Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Algorytmy sortowania

Prowadzący: Mgr inż. Marcin Ochman

Autor: Adrian Sobecki

Numer indeksu: 248942

Termin zajęć: Wtorek 15:15-16:55

Termin oddania: 14.04.2020

Wstęp

Naszym zadaniem była implementacja trzech algorytmów sortowania. Wybrane przeze mnie algorytmy to quicksort, sortowanie przez scalanie (merge sort) oraz sortowanie introspektywne (introspective sort / introsort). Po implementacji każdego z algorytmu następowała weryfikacja poprawności, a w razie błędów ich poprawa. Ostatnim etapem były testy efektywności i opracowanie otrzymanych wyników.

Opis badanych algorytmów

a) Merge sort

Sortowanie polegające na rekurencyjnym wywołaniu funkcji, która dzieli naszą tablice na dwie, aż do uzyskania tablic jednoelementowych. Następnie następuje scalanie dwóch posortowanych tablic (począwszy od pojedynczych elementów) w jedną. Złożoność obliczeniowa sortowania przez scalanie wynosi $n \log n$.

b) Quicksort

Sortowanie polegające na wybraniu pivota i uporządkowaniu pozostałych elementów tak, aby po lewej stronie były elementy nie większe od niego, a po prawej nie mniejsze. Następnie procedura zostaje wywołana dla elementów na lewo od pivota oraz na prawo od pivota, aż do uzyskania pojedynczych elementów. Średnia złożoność obliczeniowa quicksorta wynosi $n \log n$ jednak w przypadku najgorszym (wybranie pivota jako liczby największej/najmniejszej ze zbioru) złożoność wzrasta do n^2.

c) Introsort

Sortowanie wykorzystujące 3 algorytmy sortowania takie jak quicksort, heap sort oraz dowolny radzący sobie szybko z małymi zbiorami np. insertion sort. Wykorzystujemy tutaj pojęcie głębokości wywołań rekurencyjnych, którego wartość decyduje o wyborze algorytmu pomiędzy quicksortem a heap sortem. Dla małych zbiorów wywoływany jest 3 algorytm np. sortowania przez wstawianie. Taka realizacja daję nam stałą złożoność obliczeniową wynoszącą $n \log n$.

Po wcześniejszym sprawdzeniu poprawności działania algorytmów sortowania nadszedł czas na sprawdzenie efektywności. W tym celu napisałem specjalną funkcję przyjmującą 4 argumenty, które pozwalają na realizację każdego badanego przypadku. Testy przebiegły bezproblemowo.

Uzyskane wyniki

a) Wszystkie elementy tablicy losowe

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,208468s	1,05219s	2,31883s	11,9949s	26,9100s
quicksort	0,111668s	0,647278s	1,27559s	6,81282s	13,4780s
introsort	0,094785s	0,557460s	1,20376s	5,97206s	12,0328s

b) Tablica posortowana w 25%

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,194469s	1,05115s	2,17419s	12,0128s	22,8290s
quicksort	0,108710s	0,617350s	1,22370s	6,49766s	12,5434s
introsort	0,084773s	0,507643s	1,08510s	5,68181s	11,6000s

c) Tablica posortowana w 50%

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,177549s	0,968447s	1,95573s	10,3393s	21,6451s
quicksort	0,137632s	0,999326s	2,54920s	21,6103s	57,9441s
introsort	0,123706s	0,814822s	1,82911s	9,58538s	19,1658s

d) Tablica posortowana w 75%

	Ilość elementów				
Algorytm sortowania	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,157578s	0,815818s	1,76228s	9,39291s	19,4709s
quicksort	0,080784s	0,513601s	1,00832s	5,11336s	9,80280s
introsort	0,066810s	0,382976s	0,772935s	4,20479s	8,77951s

e) Tablica posortowana w 95%

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,146611s	0,807804s	1,63363s	8,80046s	18,1575s
quicksort	0,066813s	0,415883s	0,795874s	4,09007s	8,42548s
introsort	0,055851s	0,316156s	0,691153s	3,39191s	6,76991s

f) Tablica posortowana w 99%

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,176516s	0,763956s	1,58875s	8,54612s	16,9597s
quicksort	0,058846s	0,400924s	0,723057s	3,79987s	7,49498s
introsort	0,045877s	0,264261s	0,536566s	2,75164s	5,94015s

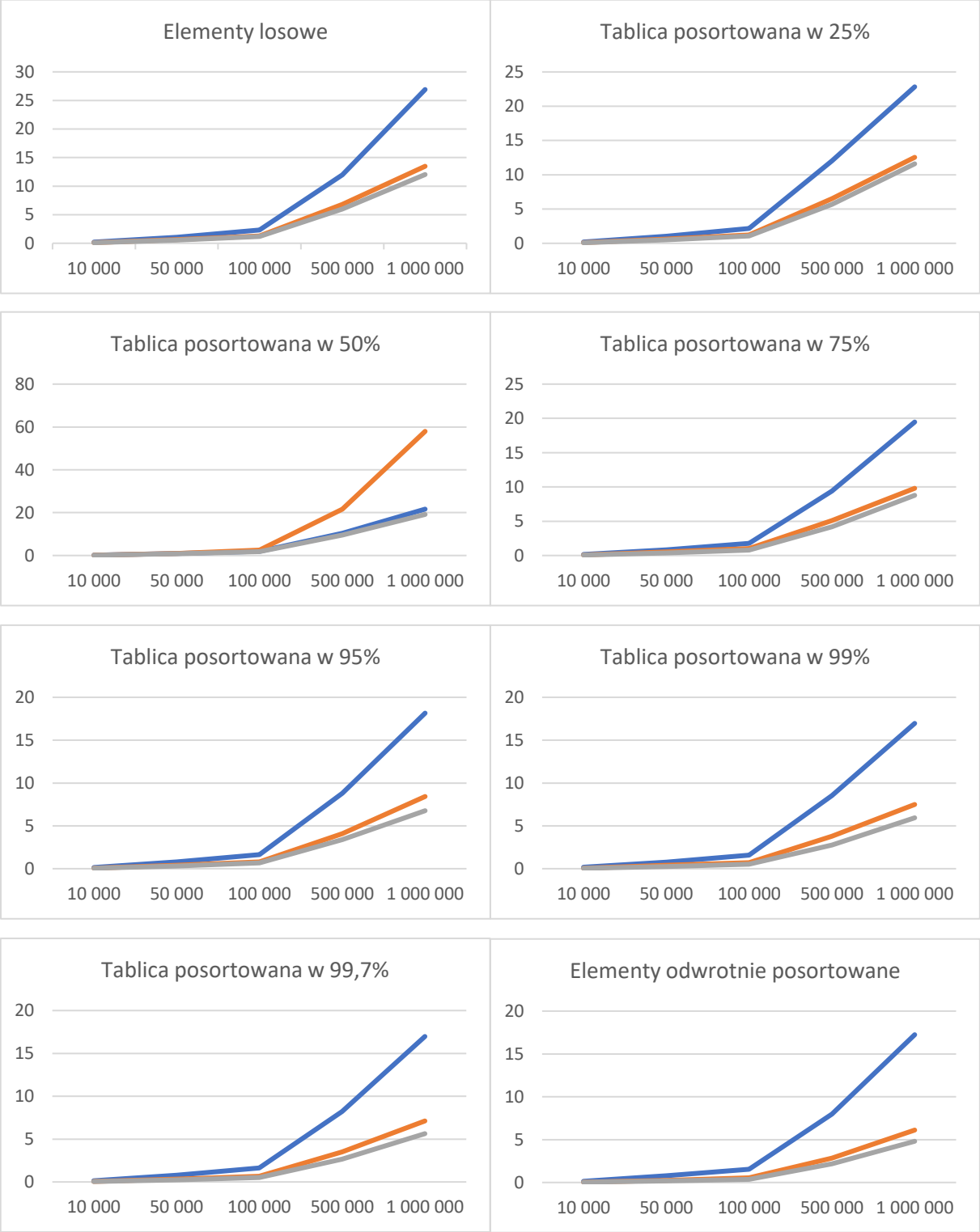
g) Tablica posortowana w 99,7%

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,141621s	0,771898s	1,63262s	8,21702s	16,9736s
quicksort	0,056847s	0,336098s	0,667217s	3,51460s	7,11498s
introsort	0,047870s	0,242352s	0,513644s	2,66588s	5,63195s

h) Tablica odwrotnie posortowana

Algorytm sortowania	**Ilość elementów**				
	10 000	50 000	100 000	500 000	1 000 000
merge sort	0,139627s	0,777957s	1,53290s	7,96769s	17,2439s
quicksort	0,039893s	0,254321s	0,553520s	2,84639s	6,11664s
introsort	0,030917s	0,182512s	0,378988s	2,17619s	4,80814s

Wykresy zależności czasu sortowania w sekundach od ilości elementów dla poszczególnych przypadków przy oznaczeniach merge sort quicksort introsort



Wnioski

Otrzymane wyniki są zgodne z oczekiwaniami. Introsort okazał się najszybszy, a quicksort ustępował merge sort tylko w jednym przypadku (tablica posortowana w 50%). Dzieje się tak, ponieważ za pivot zostaje uznany jeden z największych elementów ze zbioru. Złożoność obliczeniowa rośnie i quicksort traci na wydajności. Wraz ze wzrostem początkowego posortowania tablicy algorytmy sortowania przeważnie radziły sobie szybciej ze swoim problemem. Wyjątkiem jest wcześniej wspomniany przypadek posortowania w 50% oraz czas, który potrzebował merge sort dla 10000 elementów przy posortowaniu wynoszącym 99%. Odstępstwo to miało miejsce tylko raz, także można uznać je za przypadek.

Literatura

https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie

https://www.geeksforgeeks.org/merge-sort/

https://pl.wikipedia.org/wiki/Sortowanie_szybkie

http://www.algorytm.org/algorytmy-sortowania/sortowanie-szybkie-quicksort/quick-1-c.html

https://www.geeksforgeeks.org/know-your-sorting-algorithm-set-2-introsort-cs-sorting-weapon/

https://pl.wikipedia.org/wiki/Sortowanie_introspektywne

https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie

https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie