Sprawozdanie z ćwiczenia laboratoryjnego

Tomasz Mikołajewski

2014

Spis treści

1 Wprowadzenie

Niniejszy dokument powstał w ramach przedmiotu Projektowanie Algorytmów i Metod Sztucznej Inteligencji. Jest on rezultatem przeprowadzonych ćwiczeń w laboratorium oraz pracy wykonanej w domu.

Ćwiczenie polegało na przeprowadzeniu analizy złożoności algorytmów sortowania. Podczas tekstu sprawdzano algorytmy: szybkiego sortowania, sortowania przez kopcowanie oraz przez scalanie. Podczas wykonywania czynności sortowania liczony był czas wykonania operacji. Testowanie powtarzano wielokrotnie, aby uzyskać ilość danych pozwalającą na wyznaczenie przybliżonej funkcji opisującej złożoność algorytmu.

2 Kod programu

Program został oparty o wcześniej stworzone pliki zawierające kod potrzebny do wykonania *benchmark-u* podprogramu, czyli określenia czasu jaki potrzebny jest na jego wykonanie. Główny program zawiera funkcję otwierającą pliki z danymi oraz zapisującą rezultat operacji do pliku. Zdecydowanie ułatwia to prace przy analizie zadanego algorytmu.

Algorytmy, które były testowane to sortowanie:

- szybkie
- przez scalanie
- przez kopcowanie

3 Pomiary

Pomiary zostały przeprowadzone na dużych plikach zawierających zmienne typu *int*. Każdy z algorytmów był testowany dla co najmniej 5 różnych rozmiarach problemów powtórzonych dla dwóch różnych plików. Aby otrzymać rzetelny pomiar, każdy z testów był przeprowadzony 3-krotnie. Oznacza to iż każdy z algorytmów był testowany co najmniej 30 razy.

4 Wyniki pomiarów

Po przeprowadzeniu serii pomiarów otrzymano wyniki przedstawione w tabelach. Na podstawie wyników utworzono wykresy zamieszczone poniżej.

4.1 Algorytm sortowania szybkiego

Jest to algorytm wykorzystujący rekurencję i działający według zasady dziel i zwyciężaj. Jest bardzo często stosowany ponieważ jego implementacja jest stosunkowo prosta, a należy do jednych z najszybszych algorytmów.

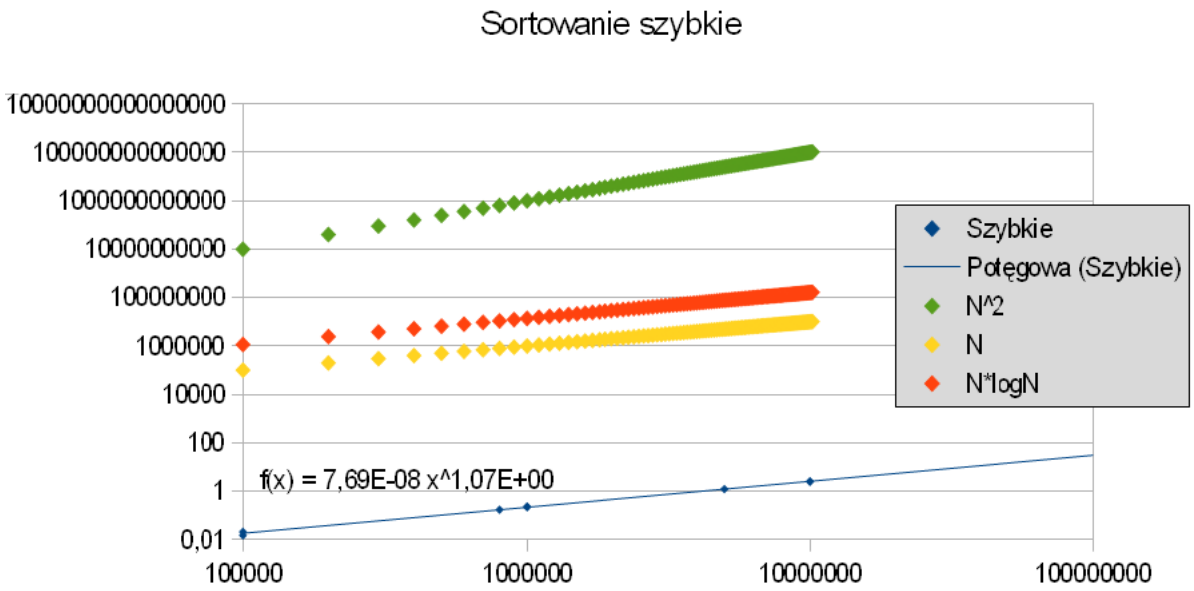
4.1.1 Powtarzające się liczby-sortowanie szybkie

Doświadczenie zostało przeprowadzone z użyciem sekwencji liczb wygenerowanych za pomocą powtarzającej się pętli.

Wielkosc	1	2	3	Średnia
10000000	2,531	2,609	2,562	2,567
10000000	2,484	2,437	2,453	2,458
5000000	1,219	1,234	1,219	1,224
5000000	1,235	1,188	1,219	1,214
1000000	0,219	0,218	0,266	0,234
1000000	0,203	0,235	0,219	0,219
800000	0,172	0,171	0,172	0,172
800000	0,172	0,172	0,172	0,172
100000	0,015	0,015	0,015	0,015
100000	0,016	0,031	0,016	0,021

4.1.2 Wersja pesymistyczna i optymistyczna-szybkie

Doświadczenie przeprowadzone w celu sprawdzenia działania algorytmu w skrajnych przypadkach.

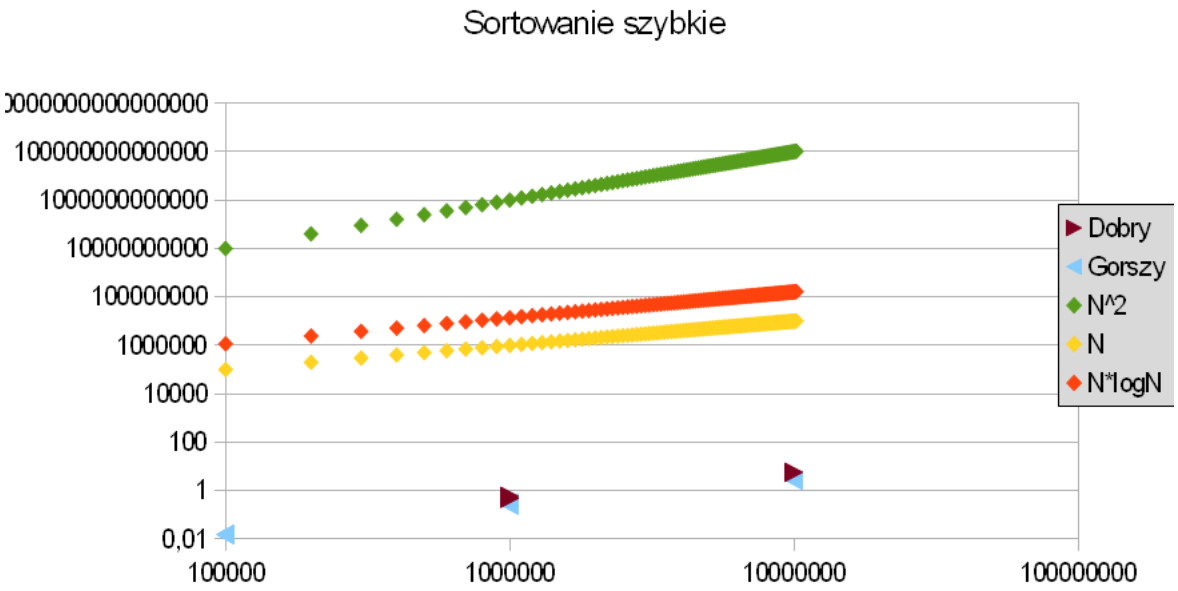


4.2 Algorytm sortowania przez scalanie

Algorytm ten rozkłada tablicę bądź listę na pojedyncze elementy, które z definicji muszą być posortowane, a następnie składa posortowane fragmenty aż do otrzymania pełnej posortowanej tablicy.

4.2.1 Powtarzające się liczby-scalanie

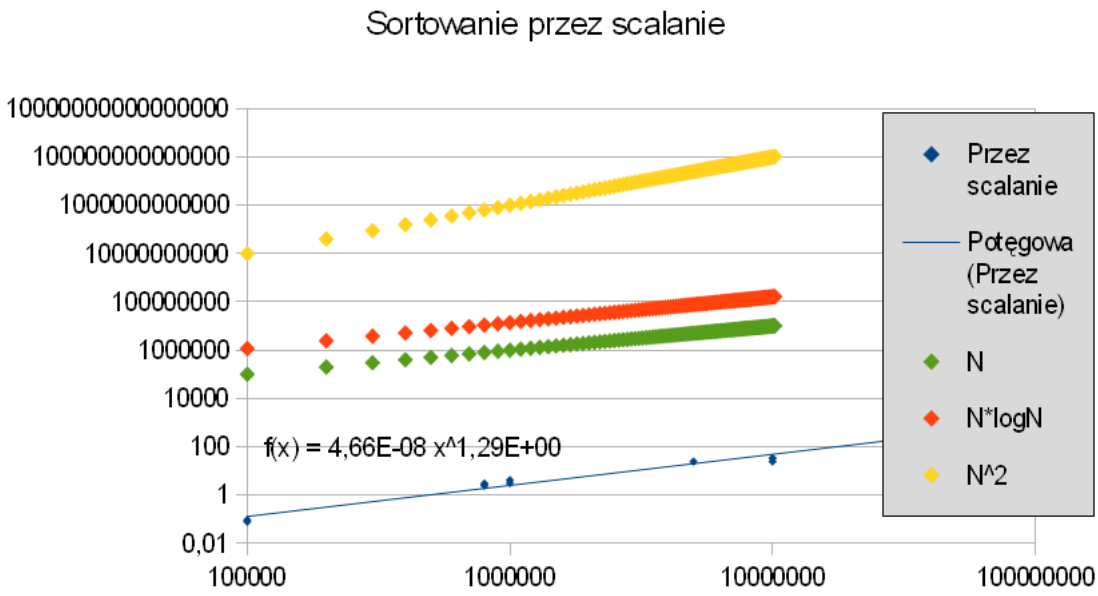
Doświadczenie zostało przeprowadzone z użyciem sekwencji liczb wygenerowanych za pomocą powtarzającej się pętli.



Wielkosc	1	2	3	Średnia
10000000	24,344	7,047	7,016	12,802
10000000	33,859	25,328	7,062	22,083
5000000	24,172	12,766	21,75	19,563
5000000	24,843	27,562	28,453	26,953
1000000	3,125	4,813	4,312	4,083
1000000	4,14	3,343	2,812	3,432
800000	2,438	3,234	2,719	2,797
800000	2,969	3,344	3,766	3,360
100000	0,078	0,078	0,578	0,245
100000	0,093	0,063	0,579	0,245

4.2.2 Wersja pesymistyczna i optymistyczna-scalanie

Doświadczenie przeprowadzone w celu sprawdzenia działania algorytmu w skrajnych przypadkach.

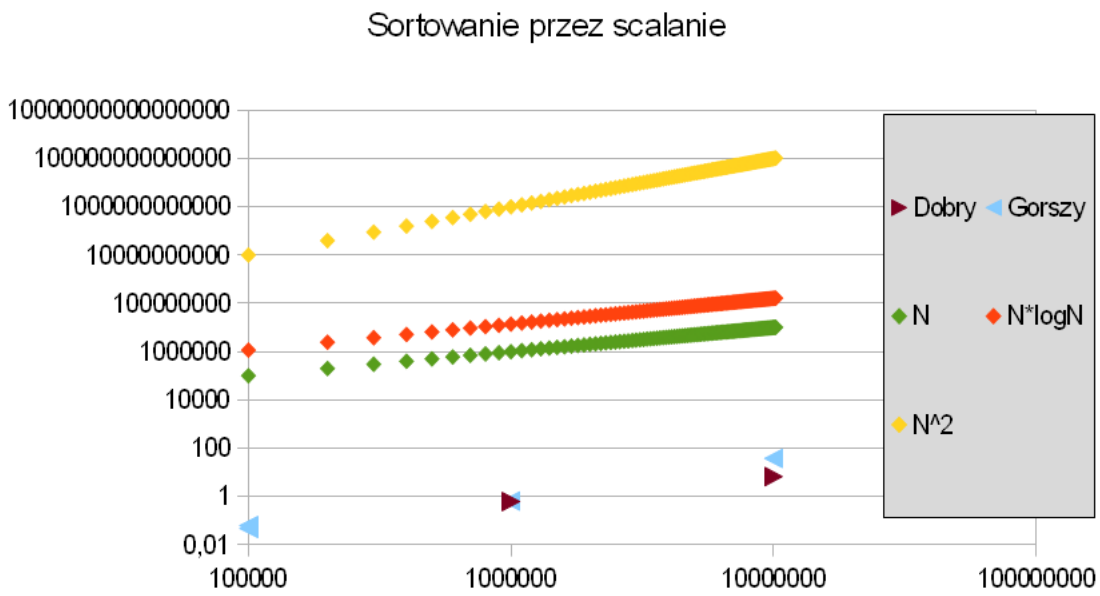


4.3 Algorytm sortowania przez kopcowanie

Jest to algorytm rozkładający tablicę na strukturę zwaną kopcem. Powstaje ona poprzez porządkowanie elementów w strukturę hierarchiczną. Na szczycie kopca znajdują się elementy o największej wadzę, a poniżej nich mniej znaczące. Z tak powstałego kopca przenosi się element umiejscowiony na samym szczycie do tablicy w której wszystkie obiekty będą już posortowane.

4.3.1 Powtarzające się liczby-kopcowanie

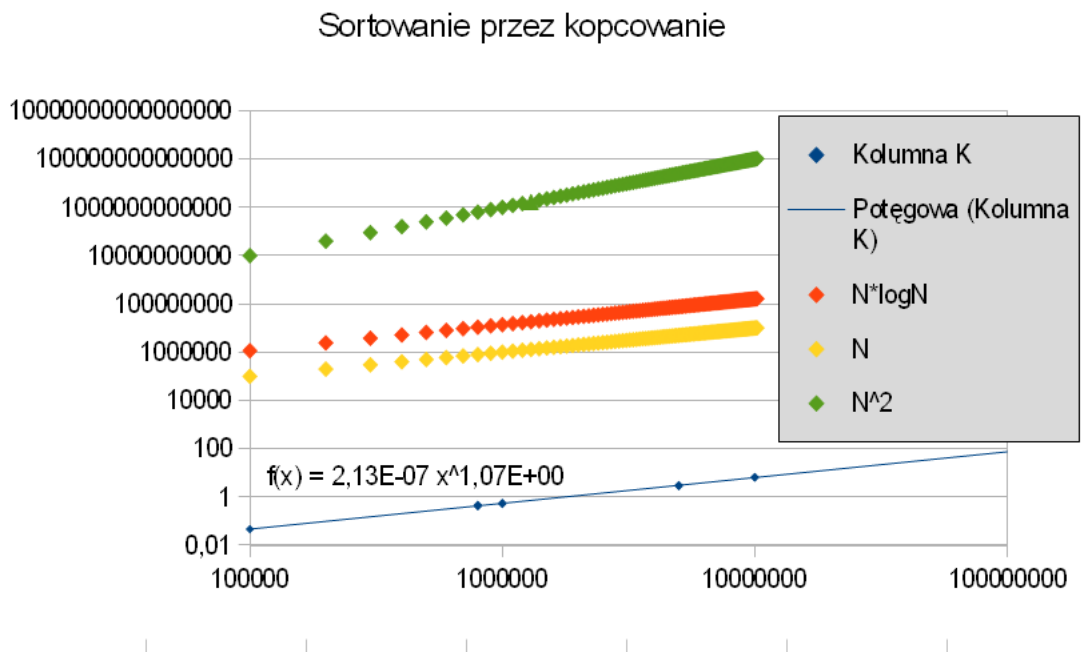
Doświadczenie zostało przeprowadzone z użyciem sekwencji liczb wygenerowanych za pomocą powtarzającej się pętli.



Wielkosc	1	2	3	Średnia
10000000	6,562	6,375	6,375	6,437
10000000	6,157	6,766	7,203	6,709
5000000	3	3,343	3,032	3,125
5000000	2,859	2,797	2,812	2,823
1000000	0,625	0,531	0,547	0,568
1000000	0,5	0,516	0,531	0,516
800000	0,422	0,562	0,437	0,474
800000	0,406	0,422	0,406	0,411
100000	0,047	0,062	0,031	0,047
100000	0,078	0,032	0,031	0,047

4.3.2 Wersja pesymistyczna i optymistyczna-kopcowanie

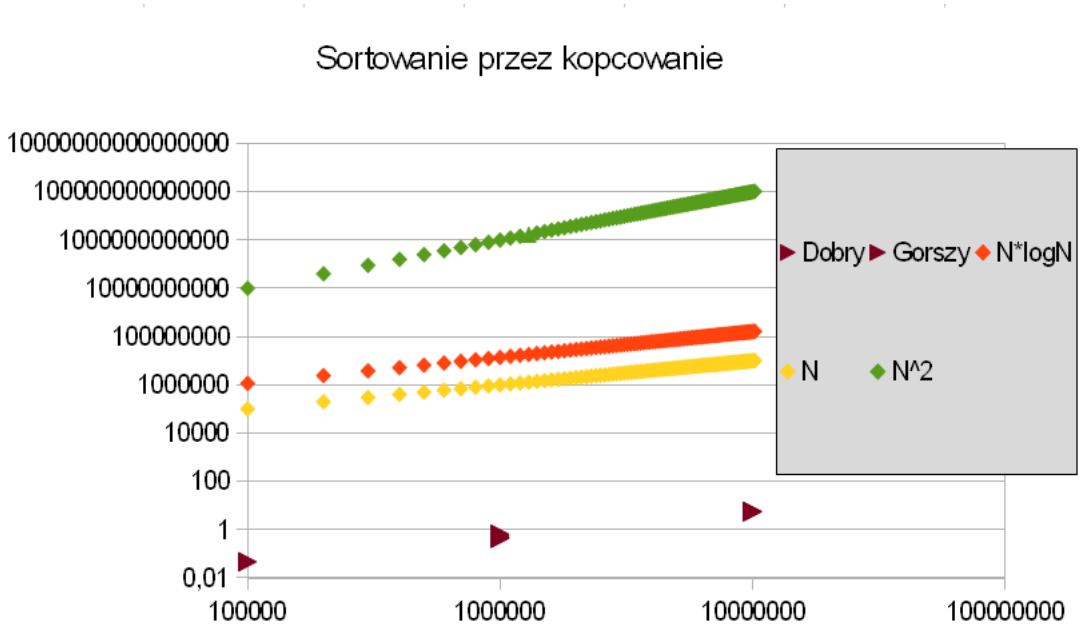
Doświadczenie przeprowadzone w celu sprawdzenia działania algorytmu w skrajnych przypadkach.



4.4 Porównanie algorytmów

W ramach porównania algorytmów przedstawiono wszystkie dane na jednym wykresie.

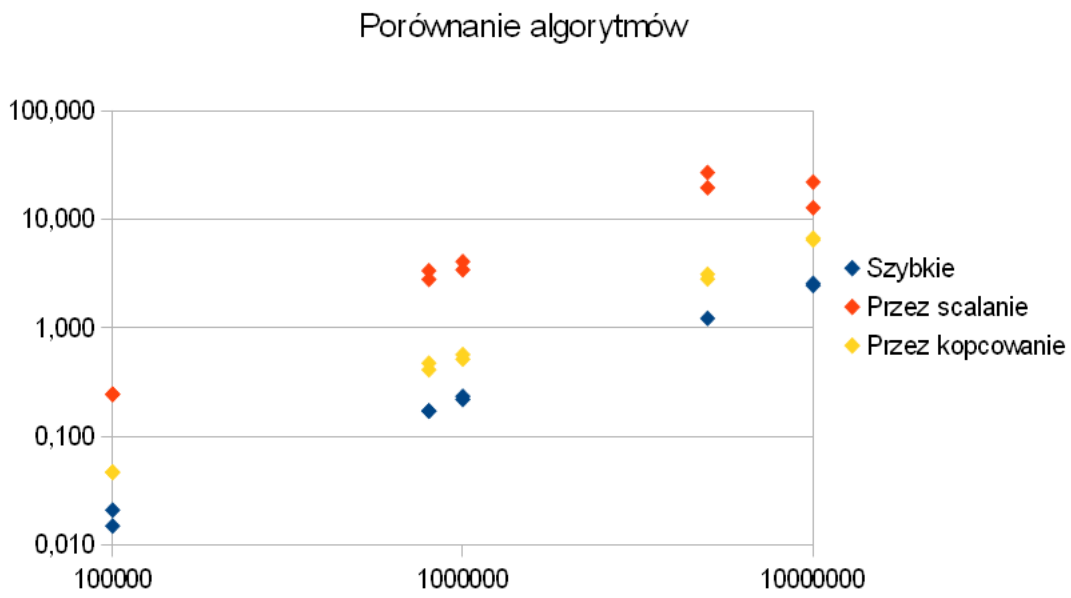
Jak widać na załączonym wykresie przedstawione algorytmy sortowanie cechują się podobną złożonością obliczeniową. Wraz ze wzrostem ilości elementów, ilość potrzebnego czasu rośnie podobnie dla każdego z algorytmów. Jednakże podczas działania programu istotny również jest czas potrzebny na wykonanie pojedynczych operacji sortowania, a w tym wypadku najbardziej skuteczny okazuje się algorytm sortowania szybkiego.



5 Wnioski

Na podstawie danych i wykresów można wyciągnąć następujące wnioski:

- wszystkie przedstawione algorytmy posiadają podobną złożoność obliczeniową
- z pominięciem wersji pesymistycznej wszystkie testowane algorytmy mają złożoność obliczeniową klasy N*log(N)
- nie tylko klasa złożoności obliczeniowej wpływa na czas realizacji algorytmu
- sortowanie przez kopcowanie oraz sortowanie szybkie posiadają dodatkową zaletę, ponieważ są to sortowania wykonywane w miejscu, czyli nie potrzebują dużych zasobów pamięci.
- sortowanie szybkie okazało się najbardziej efektywne (przy założeniu o pomijaniu wersji pesymistycznej).
- podczas realizacji ćwiczenia pozyskano informację na temat stabilności wykonanych sortowań. Okazało się, że jedynie kopcowanie przez scala-



nie jest stabilne, co oznacza że nie zamienia dwóch elementów o takiej samej wartości, ze względu na którą przeprowadza się sortowanie.

- algorytm sortowania przez scalanie okazał się bardzo prosty w modyfikacji dzięki czemu było możliwe tworzenie struktur posortowanych odwrotnie

- zauważono znaczną różnicę przy różnych ustawieniach liczb, ale tylko przy kopcowaniu. Co ciekawe dla algorytmu szybkiego teoretycznie gorszy przypadek okazał się korzystniejszy.