Babice 14.01.2021

**Projekt**

**Algorytmów i struktur danych**

Temat:

Implementacja sortowanie przez scalanie oraz sortowanie kopcowego

Autor: Adam Gardyas

Inżynieria i analiza danych sem.1 gr.2

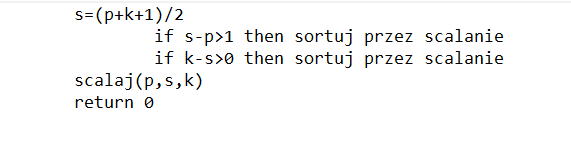
Prowadzący: dr inż. Prof. Mariusz Borkowski

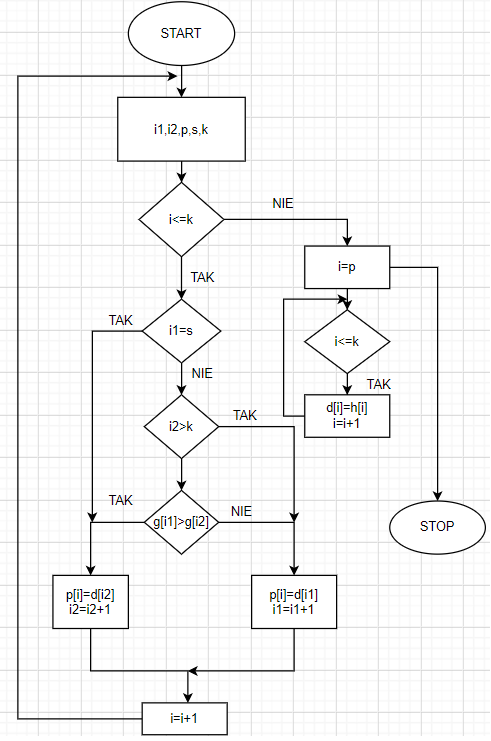
,,Zaimplementuj sortowanie przez scalanie oraz sortowanie kopcowe”

Zadanie to polegało na implementacji dwóch typów sortowania: przez scalanie oraz kopcowego. Należało przedstawić kod obydwu sortowań a także zbadać w jaki sposób różne przypadki rozmieszczenia liczb wpływają na czas tego sortowania.

Sortowanie przez scalanie posiada złożoność O(nlog n), głównym założeniem tego sposobu jest podzielenie zbioru na dwie równe części. Później należy zastosować algorytm sortowania przez scalanie dla obydwu zbiorów, a na końcu połączyć je w jeden posortowany zbiór liczb.

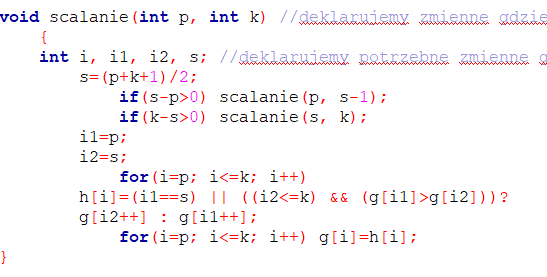
Tutaj mamy pseudokod naszego algorytmu:

Schemat blokowy sortowania przez scalanie:



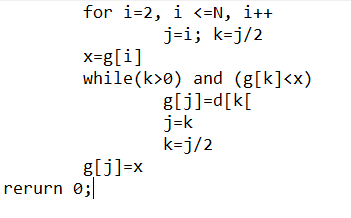
Cechu algorytmu sortowania przez scalanie:

Na początku przygotowujemy zbiór tymczasowy, dopóki żaden ze scalanych zbiorów nie jest pusty, porównuj ze sobą pierwsze elementy każdego z nich i w zbiorze tymczasowym umieszczaj mniejszy z elementów usuwając go jednocześnie ze scalanego zbioru. W zbiorze tymczasowym umieść zawartość tego scalanego zbioru, który zawiera jeszcze elementy. Zawartość zbioru tymczasowego przepisz do zbioru wynikowego i zakończ algorytm. Poniżej kod tego algorytmu.

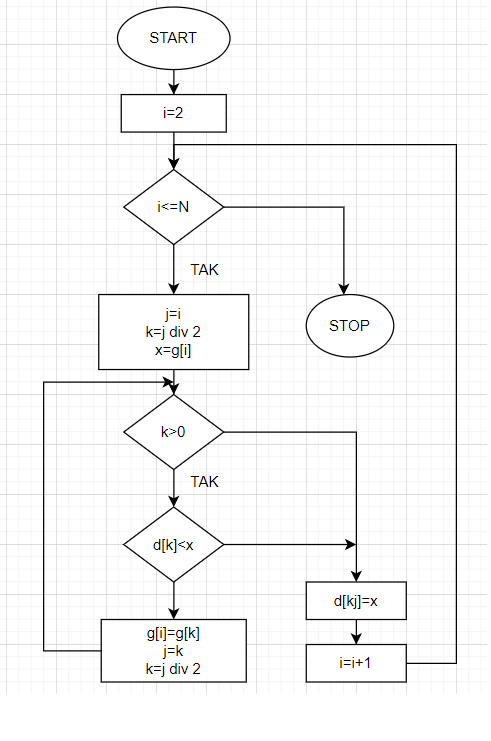


Sortowanie przez kopcowanie posiada złożoność taką samą jak sortowanie przez scalanie, czyli O(nlog n). Charakteryzuje się on tym, iż kopiec jest drzewem binarnym, spełniającym warunek kopca czyli: węzeł nadrzędny jest większy lub równy kopcom potomnym. Po każdym dołączeniu kopca do nowego węzła musimy sprawdzać odpowiednie warunki. Wyróżniającą cechą kopca jest to, że korzeń jest zawsze największym elementem tego drzewa.

Pseudokod sortowania przez kopcowanie:

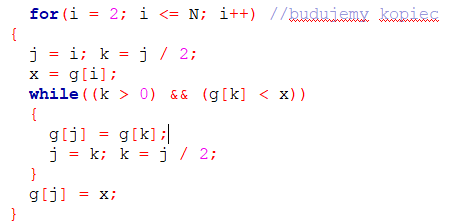


Schemat blokowy sortowania przez kopcowanie:

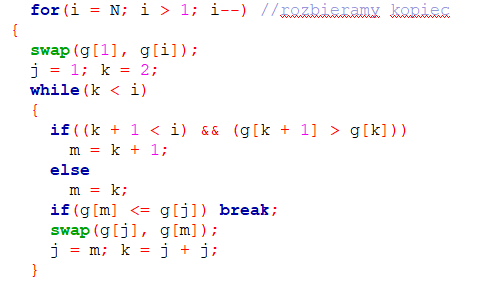


Kolejność działania algorytmu:

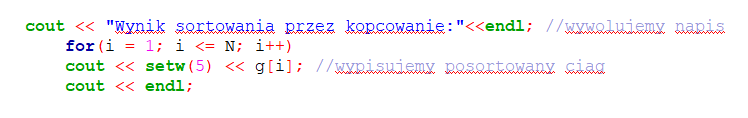
1. Na początku budujemy kopiec.



1. Rozbieramy kopiec.

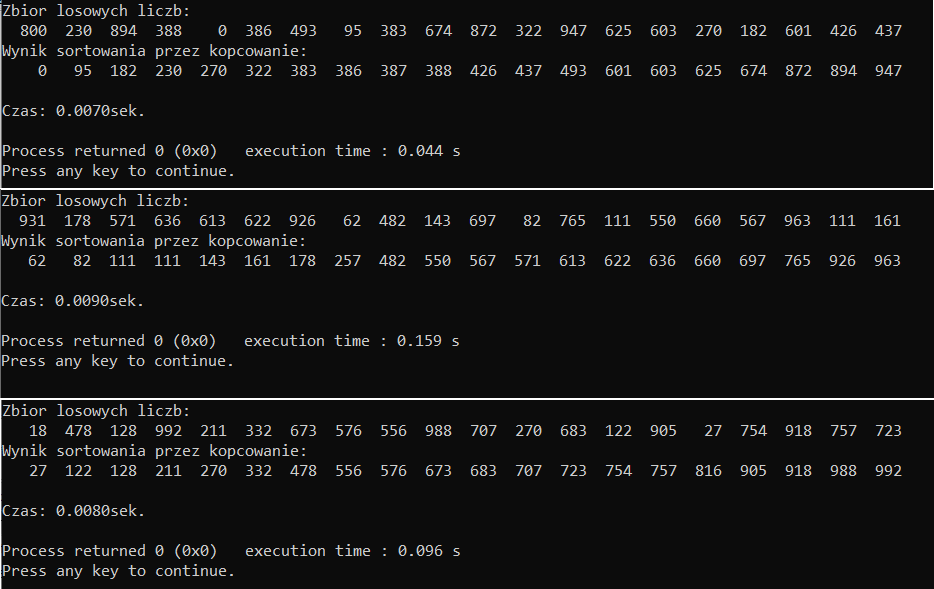


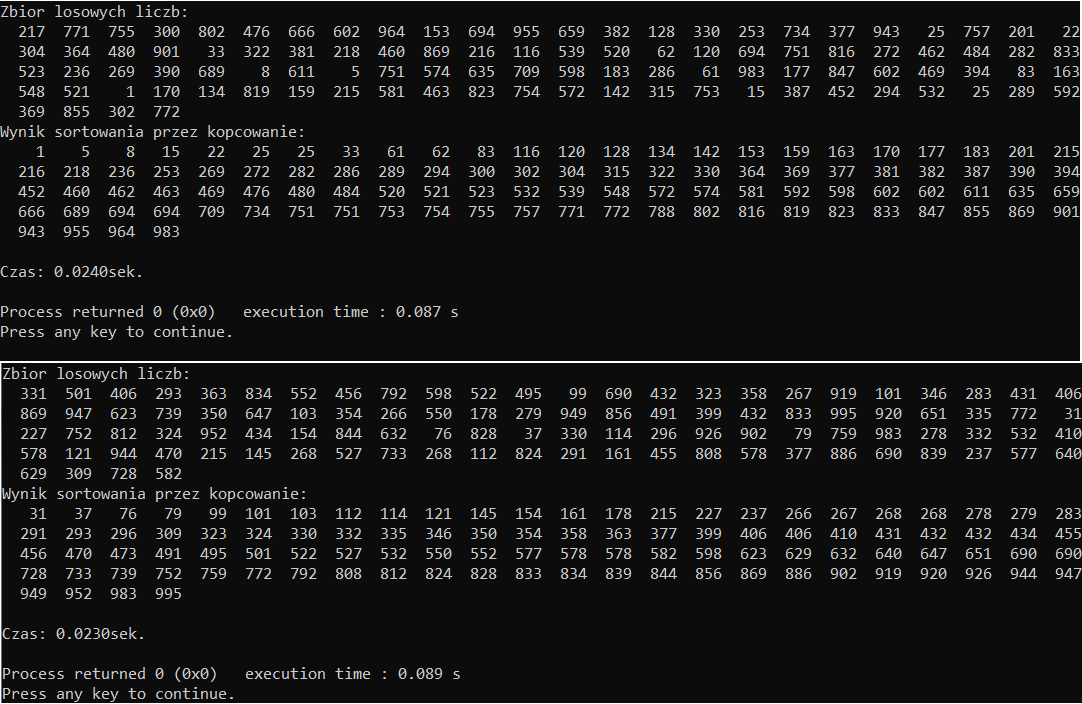
1. Wypisujemy nasz wynik.



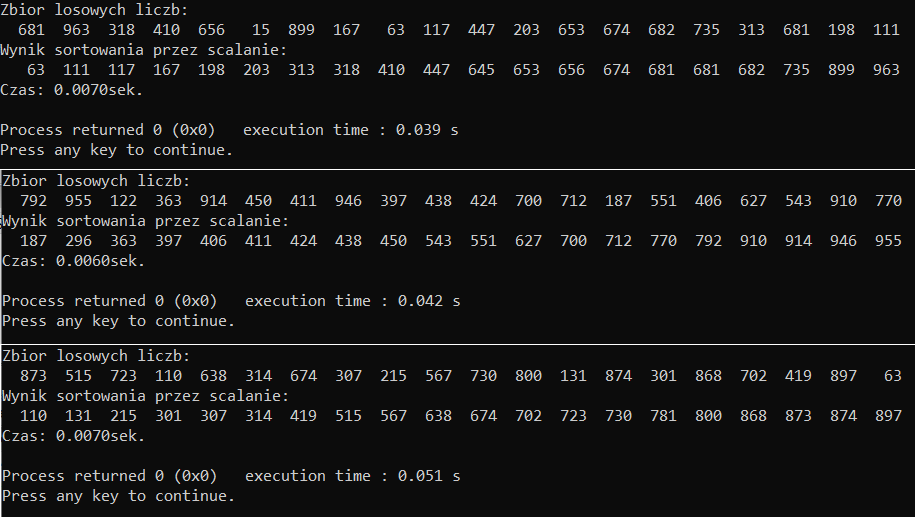
Sprawdzenie efektywności obydwu algorytmów z zależności od czasu.

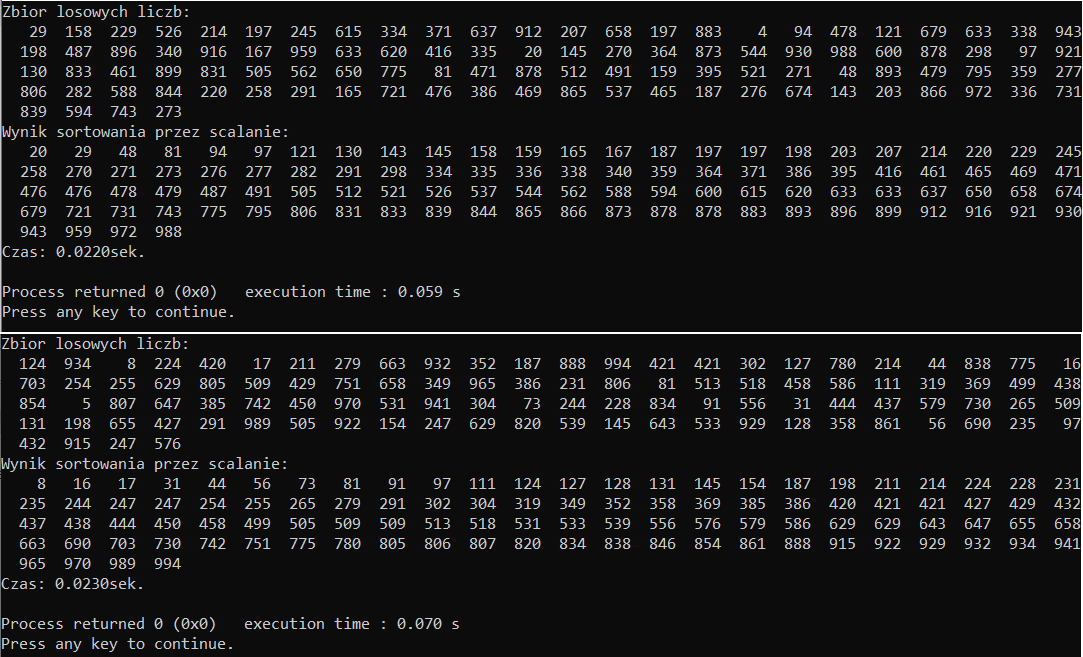
1. Sortowanie przez kopcowanie:





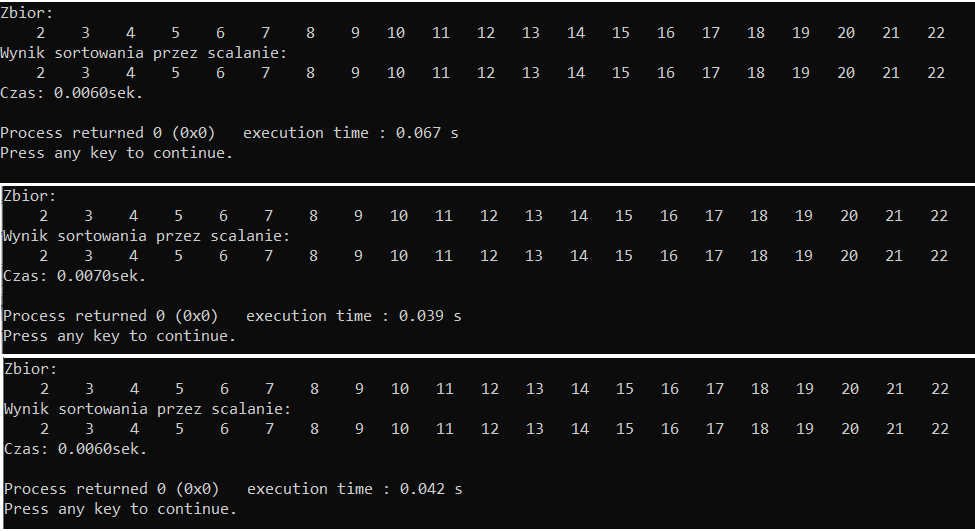
2. Sortowanie przez scalanie:

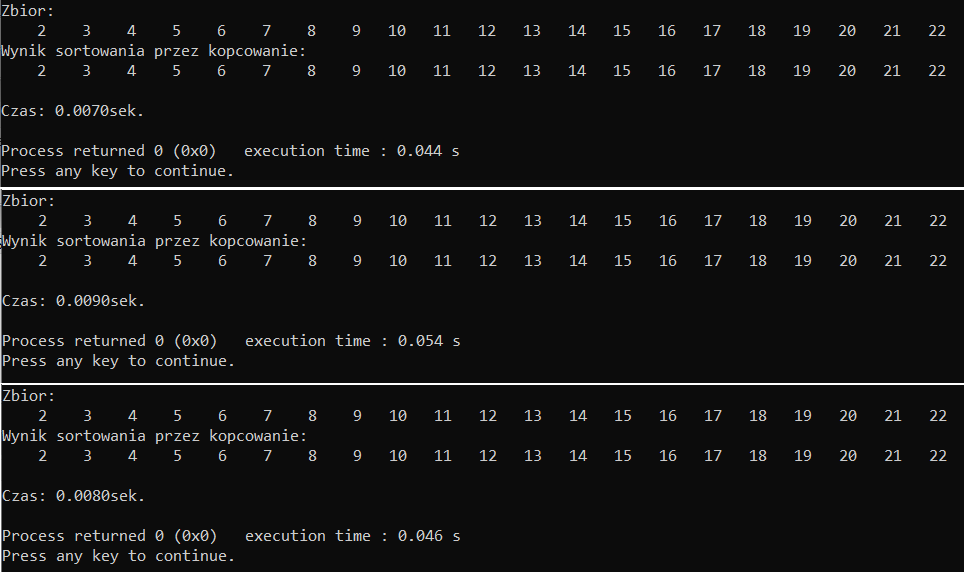


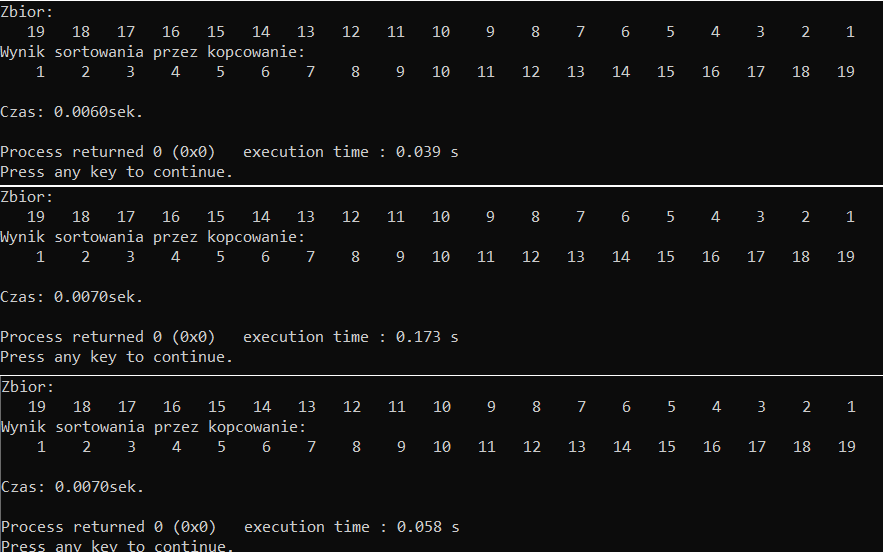


Testy zostały przeprowadzone na zbiorach 20 i 100 liczb w przedziale <0,1000> generowanym losowo przez program, można jednak zauważyć, iż sortowanie przez scalanie jest szybsze podczas od sortowania przez kopcowanie na oczekiwanej wersji. Średnim czasem jaki obydwa algorytmy potrzebują na posortowanie elementów w tych zbiorach to kolejno: przez scalanie dla 20 elementów-około 0.007s, natomiast dla 100 - około 0,0225, a dla sortowania przez kopcowanie jest to: około 0,008s dla 20 liczb oraz około 0,0235 dla 100. Chociaż sortowanie przez scalanie jest szybsze to zajmuje ono więcej pamięci ni przez kopcowanie. Ich czasy jakie potrzebują na posortowanie zbioru są podobne, dlatego że obydwa algorytmy posiadają złożoność obliczeniową wynoszącą O(nlog n). Optymistyczną wersją sortowania przez scalanie jest zbiór uporządkowany lub z niewielką ilością elementów nie na swoim miejscu. Pesymistyczną natomiast jest zbiór liczb posortowanych odwrotnie, a standardową/oczekiwaną wersją jest zbiór liczb o losowym ułożeniu. Można zauważyć jednak, że dla sortowania przez kopcowanie wersja z ciągiem odwrotnie posortowanych liczb jest szybsza, niż liczb posortowanych regularnie, więc można przyjąć, że to właśnie taka wersja jest najbardziej optymistyczną dla tego typu sortowania.

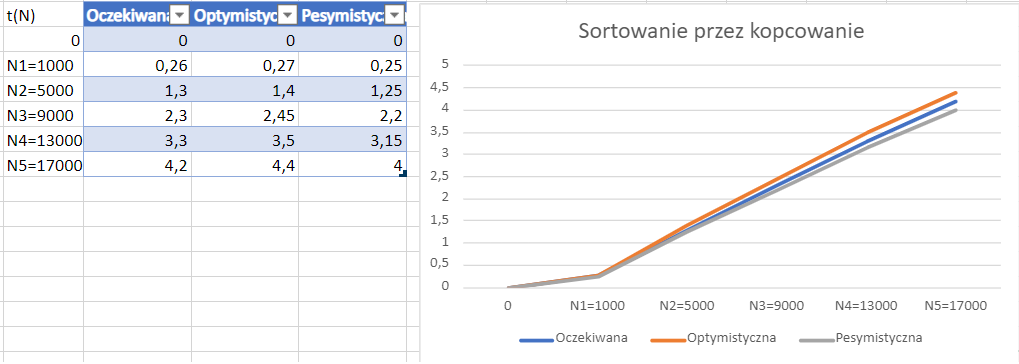
Poniżej mamy wyniki optymistycznych i pesymistycznych wersji sortowania.

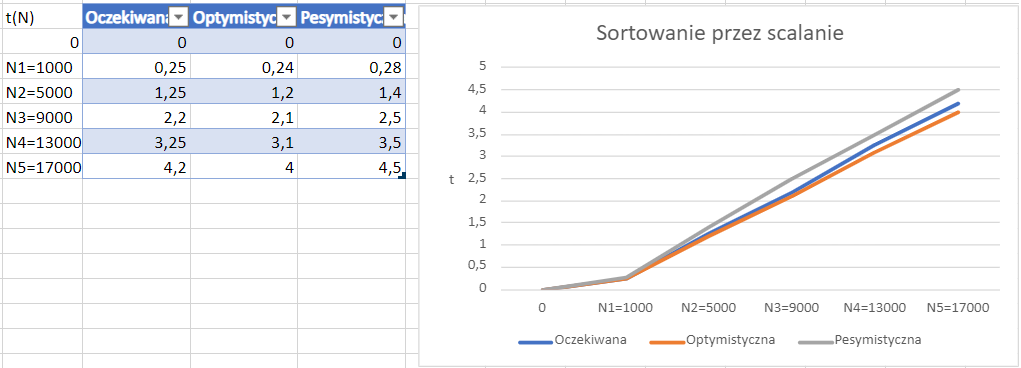






Wykresy poszczególnych wersji w zależności od wersji ułożenia zbioru:





Zadanie to pokazuje sposób działania i efektywności algorytmów sortowania o złożoności O(nlog n) w czasie, zależnie od ilości sortowanych liczb w zbiorze. Można zauważyć, że takie algorytmy są nieco wolniejsze podczas oczekiwanego przypadku sortowania (czyli ciągu losowo rozłożonym ciągu liczb), niż algorytmy o złożoności O(n^2). Ich przewagą natomiast jest lepsza złożoność pesymistyczna.

Żeby dokładniej wyliczyć średni czas działania zadanych algorytmów należałoby wykonać wiele testów na zbiorach o różnych rozmieszczeniach liczb. Niepewność pomiarowa w tym zadaniu wynika z ilości wielkości zbiorów branych pod uwagę, ich rozłożenie w zbiorze oraz od pracy komputera z różnymi danymi.