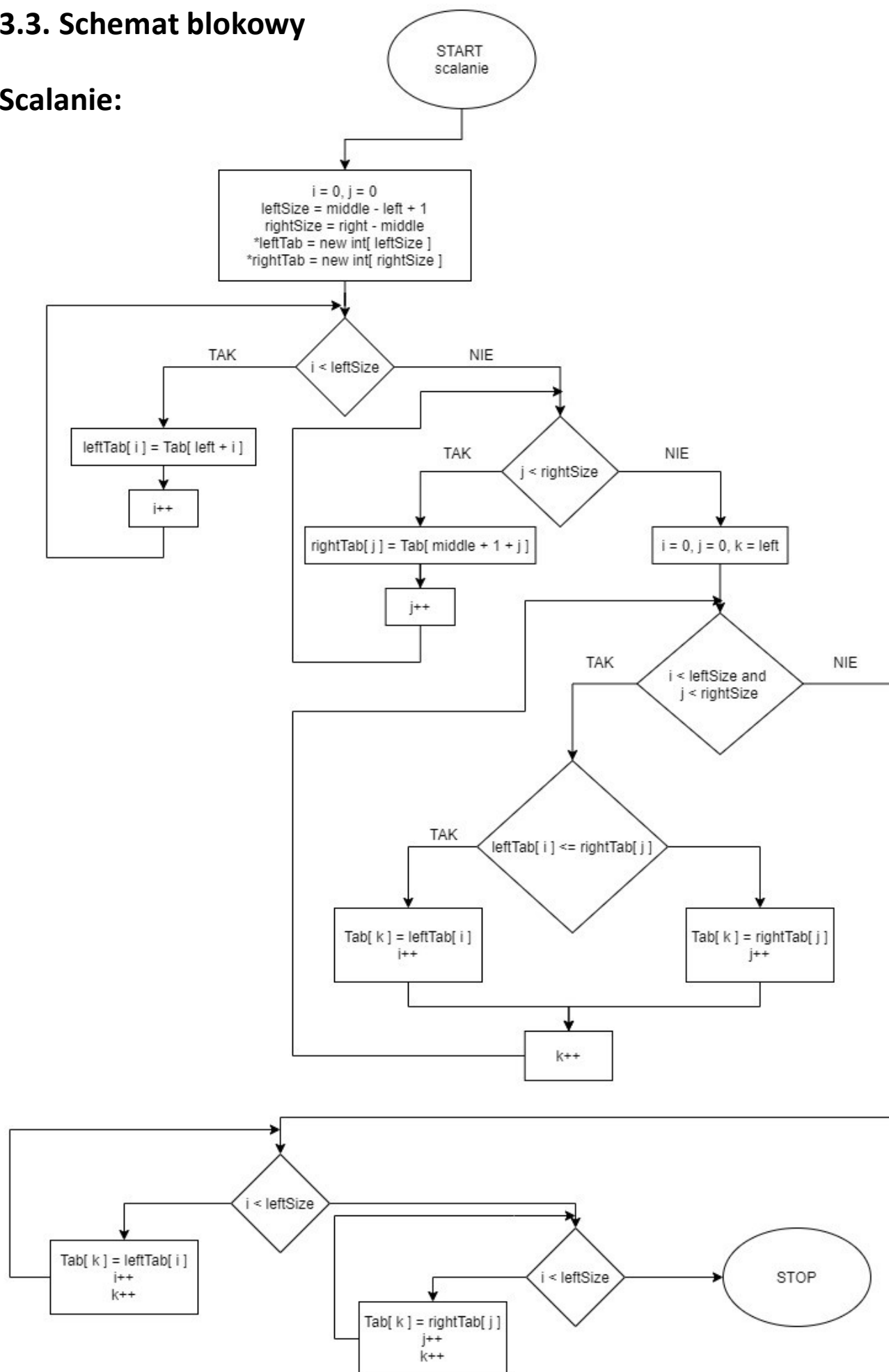
Szczególnie dobrze działa w przypadku danych dostępnych sekwencyjnie, czyli po kolei, jeden element naraz, na przykład w postaci listy jednokierunkowej.

3.2. Złożoność obliczeniowa

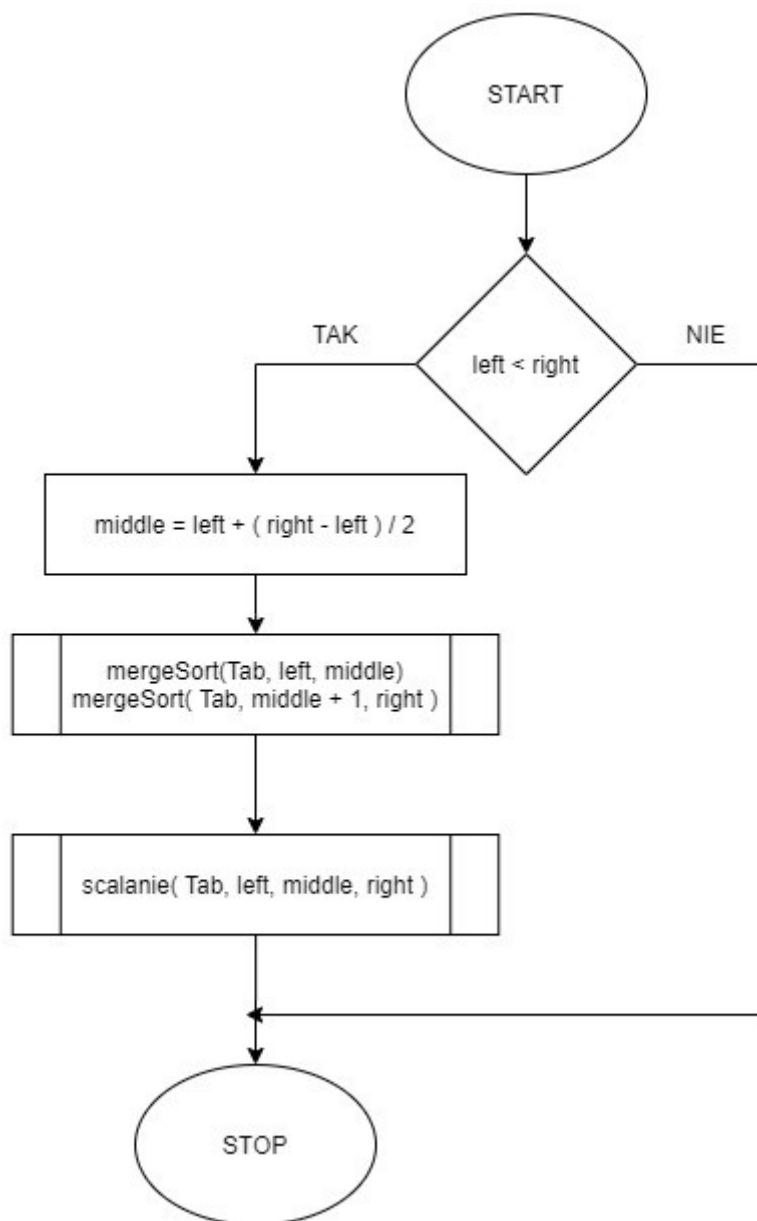
W przypadku algorytmu mergeSort złożoność czasowa nie różni się pomiędzy przypadkami(pesymistycznym, typowym i optymistycznym) i w każdym z nich jest równa **$O(n \cdot \log n)$** . Jednak szybkość działania jest okupiona większym zapotrzebowaniem na pamięć, gdyż w algorytmie korzystamy z dodatkowej tablicy potrzebnej do scalania posortowanych podtablic, której wielkość jest zależna od ilości sortowanych danych. Mówimy wtedy że sortowanie odbywa się **nie w miejscu(not in place)**. Sortowanie przez scalanie działa wydajnie na bardzo dużych zbiorach danych i jest w odróżnieniu od **quickSort** jest algorytmem stabilnym, czyli dwa elementy o tej samej wartości pojawią się w posortowanej liście w tej samej kolejności w której były jako dane wejściowe.

3.3. Schemat blokowy

Scalanie:



mergeSort:



3.4. Pseudokod

scalanie (int ***Tab**, int **left**, int **middle**, int **right**)

1. deklarujemy zmienne iteracyjne **i**, **j**, **k**
2. deklarujemy zmienną **leftSize** i przypisujemy do niej rozmiar lewej podtablicy
3. deklarujemy zmienną **rightSize** i przypisujemy do niej rozmiar prawej podtablicy
4. tworzymy tablicę dynamiczną **leftTab[]** oraz **rightTab[]** o rozmiarach **leftSize** oraz **rightSize**

5. **for** i = 0 **to** i < leftSize **do**
6. leftTab[i] = Tab [left + i]
7. **for** j = 0 **to** i < rightSize **do**
8. rightTab[j] = Tab [middle + 1 + j]
9. zmiennym **i** oraz **j** nadajemy wartość 0, a do zmiennej **k** przypisujemy wartość left
10. **while** i < leftSize **and** j < rightSize **do**
11. **if** leftTab[i] <= rightTab[j] **then**
12. Tab[k] = leftTab[i]
13. zwiększamy wartość **i** o 1
14. **else**
15. Tab[k] = rightTab[j]
16. zwiększamy wartość **j** o 1
17. zwiększamy wartość **k** o 1
18. **while** i < leftSize **do**
19. Tab[k] = leftTab[i]
20. zwiększamy wartości zmiennych **i** oraz **k** o 1
21. **while** j < rightSize **do**
22. Tab[k] = rightTab[j]
23. zwiększamy wartości zmiennych **j** oraz **k** o 1
24. **koniec działania funkcji**

mergeSort (int *Tab, int left, int right)

1. deklarujemy zmienną **middle** przechowującą środek tablicy
2. **if** left < right **then**
3. do zmiennej **middle** przypisujemy wartość $\text{left} + (\text{right} - \text{left}) / 2$
4. wywołaj mergeSort(Tab, left, middle) – sortujemy lewą stronę rekurencyjnie
5. wywołaj mergeSort(Tab, middle + 1, right) – sortujemy prawą stronę rekurencyjnie
6. wywołaj scalenie(Tab, left, middle, right) – scalamy obie części w jedną tablicę wcześniej napisaną funkcją „scalenie”
7. **koniec działania funkcji**

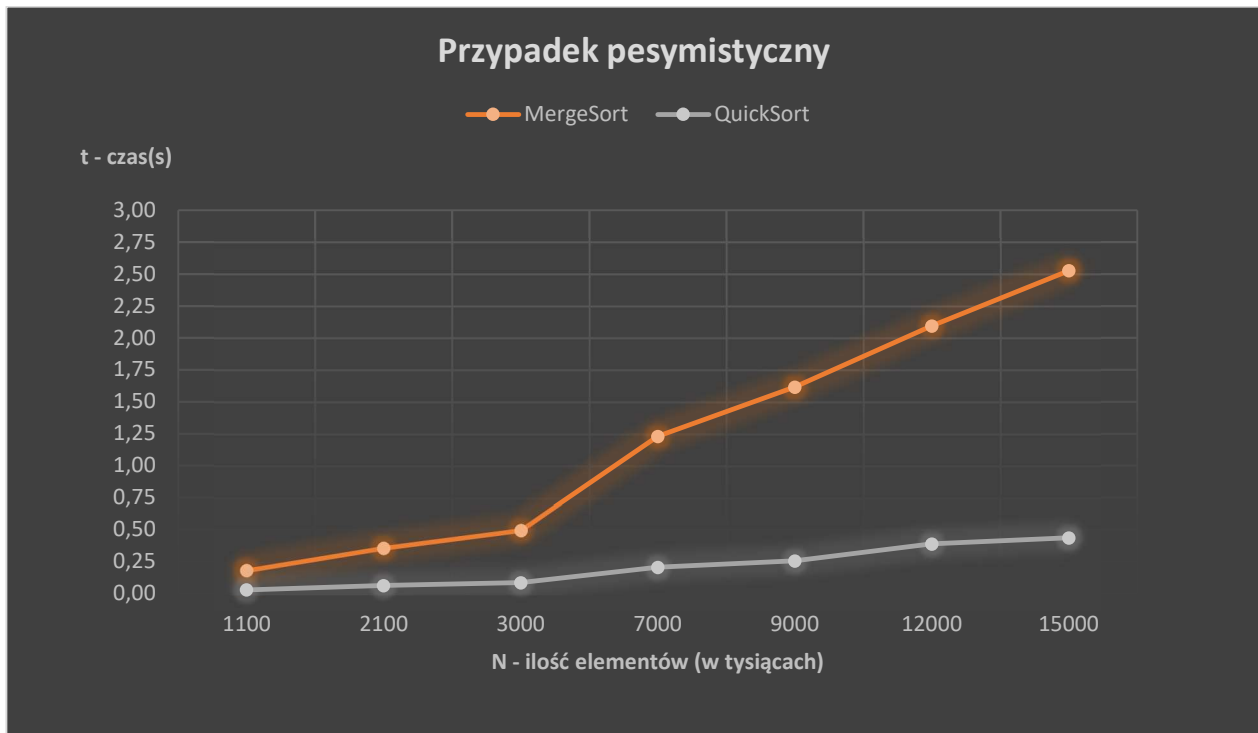
3.5. Implementacja C++

```
void scalanie(int *Tab, int left, int middle, int right) // funkcja scalajaca podtablice
{
    int i, j, k; // zmienne iteracyjne do petli "for"
    int leftSize = middle-left+1; // rozmiar lewej podtablicy
    int rightSize = right-middle; // rozmiar prawej podtablicy
    int *leftTab = new int[leftSize]; // deklaracja lewej podtablicy o odpowiednim rozmiarze
    int *rightTab = new int[rightSize]; // deklaracja prawej podtablicy o odpowiednim rozmiarze
    for(i = 0; i<leftSize; i++)
    {
        leftTab[i] = Tab[left+i]; // wypelnienie lewej podtablicy odpowiednimi liczbami z tablicy
    };
    for(j = 0; j<rightSize; j++)
    {
        rightTab[j] = Tab[middle+1+j]; // wypelnienie prawej podtablicy odpowiednimi liczbami z tablicy
    };
    i = 0; j = 0; k = left; // przypisanie wartosci zmiennym iteracyjnym potrzebnym do petli "for"
    while(i < leftSize && j<rightSize)
    {
        if(leftTab[i] <= rightTab[j])
        {
            Tab[k] = leftTab[i]; // scalenie podtablic z tablica glowna
            i++; // inkrementacja zmiennej i
        }
        else
        {
            Tab[k] = rightTab[j];
            j++; // inkrementacja zmiennej j
        }
        k++; // inkrementacja zmiennej k
    };
    while(i<leftSize)
    {
        Tab[k] = leftTab[i]; // dodatkowy element z lewej podtablicy
        i++; k++; // inkrementacja zmiennej k i zmiennej i
    }
    while(j<rightSize)
    {
        Tab[k] = rightTab[j]; //dodatkowy element z prawej podtablicy
        j++; k++; // inkrementacja zmiennej k i zmiennej j
    }
};

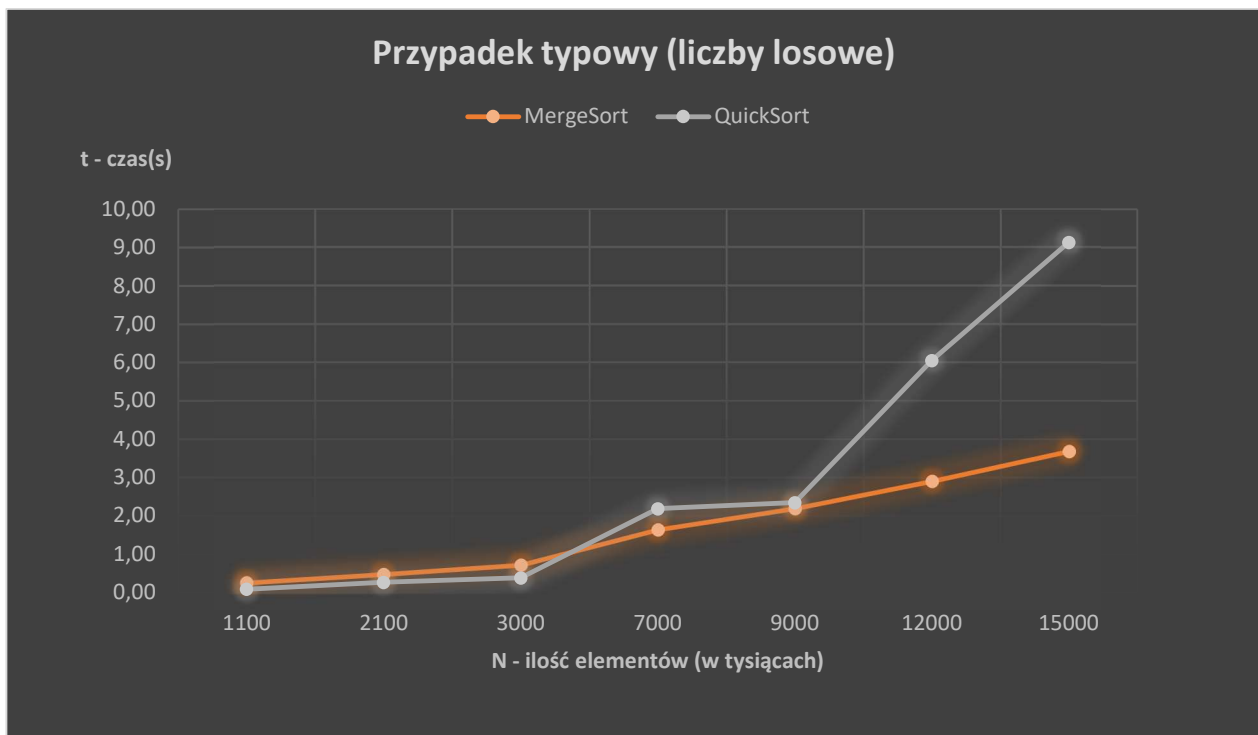
void mergeSort(int *Tab, int left, int right) // funkcja realizujaca sortowanie przez scalanie
{
    int middle; // zmienna przechowujaca srodek tablicy
    if(left < right)
    {
        middle = left+(right-left)/2; // wyznaczenie srodka tablicy, potrzebnego do podzialu na 2 tablice,
        mergeSort(Tab, left, middle); // sortowanie rekurencyjnie lewej czesci tablicy
        mergeSort(Tab, middle+1, right); // sortowanie rekurencyjnie prawej czesci tablicy
        scalanie(Tab, left, middle, right); // scalanie lewej i prawej czesci tablicy w calosc
    }
};
```

4. Efektywność algorytmów

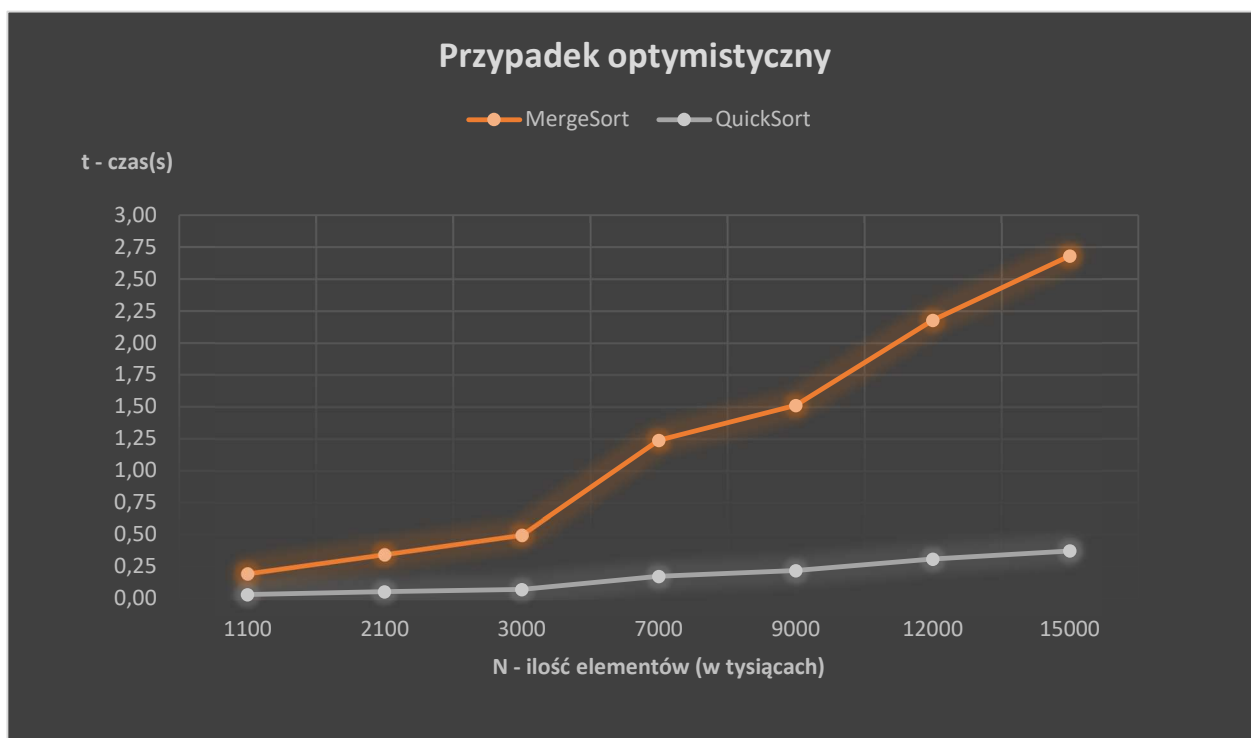
4.1. Przypadek pesymistyczny – zbiór posortowany malejąco



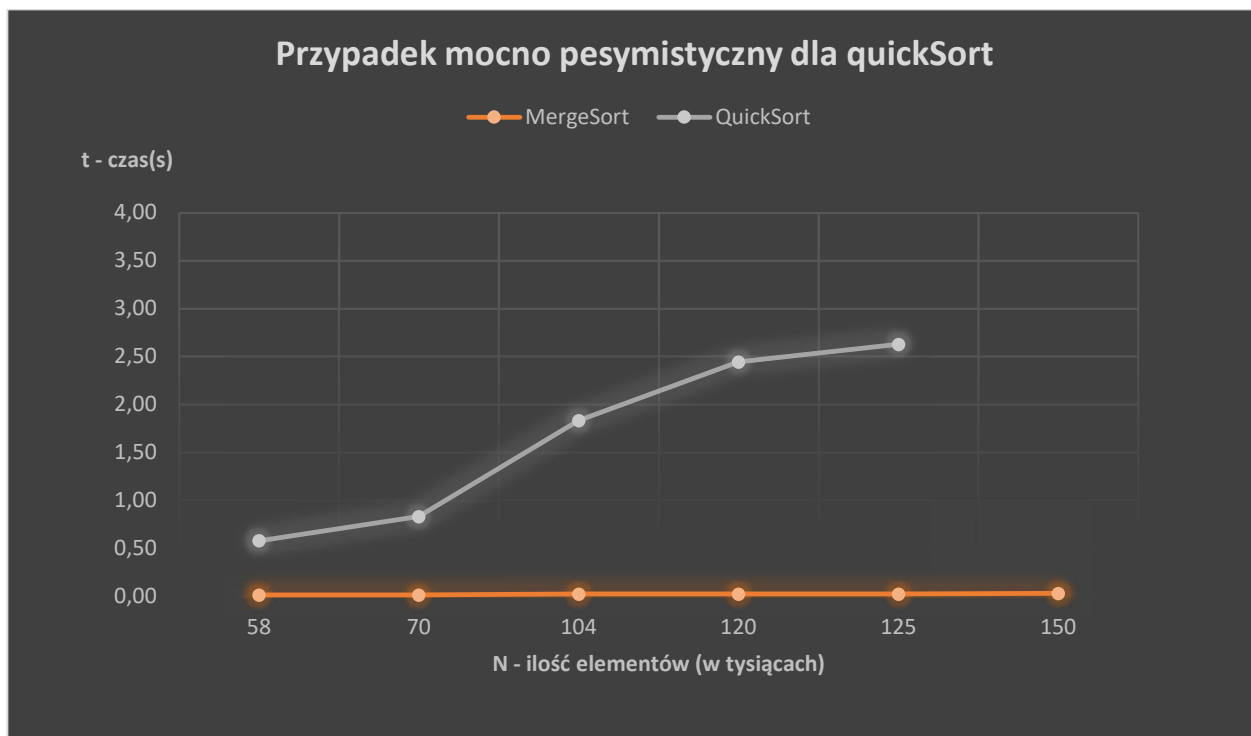
4.2. Przypadek typowy (oczekiwany) – liczby losowe



4.3. Przypadek optymistyczny – zbiór posortowany rosnąco



4.4. Przypadek mocno pesymistyczny dla algorytmu quickSort – specyficjnie ułożone liczby



Testy przeprowadzony na specyficznym ułożeniu liczb wykazał słabość algorytmu **quickSort**, który osiągnął złożoność $O(n^2)$, dla o wiele mniejszych liczb niż w poprzednich testach pojawiają się większe czasy, a przy próbie posortowania ok. 150 tysięcy elementów stos został przepełniony i program zakończył działanie. Natomiast algorytm sortowania **mergeSort** nie ma żadnych problemów z posortowaniem tych samych danych, czasy w jakie było przeprowadzone sortowanie okazały się być wręcz znikome.

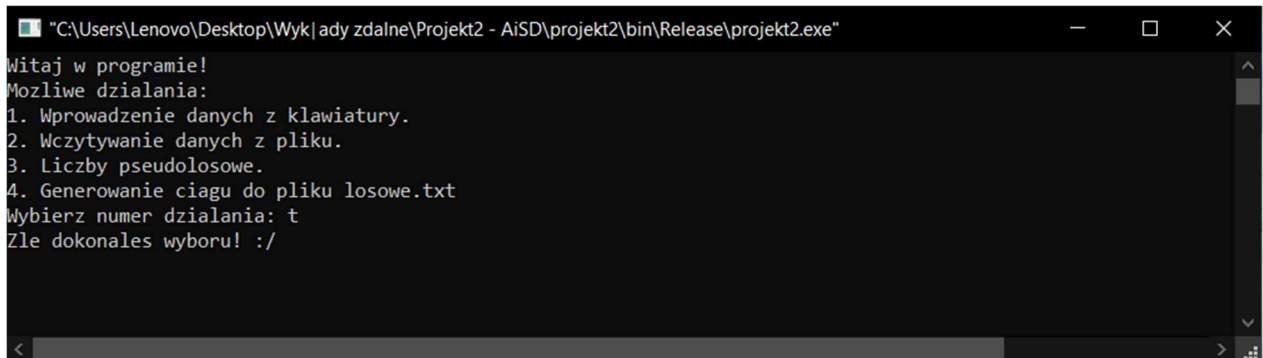
5. Specyfikacja programu

Wszystkie operacje w programie potwierdzane są klawiszem ENTER. W głównym folderze projektu znajdują się:

- folder „dane”, zawierający kilka plików z poprawnymi danymi gotowymi do wczytania oraz plik „empty.txt” nie zawierający danych (potrzebny do sprawdzenia poprawnego działania programu),
- plik „losowe.txt” do którego zapisywane są liczby wygenerowane losowo w przypadku wybrania w MENU opcji generowania ciągu do pliku, lub w przypadku wybrania opcji „Liczby pseudolosowe”,
- plik „wynik.txt” do którego zapisywany jest wynik działania programu,

6. Dokumentacja z doświadczeń

6.1. Niepoprawne dokonanie wyboru w menu początkowym:

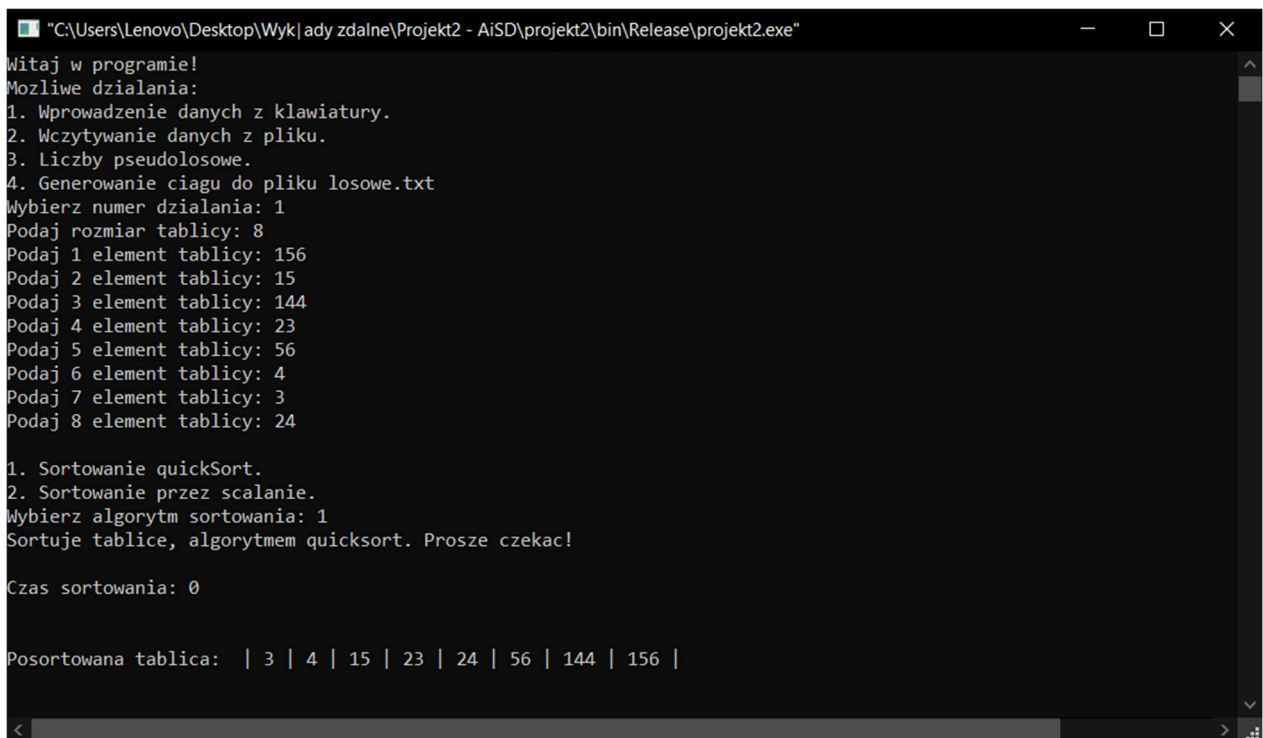


```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: t
Zle dokonales wyboru! :/
```

Program wyświetli komunikat „Złe dokonałeś wyboru! :/” i zakończy działanie.

6.2. Wprowadzanie danych z klawiatury:

6.2.1. Poprawne dane:



```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 1
Podaj rozmiar tablicy: 8
Podaj 1 element tablicy: 156
Podaj 2 element tablicy: 15
Podaj 3 element tablicy: 144
Podaj 4 element tablicy: 23
Podaj 5 element tablicy: 56
Podaj 6 element tablicy: 4
Podaj 7 element tablicy: 3
Podaj 8 element tablicy: 24

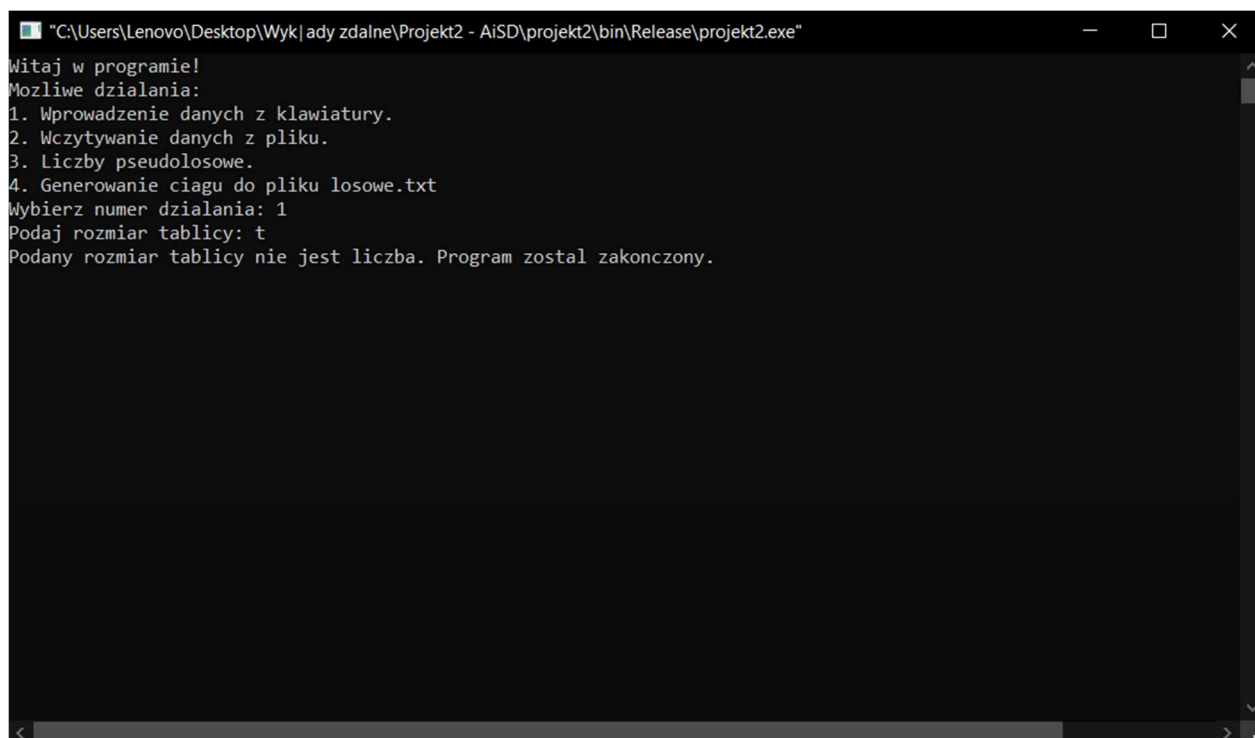
1. Sortowanie quickSort.
2. Sortowanie przez scalanie.
Wybierz algorytm sortowania: 1
Sortuje tablice, algorytmem quicksort. Prosze czekac!

Czas sortowania: 0

Posortowana tablica: | 3 | 4 | 15 | 23 | 24 | 56 | 144 | 156 |
```

Jeśli wprowadzone dane są poprawne, program prosi o wybór algorytmu sortowania, a następnie wyświetla czas sortowania, posortowaną tablicę, zapisuje posortowaną tablicę do pliku „wynik.txt” i kończy działanie.

6.2.2. Niepoprawna długość tablicy, wprowadzenie znaku innego niż liczba:



```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 1
Podaj rozmiar tablicy: t
Podany rozmiar tablicy nie jest liczba. Program został zakończony.
```

Program zakończy działanie i wyświetli komunikat „Podany rozmiar nie jest liczbą. Program został zakończony.”

6.2.3. Próba wprowadzenia innego elementu niż liczba do tablicy:



```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 1
Podaj rozmiar tablicy: 5
Podaj 1 element tablicy: 2
Podaj 2 element tablicy: 1
Podaj 3 element tablicy: g
Wprowadz liczbe!
Podaj 3 element tablicy: h
Wprowadz liczbe!
Podaj 3 element tablicy: 2
Podaj 4 element tablicy: 5
Podaj 5 element tablicy: j
Wprowadz liczbe!
Podaj 5 element tablicy: 5

1. Sortowanie quickSort.
2. Sortowanie przez scalanie.
Wybierz algorytm sortowania: 1
Sortuje tablice, algorytmem quicksort. Prosze czekac!

Czas sortowania: 0

Posortowana tablica: | 1 | 2 | 2 | 5 | 5 |
```

Program wyświetli komunikat „Wprowadź liczbę!” i poprosi o ponowne podanie elementu tablicy.

6.3.Wczytywanie danych z pliku

6.3.1. Poprawne dane w pliku:

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 2
Podaj sciezke pliku: dane/liczby.txt
Ilosc: 15

1. Sortowanie quickSort.
2. Sortowanie przez scalanie.
Wybierz algorytm sortowania: 1
Sortuje tablice, algorytmem quicksort. Prosze czekac!

Czas sortowania: 0

Posortowana tablica: | 6 | 18 | 21 | 26 | 65 | 65 | 69 | 70 | 73 | 92 | 97 | 99 | 121 | 127 | 155 |

Process returned 0 (0x0)   execution time : 11.219 s
Press any key to continue.
```

Program odczytuje dane z pliku tekstowego, a następnie prosi o wybór algorytmu sortowania. Po wybraniu algorytmu którym chcemy sortować, program sortuje ciąg, wyświetla czas sortowania i zapisuje wynik sortowania do pliku tekstowego „wynik.txt”. Opcjonalnie program wyświetla posortowaną tablicę na ekranie (ta opcja jest domyślnie wyłączona ze względu na duże liczby wczytywane z pliku w czasie testów).

6.3.2. Plik nie zawiera danych (plik pusty):

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 2
Podaj sciezke pliku: dane/empty.txt
Plik jest pusty.

Process returned 0 (0x0)   execution time : 17.890 s
Press any key to continue.
```

Program wyświetla komunikat „Plik jest pusty.” i kończy działanie.

6.3.3. Próba otworzenia pliku, który nie istnieje:

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 2
Podaj sciezke pliku: liczby.txt
Nie mozna otworzyc pliku

Process returned 0 (0x0)   execution time : 5.442 s
Press any key to continue.
```

Program wyświetla komunikat „Nie można otworzyć pliku” i kończy działanie.

6.4. Liczby pseudolosowe

6.4.1. Podanie poprawnego rozmiaru i zakresu liczb (N):

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 3
Podaj rozmiar tablicy: 15
Zakres losowania [ 0, N ], podaj N: 252

| 95 | 50 | 152 | 59 | 194 | 125 | 98 | 120 | 182 | 206 | 40 | 17 | 197 | 222 | 29 |

1. Sortowanie quickSort.
2. Sortowanie przez scalanie.
Wybierz algorytm sortowania: 2
Sortuje tablice, algorytmem przez scalanie. Prosze czekac!

Czas sortowania: 0

Posortowana tablica: | 17 | 29 | 40 | 50 | 59 | 95 | 98 | 120 | 125 | 152 | 182 | 194 | 197 | 206 | 222 |
```

Program przyjmie rozmiar tablicy i zakres losowania, następnie wylosuje liczby z podanego przedziału i poprosi o wybór algorytmu sortowania, po wybraniu algorytmu którym chcemy posortować nasz ciąg program wyświetla: czas sortowania, posortowaną tablicę oraz zapisuje posortowaną tablicę do pliku „wynik.txt”.

6.4.2. Podanie rozmiaru tablicy, który nie jest liczbą:

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 3
Podaj rozmiar tablicy: h
Podany rozmiar tablicy nie jest liczba. Program został zakończony.

Process returned 0 (0x0)   execution time : 4.688 s
Press any key to continue.
```

Program zakończy działanie i wyświetli komunikat „Podany rozmiar nie jest liczbą. Program został zakończony.”

6.5. Generowanie ciągów liczb losowych do pliku

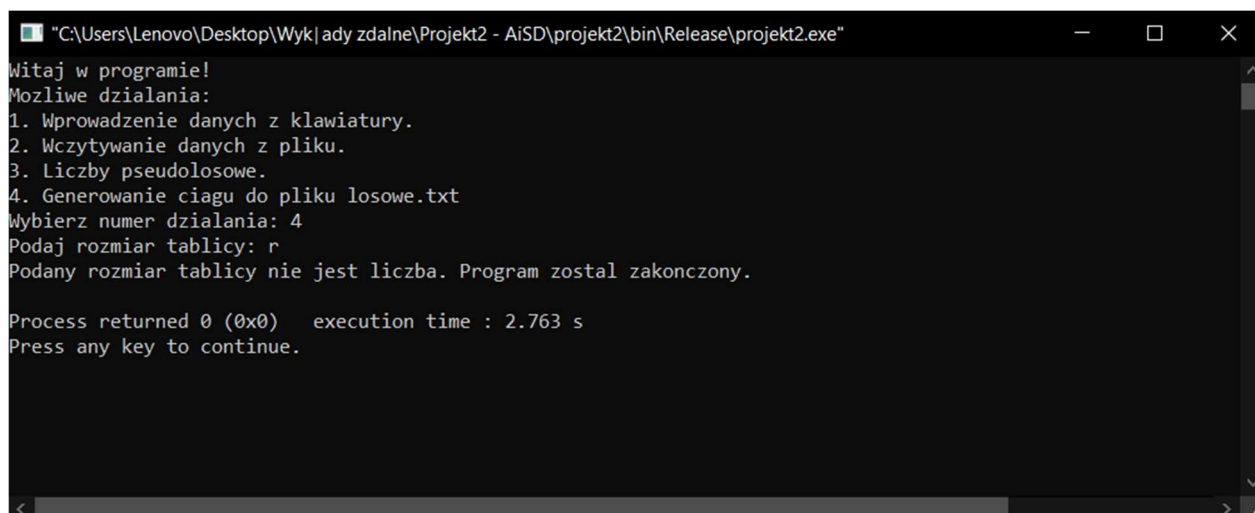
6.5.1. Podanie poprawnego rozmiaru i zakresu liczb do losowania:

```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 4
Podaj rozmiar tablicy: 250000
Zakres losowania [ 0, N ], podaj N: 26745

Gotowe!
```

Program przyjmie rozmiar tablicy oraz zakres losowania, a następnie rozpocznie losowanie. Po wylosowaniu i wypisaniu liczb do pliku „losowe.txt” program zakończy działanie i wyświetli komunikat „Gotowe!”.

6.5.2. Podanie rozmiaru tablicy, nie będącego liczbą:



```
"C:\Users\Lenovo\Desktop\Wyk\ady zdalne\Projekt2 - AiSD\projekt2\bin\Release\projekt2.exe"
Witaj w programie!
Mozliwe dzialania:
1. Wprowadzenie danych z klawiatury.
2. Wczytywanie danych z pliku.
3. Liczby pseudolosowe.
4. Generowanie ciagu do pliku losowe.txt
Wybierz numer dzialania: 4
Podaj rozmiar tablicy: r
Podany rozmiar tablicy nie jest liczba. Program zostal zakonczony.

Process returned 0 (0x0)   execution time : 2.763 s
Press any key to continue.
```

Program zakończy działanie i wyświetli komunikat „Podany rozmiar nie jest liczbą. Program został zakończony.”

7. Wnioski

Po wielu próbach i modyfikacjach udało się doprowadzić program do stanu, w którym działa on w sposób zadowalający, wszystkie testy zostały przeprowadzone pomyślnie. Program ma możliwość odczytu/zapisu danych z/do plików tekstowych, a także wprowadzania danych z klawiatury. Zaimplementowana została także funkcja generująca i zapisująca do pliku „losowe.txt” ciągi liczb pseudolosowych z zakresu $[0, N]$, (element N przedziału jest podawany przez użytkownika w czasie działania programu). Wykonane zostało także porównanie efektywności pracy obu rozpatrywanych algorytmów na różnych liczbach. Dla lepszej przejrzystości porównanie to zostało przedstawione na wykresach.

Źródłaⁱ

ⁱ <https://eduinf.waw.pl/inf/alg/>
<https://pl.wikipedia.org/wiki/>
<https://www.softwaretestinghelp.com/>
<https://www.tutorialspoint.com/>