Kornel Zieliński nr albumu 230979

PROJEKTOWANIE ALGORYTMÓW I METOD SZTUCZNEJ INTELIGENCJI

PROJEKT 2 – SORTOWANIE

Cel projektu

Zapoznanie się z trzema typami sortowania (przez scalanie, quicksort, introspektywne), zaimplementowanie ich oraz wykonanie testów efektywności.

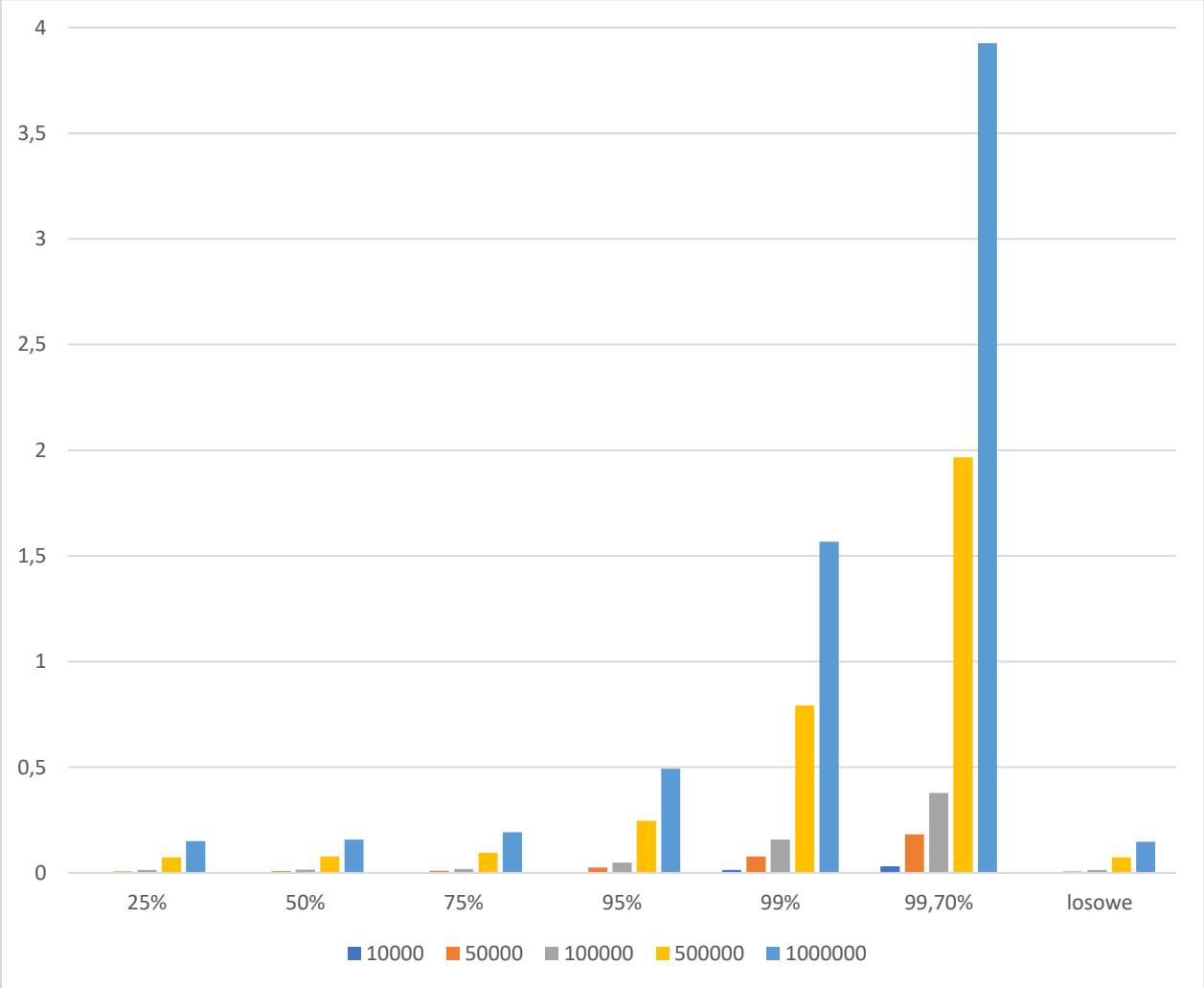
Przebieg ćwiczenia

1) Zaimplementowanie wszystkich trzech typów sortowania
2) Zatwierdzenie poprawności ich działania poprzez wykonywanie testów na niedużych tablicach oraz weryfikowanie wyników
3) Wykonanie testów efektywności dla 100 tablic o rozmiarach 10000, 50000, 100000, 500000 oraz 1000000 o elementach losowych
4) Wykonanie dokładnie takich samych testów dla tablic z 25%, 50%, 75%, 95%, 99%, 99,7% początkowych elementów już posortowanych
5) Wykonanie tych samych testów dla tablic posortowanych w odwrotnej kolejności

Quick Sort

Metoda sortowania polegająca na wybraniu tzw. elementu osiowego (piwot) i dzielenia tablicy względem niego na dwie podtablice z elementami mniejszymi oraz większymi od piwota. Proces dzielenia czyli tzw. partycjonowanie wykonywane jest poprzez rekurencję, aż do momentu uzyskania tablic jednoelementowych nie wymagających sortowania. Jego średnia złożoność obliczeniowa to $O(n \log(n))$, a w najgorszym przypadku to $O(n^2)$. Wyniki sortowania zostały umieszczone w poniższej tabeli oraz wykresie.

	25%	50%	75%	95%	99%	99,70%	losowe	odwrotne
10000	0,0013798	0,001715	0,0018099	0,00466107	0,01368273	0,031516	0,00137	0,0935131
50000	0,0072394	0,008069	0,0094513	0,0253236	0,07765724	0,182336	0,00716	1,1749337
100000	0,01458	0,016102	0,0191564	0,04930454	0,15725348	0,377828	0,01434	3,019390
500000	0,0736139	0,07792	0,0958642	0,24659175	0,79235785	1,966258	0,07293	22,304353
1000000	0,1511666	0,15802	0,1926173	0,49283623	1,5667743	3,926726	0,14813	50,650337

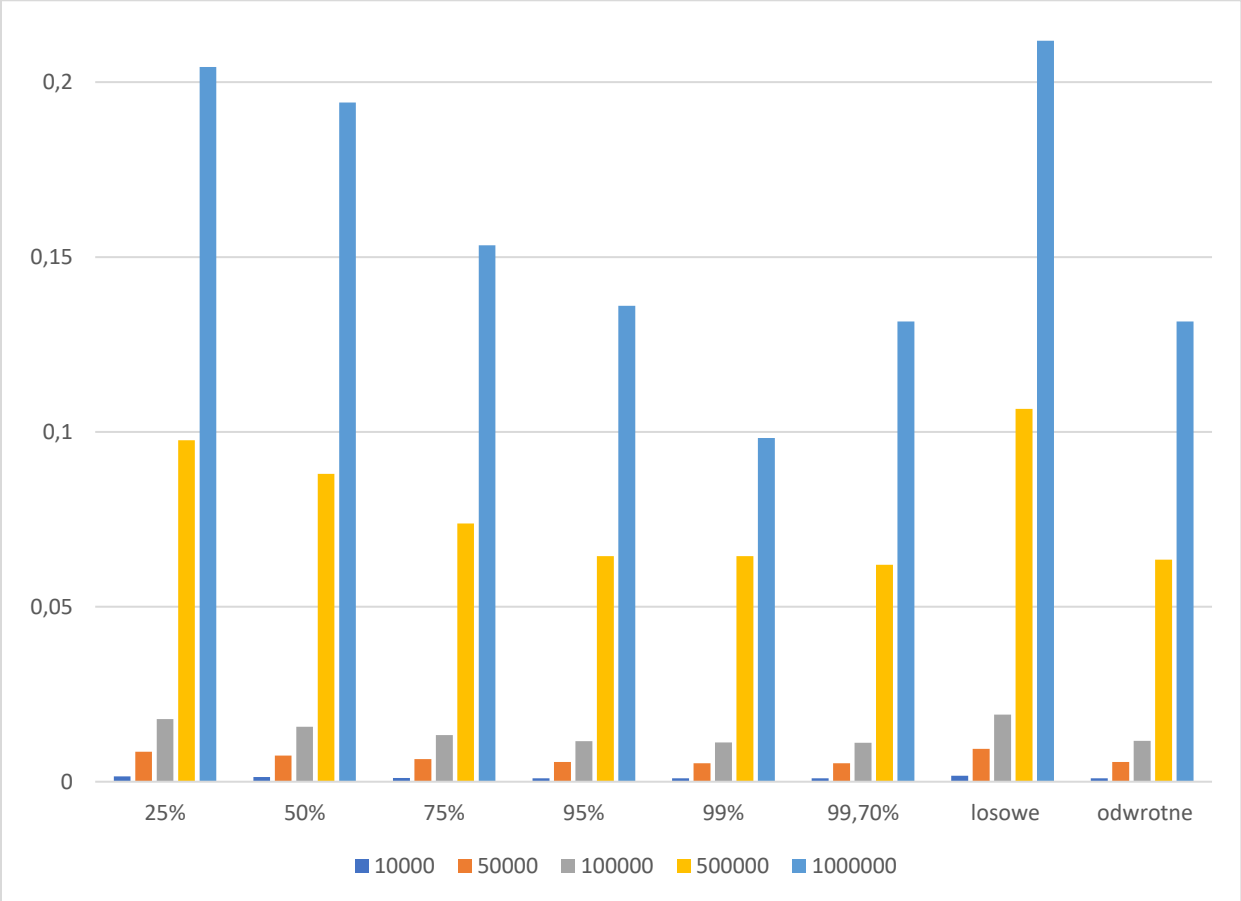


Przez wzgląd na duże różnice, na wykresie nie zostały ujęte wyniki tablic posortowanych odwrotnie. Rozbieżność ta wynika z nagromadzenia najgorszych przypadków dla tego typu sortowania. Sortowanie było wykonywane na posortowanej już tablicy, do tego jako piwot przyjęty jest zawsze pierwszy element tablicy, który w tym przypadku był elementem największym co doprowadziło do ogromnego wręcz spadku efektywności sortowania. Przy tablicach wstępnie posortowanych również można zobaczyć spadki efektywności tegoż algorytmu. Najlepsze wyniki quick sort uzyskał dla tablic o losowych elementach, gdzie był najszybszym algorytmem wykorzystanym przeze mnie w tym ćwiczeniu.

Sortowanie przez scalanie

Sortowanie wykorzystujące rekurencje do ciągłego podziału tablicy, na dwie podtablice az do uzyskania tablic jednoelementowych. Na końcu tablice są poddawane czynności nazywanej scalaniem dając w rezultacie tablicę posortowaną. Dla tego typu sortowania zarówno średni jak i najgorszy przypadek złożoności obliczeniowej to O(n log(n)), co pozwala założyć jego bardzo szeroką funkcjonalność. Wyniki testów zostały ujęte na poniższych tabeli oraz wykresie.

	25%	**50%**	**75%**	**95%**	**99%**	**99,70%**	**losowe**	**odwrotne**
10000	0,001545	0,0013666	0,001113	0,000961	0,001016	0,000938	0,0016767	0,00099734
50000	0,008593	0,0074724	0,006469	0,0056	0,005304	0,00529	0,0094096	0,00561923
100000	0,017898	0,0157172	0,013378	0,011555	0,011217	0,01113	0,0192223	0,01166056
500000	0,097593	0,0880199	0,073825	0,064513	0,064467	0,062029	0,1065861	0,06350842
1000000	0,204355	0,1941765	0,153363	0,136066	0,098298	0,131579	0,211843	0,13154562

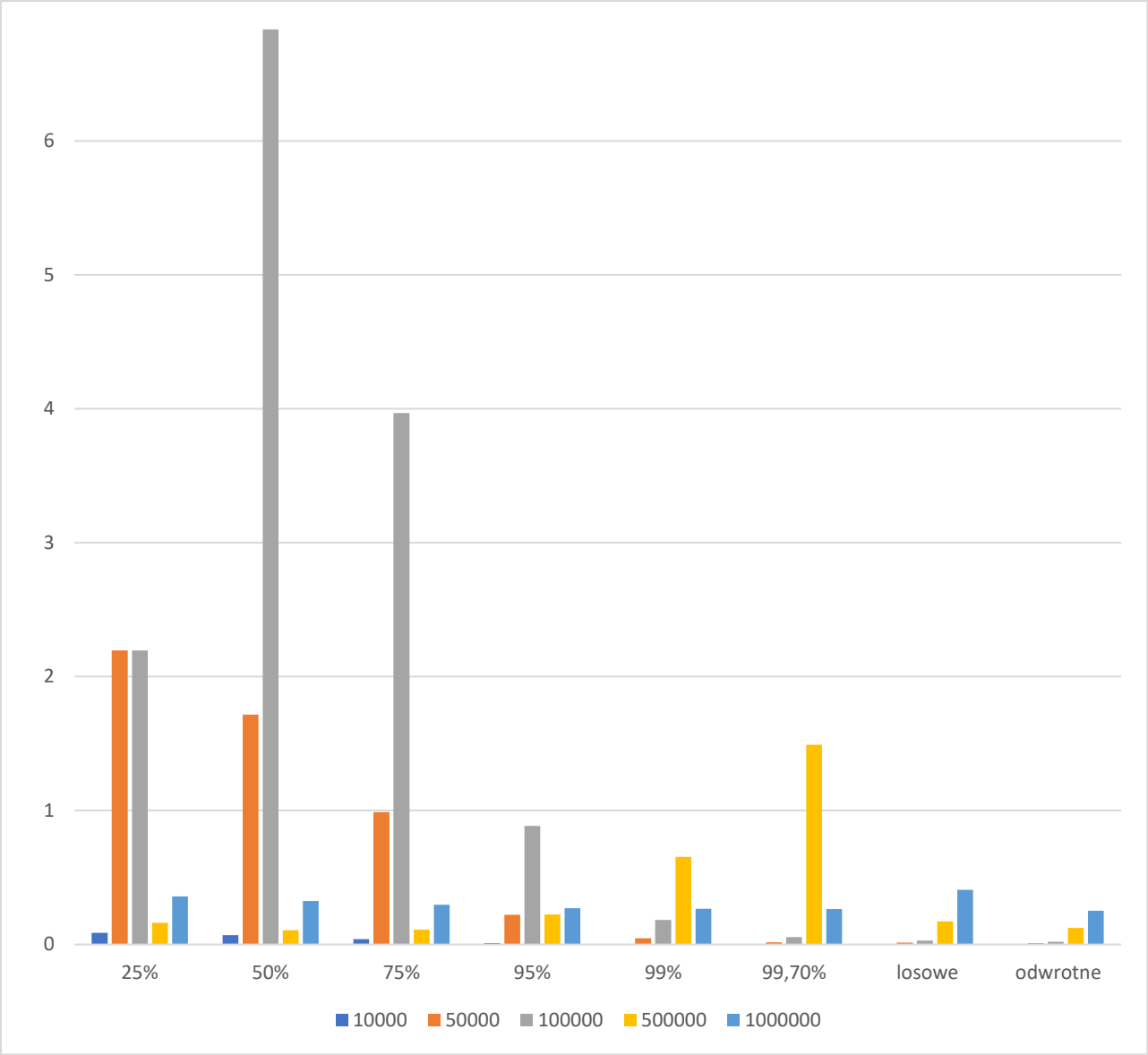


Dzięki temu ćwiczeniu tzw. merge sort dał mi się poznać jako najbardziej wszechstronny. Ze wszystkich użytych algorytmów najlepiej poradził sobie z tablicami wstępnie posortowanymi oraz tymi posortowanymi w odwrotnej kolejności, nie ustępując przy tym nazbyt quick sortowi przy tablicach z elementami rozmieszczonymi losowo. Można tutaj zauważyć jego działania w średniej oraz najgorszej możliwej złożoności i przy obu jest w stanie się wybronić.

Sortowanie introspektywne

Tak zwana metoda hybrydowa, to znaczy wykorzystująca połączenie trzech różnych typów sortowania: quick sort, sortowania przez wstawianie oraz kopcowanie. Sortowanie to eliminuje problem najgorszego przypadku algorytmu sortowania szybkiego, zapewniając przy tym logarytmiczno-liniową złożoność obliczeniową. W tym celu tworzona jest stała określająca głębokość wywołań rekurencyjnych uzależniona od ilości elementów sortowanych. Złożoność obliczeniowa wynosi jak w przypadku sortowania przez scalanie O(n log(n)) zarówno dal średniego oraz najgorszego przypadku. Wyniki zawarte zostały poniżej.

	25%	**50%**	**75%**	**95%**	**99%**	**99,70%**	**losowe**	**odwrotne**
10000	0,0874568	0,070289	0,039993	0,009707	0,001972	0,000698	0,002538	0,00182388
50000	2,1960866	1,716025	0,987191	0,220912	0,045455	0,016113	0,013835	0,01005047
100000	2,196087	6,833782	3,96761	0,885667	0,182487	0,055247	0,029345	0,0212487
500000	0,160741	0,106871	0,109265	0,223763	0,653824	1,49176	0,172453	0,12252142
1000000	0,359190	0,324067	0,297489	0,270903	0,265557	0,263872	0,408178	0,25203995



Z moich testów wynika, że spadki efektywności następowały dla tablic o różnych długościach, czasami będąc ciężkie do oszacowania. Może to wynikać ze stosowania dwóch średnio efektywnych algorytmów sortowania (przez wstawianie oraz kopcowanie) lub zaimplementowana została nie najbardziej efektywna wersja samego algorytmu sortowania introspektywnego. Można znaleźć kilka różnych wariantów tego algorytmu, wybrany przeze mnie mógł być nienajlepszy, możliwym wyjściem z problemu byłoby użycie wersji wykorzystującej rekurencje, oraz użycie algorytmu sortowania przez kopcowanie, który posiada osobne funkcje do budowania kopca i zmieniania elementu maksymalnego.

Wnioski

- Niektóre wyniki sortowań zgadzają z wstępnymi założeniami, w przypadku tych niezgadzających się łatwo znaleźć tego przyczynę i zostały one przeze mnie omówione w przypadku analizowania wyników
- Udało mi się zapoznać z różnymi algorytmami sortującymi, zrozumieć ich działanie oraz „odczuć na własnej skórze” ich najlepsze, średnie oraz najgorsze przypadki
- Wszystkie wygenerowane przeze mnie wyniki, można znaleźć w moich repozytorium GitHub, aby zatwierdzić zgodność sprawozdania z funkcjonalnością moich algorytmów