Sprawozdania – algorytmy sortowania

Adam Krzykała

Automatyka i robotyka II rok

Numer albumu: 235 411

Wprowadzenie

Projekt polegał na zapoznaniu się z metodami sortowania. W tym przypadku użyto następujących sortowań:

* Sortowanie przez scalanie,
* Sortowanie szybkie,
* Sortowanie introspektywne.

Sortowanie to proces uporządkowania danego zbioru o skończonym rozmiarze według ściśle określonego kryterium. Podstawowe sortowanie to sortowanie zbioru rosnąco, od najmniejszego do największego (tak jak w zadaniu). Głównym kryterium podziału algorytmów sortowania jest złożoność obliczeniowa. W praktyce jest to czas jaki jest potrzebny danemu algorytmowi na danych sprzęcie, aby zbiór mógł zostać posortowany.

Czas wykonywania algorytmu zależy w głównej mierze od użytego sprzętu oraz ustawień programowych. Jednakże w poniższym sprawozdaniu zbadano jak zmienia się czas wykonywania skryptu w zależności od użytego sortowania, ilości danych wejściowych, a także ich struktury. Przez strukturę danych rozumiemy fakt, czy jest to najlepszy, średni, czy najgorszy przypadek.

Oszacowania złożoności obliczeniowej: notacja Ο (dużego O), notacja Ω (omega) i notacja Θ (theta). W sprawozdaniu przykłady zostaną omówione na podstawie notacji dużego O.

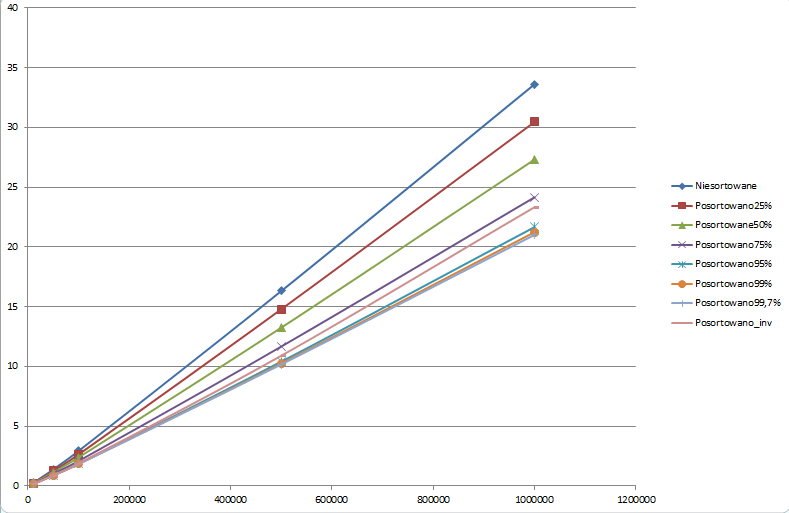
# Sortowanie przez scalanie

Pierwsze z użytych sortowań to sortowanie przez scalanie.

Algorytm ten dzieli zestaw na dwie równe części. Odbywa się dzielenie dopóki nie zostanie jeden element. Dla każdej z dwóch podzielonych tabel stosuje się łączenie. Podczas łączenia zostaje posortowany szereg. Odbywa się to w funkcji rekurencyjnie.

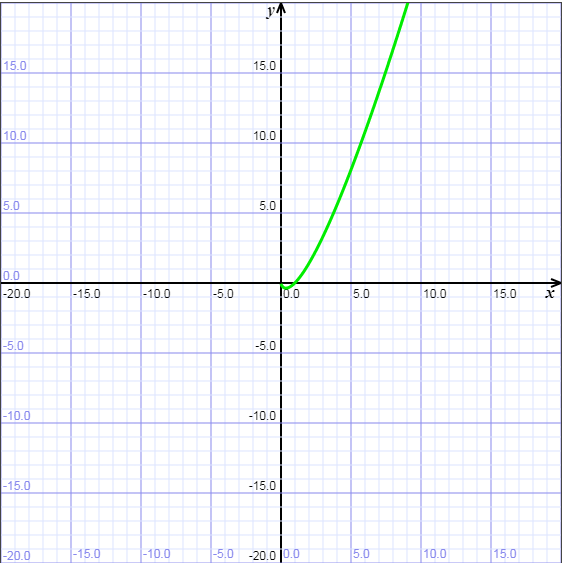
Sortowanie przez scalanie oznacz się jako drzewo o głębokości log2n, gdzie n to ilość danych wejściowych. Złożoność czasowa sortowania przez scalanie to 2nlog2n, czyli stosując notację O wynosi ono O(n\*logn).

Sortowanie przez scalanie



Wykres 1. Sortowanie przez scalanie dla różnych danych wejściowych i ich ilości

W sortowaniu przez scalanie można zaobserwować najlepszy przypadek dla zbioru już posortowanego. Zbiór w 99,7 % posortowany zajął najmniej czasu do pełnego uporządkowania. Im wyższy stopień uporządkowania zbioru – tym niższy czas wykonywania sortowania. Nie obserwuje się najgorszego przypadku, czyli posortowanej tablicy malejąco, jako przypadku, który wykonuje się najdłużej. Czas jego sortowania dla różnej ilości danych mieścił się w granicach czasów uzyskanych przy sortowaniu zbiorów w 75 % uporządkowanych oraz 95 % uporządkowanych. Najdłużej wykonywał się algorytm, gdy na wejściu pojawiła się tablica z losowymi danymi.



Wykres danej funkcji, która reprezentuje notację n\*log(n), dla pewnego zakresu jest bardzo zbliżony do funkcji liniowej. Takie właśnie wykresy obserwujemy na przykładzie na wykresie 1.

Wykres 2. f(x) = n\*log(n)

# Sortowanie szybkie

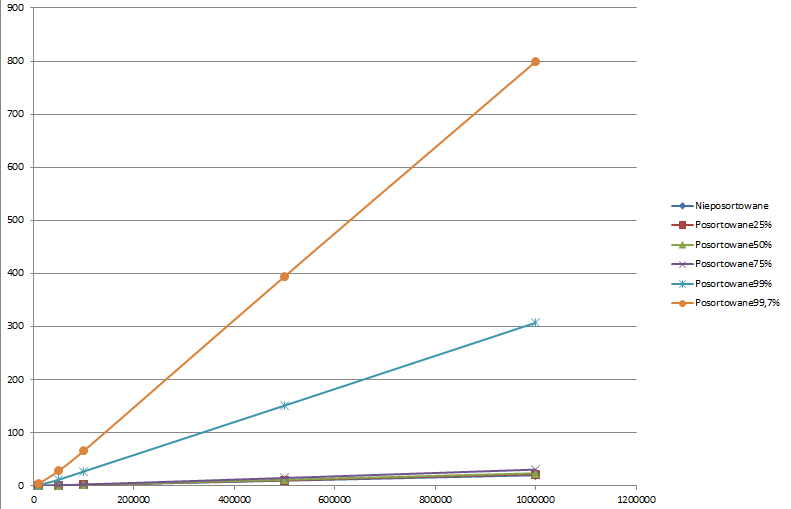
Sortowanie szybkie podobnie jak sortowanie przez scalanie reprezentuje grupę algorytmów działających na zasadzie „dziel i zwyciężaj”. Jest powszechnie używany nawet w standardowych bibliotekach programistycznych.

Sortowanie szybkie (quicksort) polega na dzieleniu tablicy na dwie podtablice. Do prawej tablicy przenoszone są wszystkie elementy mniejsze niż liczba środkowa, a do lewej większe. Następnie następuje rekurencyjny podział tablic, aż do momentu uzyskania pojedynczych elementów. Tak przebiega sortowanie szybkie.

Złożoność sortowania szybkiego jest zależna od wyboru parametru rozdzielającego. Występują trzy przypadki. Pierwszy jest optymistyczny i zakłada złożoność rzędu O (n\*logn). Następny jest przypadek przeciętny, który okazuje się być o 39% wolniejszy od przypadku optymistycznego. Na samym końcu wyróżnia się przypadek pesymistyczny, który ma kwadratową złożoność obliczeniową O(n2). Wybór skrajnego elementu do podziału charakteryzowałby się skrajnie długim czasem wykonywania szczególnie dla już posortowanych tablic. W naszym przypadku początkowym elementem dzielącym jest pierwszy element. To oznacza, że dla przypadku najgorszego, jak i średniego spodziewamy się dłuższego sortowania (im bardziej posortowany zbiór tym dłużej).

# 

Wykres 3. Wykres pokazujący długość obliczeń dla tablic 10 000 elementowych w zależności od stopnia posortowania początkowego

Wykres 4. Wykres zależności ilości danych od czasu wykonywania. Tutaj wyraźnie jest widoczne, że zwiększanie stopnia posortowania nie powoduje liniowego wzrostu czasu wykonywania.

Im dane na początku były w większym stopniu uporządkowane, tym czas ten się wydłużał. Przypuszczenia się potwierdziły. Pomiar dla tablicy uporządkowanej malejąco okazał się niemożliwy. Ten sam algorytm, który pracował dla wszystkich przypadków, teraz okazał się zawodny. Segmentation fault mógł zostać spowodowany zbyt dużą złożonością i zbyt małą ilością pamięci. Powodem mógł być sprzęt.

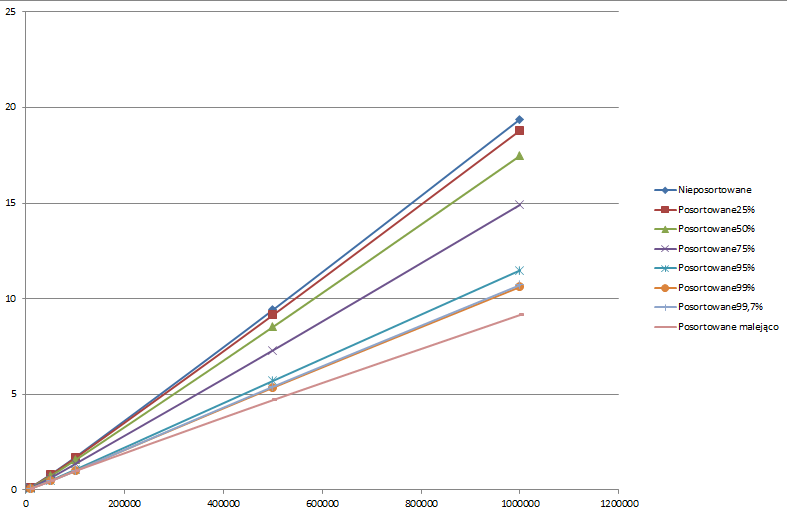
# Sortowanie introspektywne

Można powiedzieć, że sortowanie introspektywne jest swojego rodzaju sortowaniem hybrydowym. Zależnie od sytuacji wykorzystuje sortowanie przez kopcowanie (heapsort), bądź sortowanie przez wstawianie (insertsort). W sortowaniu tym wyeliminowano problem kwadratowej złożoności dla przypadków najgorszych i średnich. Ogranicza się zatem ilość wywołań rekurencyjnych poprzez wyliczenie maksymalnej głębokości rekurencji. Warunkiem wywołania rekurencyjnego jest rozmiar tablicy większy niż 9.

Celem sortowania introspektywnego jest zapewnienie logarytmiczno – liniowej złożoności obliczeniowej. Sortowanie introspektywne ma złożoność ogólną, czyli nawet w najgorszym przypadku wynosi O(n\*log2n).

W przeprowadzonych testach sortowanie introspektywne zapewniało najkrótsze czasy pracy.

SORTOWANIE INTROSPEKTYWNE



Wykres 5. Wyniki dla sortowania introspektywnego

# WNIOSKI

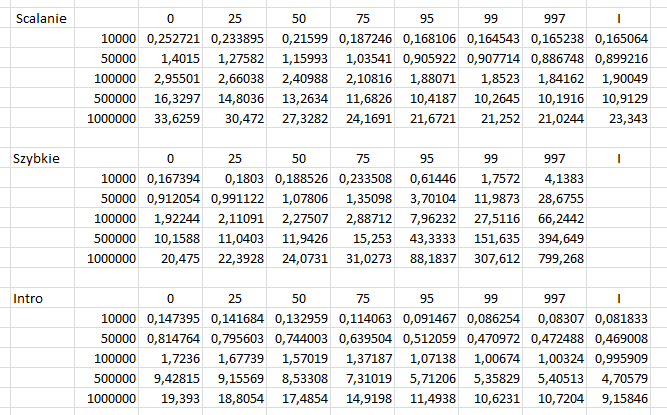
Wśród trzech wybranych algorytmów losowań najszybsze w każdym przypadku okazało się sortowanie introspektywne. Dzięki wprowadzonemu warunkowi, że tablice mniejsze niż 9 nie będą dzielone rekurencyjnie, pozbyto się złożoności kwadratowej, które ma miejsce w najgorszym przypadku sortowania szybkiego.

Sortowanie szybkie okazało się wydajniejsze w początkowej fazie dla mniejszej ilości danych aniżeli sortowanie przez scalanie. W późniejszej fazie okazało się, że złożoność kwadratowa wydłuża znacznie czas oczekiwania na rozwiązanie. W sortowaniu szybkim bez kontroli głębokości wywołań rekurencyjnych wraz ze zwiększaniem się uporządkowania zbioru, czas jego sortowania rósł. Również czas ten rósł dla zwiększającej się ilości danych.

Sortowanie przez scalanie zachowało złożoność obliczeniową (logarytmiczno-liniową).

Na samym początku działania programu potwierdzono funkcyjnie, że algorytmy sortowania działają w sposób prawidłowy.

Tabela zbiorcza wyników:



Bibliografia:

1. <http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/>
2. <http://www.algorytm.org/algorytmy-sortowania/sortowanie-przez-scalanie-mergesort.html>
3. <https://pl.wikipedia.org/wiki/Sortowanie_szybkie>
4. <https://pl.wikipedia.org/wiki/Sortowanie_introspektywne>