**SPRAWOZDANIE**

**PROJEKT Z PRZEDMIOTU**

**„ALGORYTMY I STRUKTURY DANYCH”**

**TEMAT**

„Zaimplementuj sortowanie przez scalanie oraz sortowanie kopcowe.”

Bartłomiej Trzepacz, PRz, semestr 2020/2021, Inżynieria i Analiza Danych

Spis treści

[**1.** **Podstawy teoretyczne algorytmów sortowania** 3](#_Toc61287205)

[1.1 Sortowanie przez kopcowanie 3](#_Toc61287206)

[1.1.1 Opis algorytmu 3](#_Toc61287207)

[1.1.2 Tworzenie kopca 3](#_Toc61287208)

[1.1.3 Sortowanie 4](#_Toc61287209)

[1.2 Sortowanie przez scalanie 5](#_Toc61287210)

[1.2.1 Opis algorytmu 5](#_Toc61287211)

[**2.** **Schematy blokowe** 7](#_Toc61287212)

[**3.** **Pseudokod** 10](#_Toc61287213)

[3.1 Pseudokod algorytmu sortowania przez kopcowanie 10](#_Toc61287214)

[3.1.1 Budowa kopca 10](#_Toc61287215)

[3.1.2 Rozbiór kopca 10](#_Toc61287216)

[3.2 Pseudokod algorytmu sortowania przez scalanie 11](#_Toc61287217)

[3.2.1 Algorytm scalający 11](#_Toc61287218)

[3.2.2 Algorytm sortujący 11](#_Toc61287219)

[**4.** **Testy porównujące działanie obu metod na różnych próbkach danych** 12](#_Toc61287220)

[4.1 Liczby pseudolosowe (oczekiwany) 12](#_Toc61287221)

[4.2 Przypadek optymistyczny dla obu sortowań 14](#_Toc61287222)

[4.3 Przypadek pesymistyczny dla obu sortowań 15](#_Toc61287223)

[**5.** **Wnioski** 16](#_Toc61287224)

# **Podstawy teoretyczne algorytmów sortowania**

## Sortowanie przez kopcowanie

Sortowanie przez kopcowanie– jeden z algorytmów sortowania, choć niestabilny, to jednak szybki i niepochłaniający wiele pamięci (złożoność czasowa wynosi O(nlogn) a pamięciowa O(n). Jest on w praktyce z reguły nieco wolniejszy od sortowania szybkiego, lecz ma lepszą pesymistyczną złożoność czasową (przez co jest odporny np. na atak za pomocą celowo spreparowanych danych, które spowodowałyby jego znacznie wolniejsze działanie.

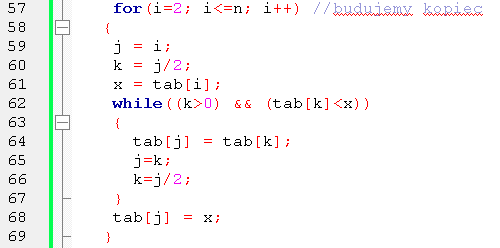
### 1.1.1 Opis algorytmu

Podstawą algorytmu jest użycie kolejki priorytetowej zaimplementowanej w postaci binarnego kopca zupełnego. Zasadniczą zaletą kopców jest stały czas dostępu do elementu maksymalnego (lub minimalnego) oraz logarytmiczny czas wstawiania i usuwania elementów; ponadto łatwo można go implementować w postaci tablicy.

Algorytm sortowania przez kopcowanie składa się z dwóch faz. W pierwszej sortowane elementy reorganizowane są w celu utworzenia kopca. W drugiej zaś dokonywane jest właściwe sortowanie.

### 1.1.2 Tworzenie kopca

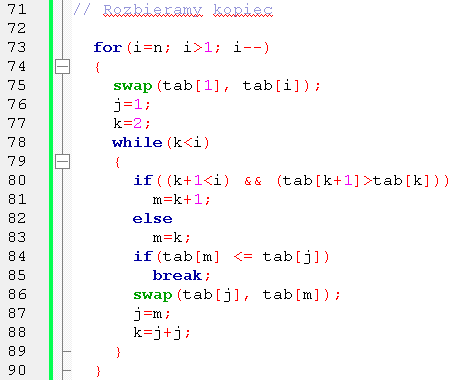
Podstawową zaletą algorytmu jest to, że do stworzenia kopca wykorzystać można tę samą tablicę, w której początkowo znajdują się nieposortowane elementy. Dzięki temu uzyskuje się stałą złożoność pamięciową. Początkowo do kopca należy tylko pierwszy element w tablicy. Następnie kopiec rozszerzany jest o drugą, trzecią i kolejne pozycje tablicy, przy czym przy każdym rozszerzeniu, nowy element jest przemieszczany w górę kopca, tak aby spełnione były relacje pomiędzy węzłami.



Rys. 1 Algorytm budowy kopca

### 1.1.3 Sortowanie

Po utworzeniu kopca następuje właściwe sortowanie. Polega ono na usunięciu wierzchołka kopca, zawierającego element maksymalny (minimalny), a następnie wstawieniu w jego miejsce elementu z końca kopca i odtworzenie porządku kopcowego. W zwolnione w ten sposób miejsce, zaraz za końcem zmniejszonego kopca wstawia się usunięty element maksymalny. Operacje te powtarza się aż do wyczerpania elementów w kopcu.



Rys. 2 Algorytm rozbioru kopca

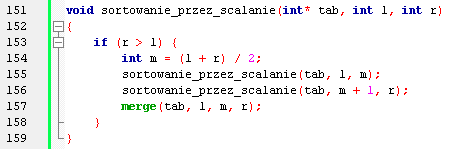
## Sortowanie przez scalanie

Algorytm sortowania przez scalanie opiera się na zasadzie „dziel i zwyciężaj”. Główna zasada działania polega na rekurencyjnym dzieleniu tablicy na podtablice. Dzielenie kończymy, w którym, każda z podtablic w danej grupie jest tablicą jednoelementową. Łączymy je kolejno porównując wartości ich elementów.

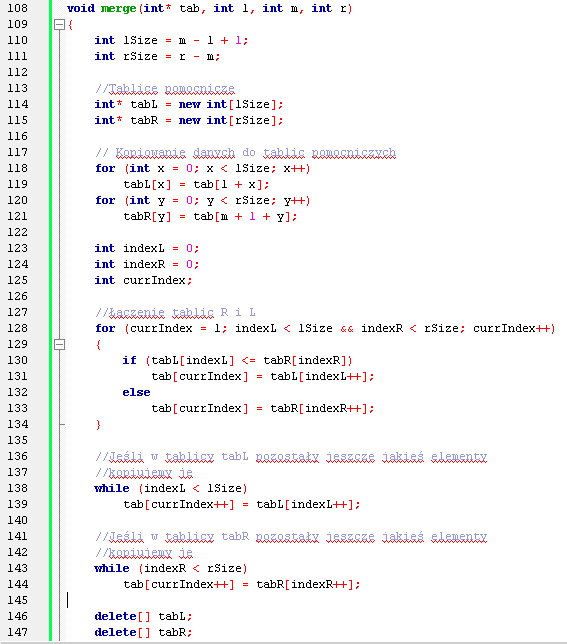
### Opis algorytmu

Jeśli indeks prawej części tablicy (r) jest większy od indeksu lewej części tablicy wykonaj następujące kroki:

1. Znajdź środkowy indeks tablicy (m), ze wzoru m=(l+r)/2 (resztę z dzielenia zaokrąglamy w dół)
2. Wywołaj sortowanie przez scalanie dla podtablicy o indeksach od l do m
3. Wywołaj sortowanie przez scalanie dla podtablicy o indeksach od m+1 do r
4. Połącz podtablicę l – m i m+1 – r, przyrównując przy tym ich elementy – Jeśli element lewej tablicy na aktualnej pozycji jest mniejszy od tego na prawej, wstaw go do tablicy docelowej i zwiększ indeks lewej tablicy. W przeciwnym przypadku, wykonaj analogiczne działanie dla prawej tablicy.

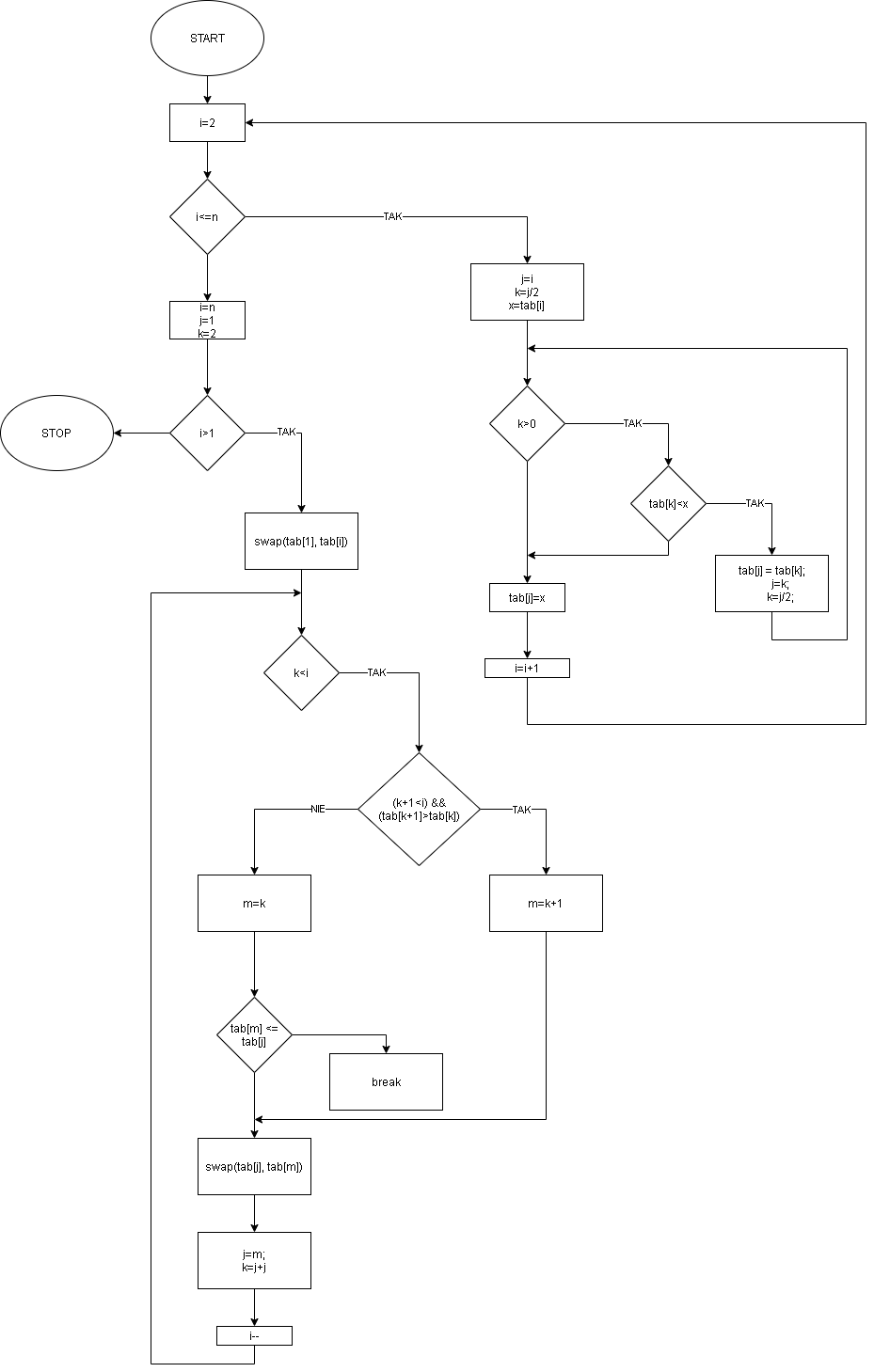


Rys. 3 Algorytm sortowania\_przez\_scalanie

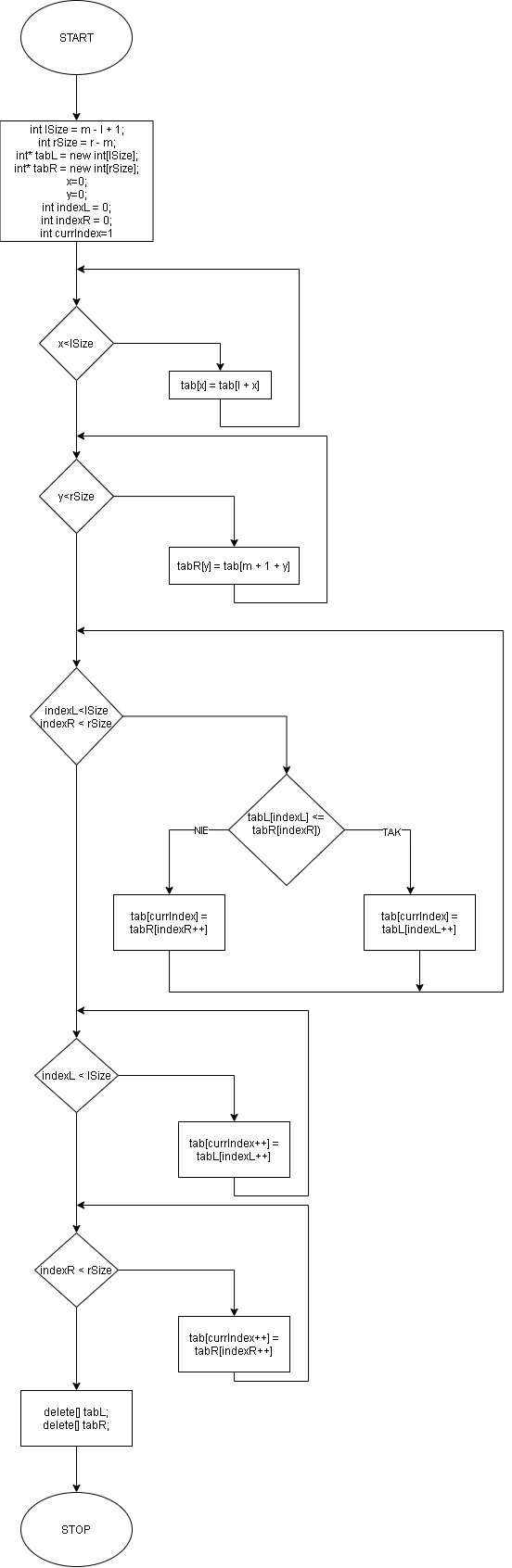


Rys. 4 Algorytm scalania

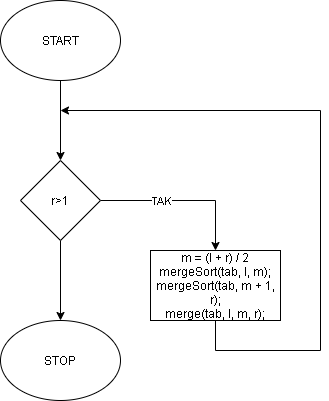
# **Schematy blokowe**



Schemat blokowy 1 Sortowanie przez kopcowanie



Schemat blokowy 2 Merge



Schemat blokowy 3 Sortowanie\_przez\_scalanie

# **Pseudokod**

## Pseudokod algorytmu sortowania przez kopcowanie

### Budowa kopca

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

wykonuj kroki K02...K05

K02: j ← i; k ← j div 2

K03: x ← d[i]

K04: Dopóki (k > 0) ∧ (d[k] < x):

wykonuj: d[j] ← d[k]

j ← k

k ← j div 2

K05: d[j] ← x

K06: Zakończ

### Rozbiór kopca

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

wykonuj kroki K02...K08

K02: d [1] ↔ d[i]

K03: j ← 1; k ← 2

K04: Dopóki (k < i):

wykonuj kroki K05...K08

K05: Jeżeli (k + 1 < i) ∧ (d k + 1] > d[k]),

to m ← k + 1

inaczej m ← k

K06: Jeżeli d[m] ≤ d[j],

to wyjdź z pętli K04

i kontynuuj następny obieg K01

K07: d[j] ↔ d[m]

K08: j ← m; k ← j + j

K09: Zakończ

## Pseudokod algorytmu sortowania przez scalanie

### Algorytm scalający

K01: lSize ← m - l + 1;

rSize ← r - m;

K02: Dla x<lSize: wykonuj K03:

K03: tabL[x] = tab [l + x]

K04: Dla y<rSize: wykonuj K05:

K05: tabR[y] = tab[m + 1 + y]

K06: int indexL ← 0; int indexR ← 0; int currIndex ← l

K07: Dla indexL < lSize i indexR < rSize wykonuj od K08 do K10:

K08: Jeśli tabL[indexL] <= tabR[indexR] wykonuj K09, w przeciwnym wypadku przejdź do K10

K09: tab[currIndex] ← tabL[indexL++]

K10: tab[currIndex] ← tabR[indexR++]

K11: Dopóki indexR < rSize wykonuj K12:

K12: tab[currIndex++] = tabL[indexL++]

K13: Dopóki indexR < rSize wykonuj K14:

K14: tab[currIndex++] = tabR[indexR++]

K15: Usuń tablice dynamiczne i zakończ

### Algorytm sortujący

K01: m ← (l + r) /2

K02: Jeśli r>1, to Sortowanie\_przez\_scalanie (tab, l,m)

K03: Jeśli r>1, to Sortowanie\_przez\_scalanie (tab, m+1, r)

K04: Scalaj (tab, l, m, r)

K05: Zakończ

# **Testy porównujące działanie obu metod na różnych próbkach danych**

## Liczby pseudolosowe (oczekiwany)

W pierwszym przypadku algorytm sortuje tablice wypełnioną liczbami pseudolosowymi. Wykorzystane zostały tablice dynamiczne co pozawala na sprawdzanie algorytmów na tablicach małych jak i dużych. Program będzie od nas wymagał podania wielkości tablicy.

Dzięki tablicy wypełnionej liczbami pseudolosowymi otrzymamy oczekiwanązłożoność czasową działania algorytmów.

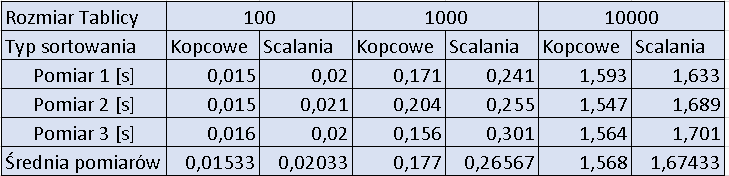
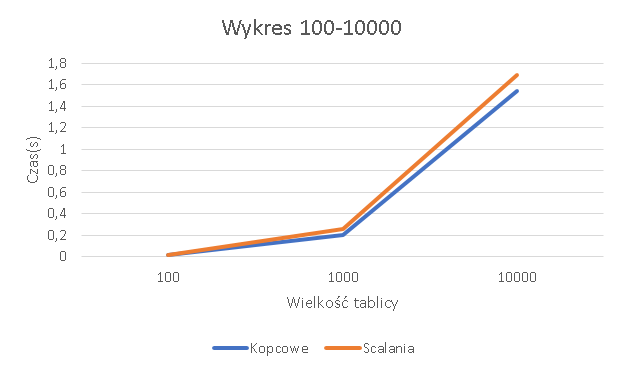


Tabela 1 Pomiary czasów dla tablic 100-100000



Wykres 1 dla tablic 100-10000

Jak przedstawia wykres powyżej od początku można zauważyć, że czasy obu algorytmów sortowania są podobne. Wynika to z faktu, że ich złożoność wynosi

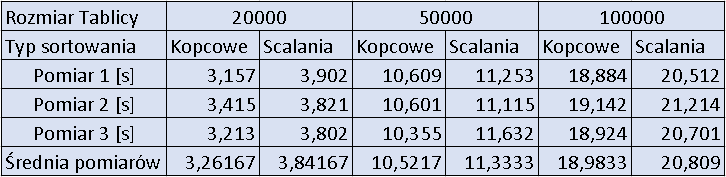
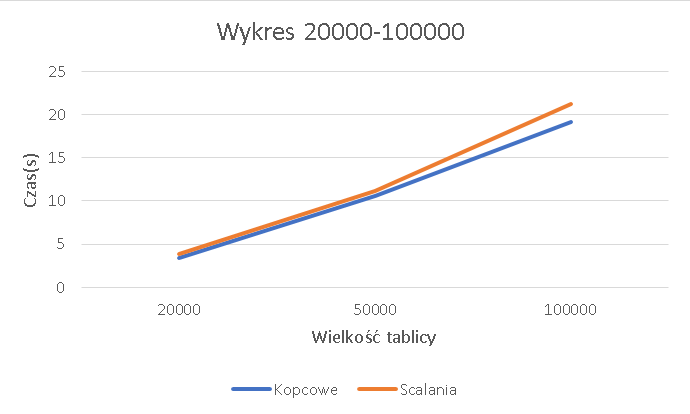


Tabela 2 Pomiary czasów dla tablic 20000-100000



Wykres 2 dla tablic 20000-100000

Dla większych tablic czas sortowania przebiega podobnie. Sortowanie przez kopcowanie z tablicą 100000- elementową poradziło sobie w czasie równym niespełna 19s, a sortowanie przez scalanie potrzebowało dla tej tablicy 2 sekundy więcej.

## Przypadek optymistyczny dla obu sortowań

Optymistyczną złożoność czasową otrzymamy wtedy, gdy tablica, którą trzeba posortować, jest już posortowana. Do tego celu użyjemy możliwości pobrania danych do posortowania z pliku tekstowego.

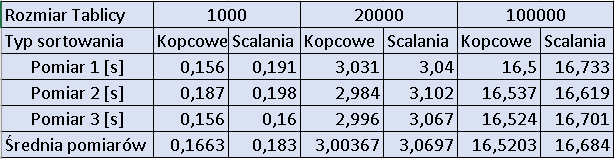
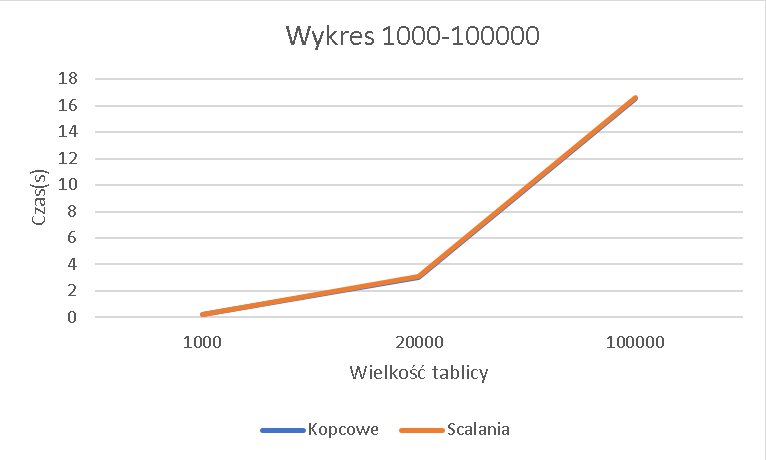


Tabela 3 Pomiary czasów dla tablic 1000-100000



Wykres 3 dla tablic 1000-100000

W tym przypadku, gdy tablice są już posortowane czasy pracy algorytmów uległ zmniejszeniu co widać w Tabela 3.

## Przypadek pesymistyczny dla obu sortowań

Pesymistyczną złożoność czasową otrzymamy wtedy, gdy tablica, którą trzeba posortować, jest posortowana odwrotnie. Do tego celu użyjemy możliwości pobrania danych do posortowania z pliku tekstowego.

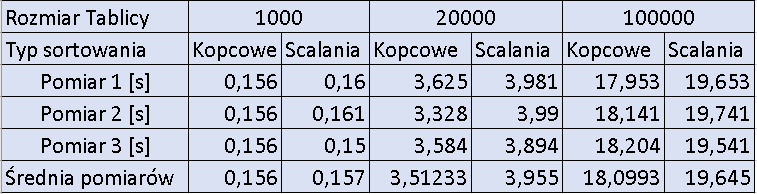
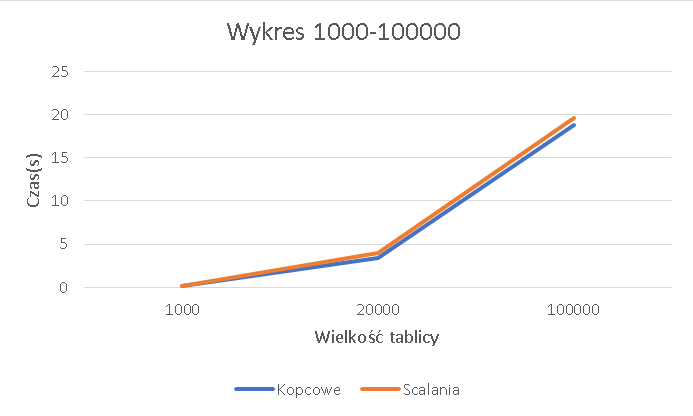


Tabela 4 Pomiary czasów dla tablic 1000-100000



Wykres 4 dla tablic 1000-100000

Jak widać w Tabela 4 czasy pomiarów zwiększyły się względem poprzednich badań. Różnice czasów sięgają nawet 10% przy większych tablicach.

# **Wnioski**

Projekt został zrealizowany, działanie algorytmu jest poprawne. W programie główny algorytm został zaimplementowany w osobnej funkcji, która jest wywoływana w późniejszych etapach działania programu. Projekt posiada możliwość odczytu z pliku tekstowego jak i późniejsze zapisanie wyników do pliku tekstowego. W programie zostały umieszczone stosowane komentarze, które pomagają w zrozumieniu kodu również w sprawozdaniu zostały umieszczone elementy które ułatwiają nam zrozumienie działania algorytmu szukającego jest to pseudokod i schemat blokowy.