Sprawozdanie z sortowania tablic

Nazwisko, Imię, Indeks	Kajetan Zdanowicz, 248933
Prowadzący kurs	Mgr inż. Marcin Ochman
Termin zajęć	WT 15:15
Data oddania sprawozdania	14.04.2020r.

1 Wstęp

Celem projektu jest porównanie wydajności następujących metod sortowania: przez scalanie, szybkiego oraz introspektywnego. Do testów użyte były generowane losowo tablice o rozmiarach 10000, 50000, 100000, 500000 oraz 1000000. Przeprowadzone testy obejmowały tablice w pełni losowe, posortowane kolejno w $25\%, 50\%, 75\%, 95\%, 99\%, 99,7\%$ oraz posortowane w odwrotnej kolejności. Warto w tym miejscu zaznaczyć, że w przypadku posortowanych tablic, po posortowanej części występują losowe liczby, a nie większe od już posortowanych. Ilość tablic każdego typu wynosi 100.

2 Opis zaimplementowanych algorytmów

2.1 Sortowanie przez scalanie

Polega na rekurencyjnym dzieleniu tablicy, aż do uzyskania pojedynczych elementów. Następnie, algorytm zamienia elementy miejscami (jeśli jest taka potrzeba), scala, a tym samym sortuje tablicę. Złożoność obliczeniowa algorytmu jest stała dla każdego przypadku i wynosi $O(n \cdot logn)$. Złożoność pamięciowa wynosi $O(n)$. Wynika to z konieczności posiadania dodatkowej, tymczasowej struktury przechowującej dane. Jest to algorytm stabilny, co oznacza, że elementy o tej samej wartości w tablicy wejściowej zachowują swoją kolejność w tablicy wyjściowej.

2.2 Sortowanie szybkie

Algorytm ten rozpoczyna się od wybrania piwota. Elementy mniejsze od piwota są przenoszone na lewą stronę tablicy, a wieksze - na prawą. Tablica dzielona jest na dwie części i powtarzana jest analogiczna instrukcja, az do posortowania tablicy. Sortowanie szybkie działa rekurencyjnie. Złożoność obliczeniowa algorytmu zależy w dużej mierze od wybrania piwota. W najgorszym wypadku wynosi $O(n^2)$, w średnim $O(n \cdot logn)$. Złożoność pamięciowa w średnim przypadku wynosi $O(logn)$, w pesymistycznym $O(n)$. Jest to algorytm niestabilny.

2.3 Sortowanie introspektywne

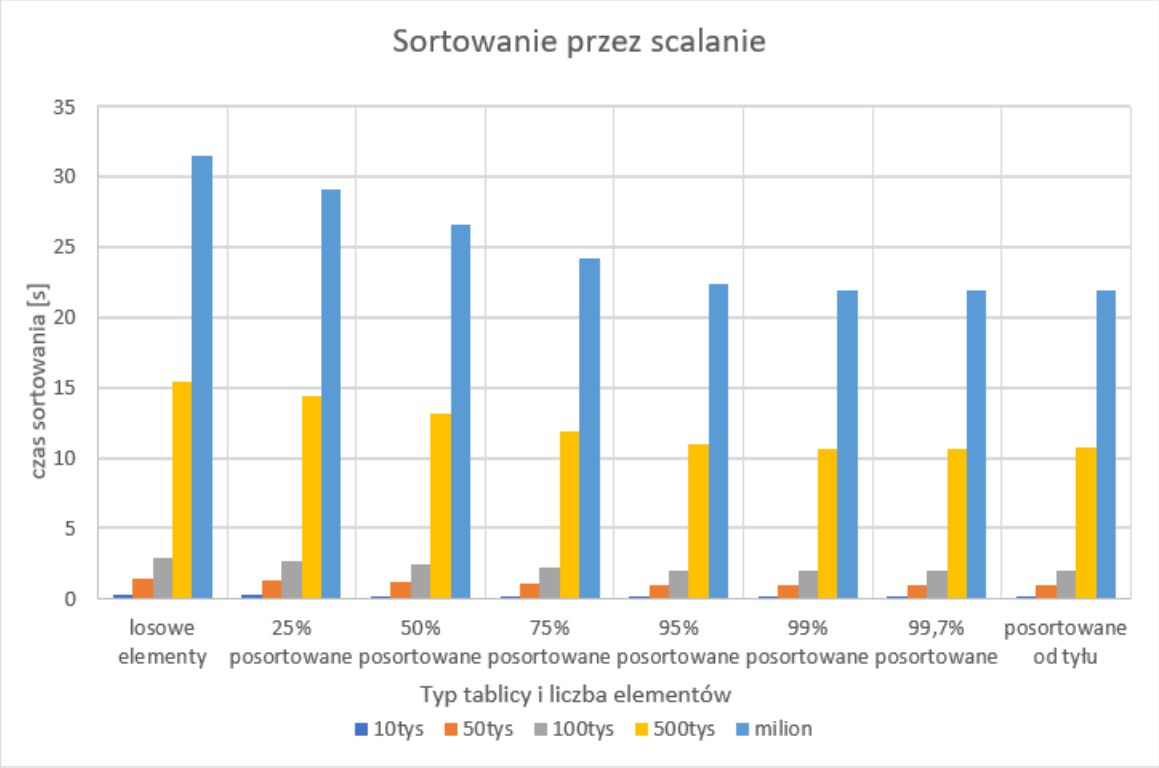
Hybrydowy algorytm wykorzystujący rekurencję. Składają się na niego sortowania przez wstawianie, szybkie i kopcowanie. Jego celem jest usprawnienie sortowania szybkiego, poprzez wykluczenie pesymistycznego przypadku złożoności obliczeniowej $O(n^2)$. Obliczona jest ilość rekurencyjnych wywołań i w możliwym niekorzystnym przypadku, algorytm jest zmieniany z sortowania szybkiego na sortowanie przez kopcowanie. W przypadku tablicy zawierającej 16 elementów lub mniej, korzysta z sortowania przez wstawianie. Z racji niestabilności sortowania szybkiego, sortowanie introspektywne również jest niestabilne. W tym algorytmie wykorzystałem funkcję znajdującą najlepszego piwota spośród trzech liczb, co dodatkowo przyspiesza algorytm. Złożoność obliczeniowa jest stała: $O(n \cdot logn)$, pamięciowa również: $O(logn)$.

3 Wyniki pomiarów

Tabele zawierają zmierzone w sekundach czasy sortowań dla 100 tablic każdego typu. Pomiaru dokonałem wewnątrz programu. Wykresy są graficznymi odpowiednikami tabel.

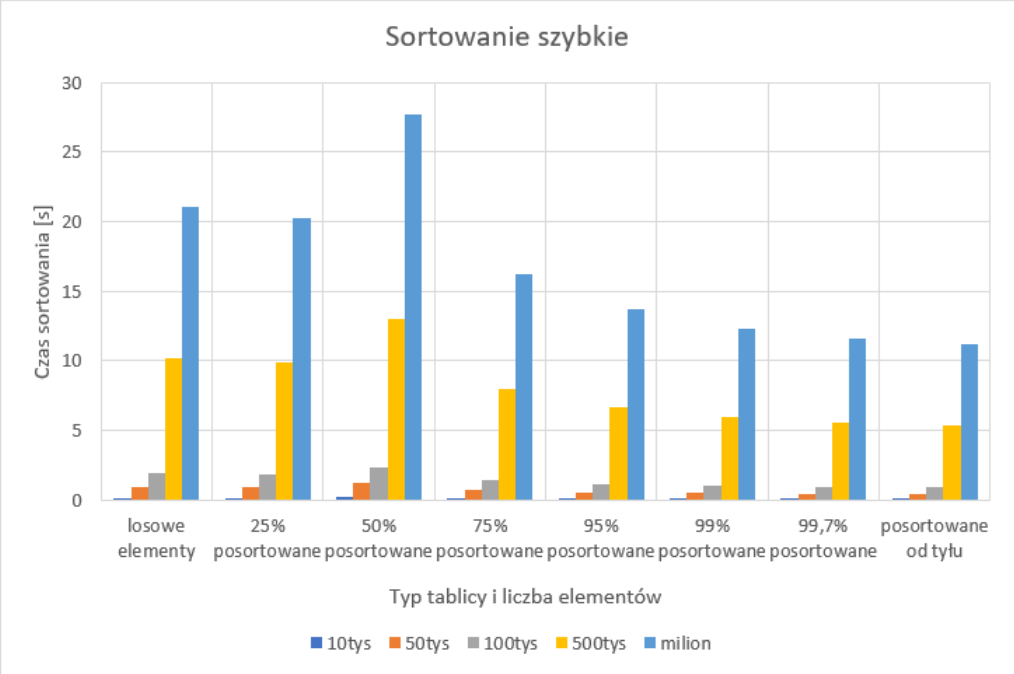
3.1 Sortowanie przez scalanie

rodzaj tablicy	losowe	25% posortowane	50% posortowane	75% posortowane	95% posortowane	99% posortowane	99,7% posortowane	posortowane od tyłu
10tys	0,259951	0,226894	0,211267	0,189059	0,164361	0,161172	0,1608	0,169414
50tys	1,43224	1,27718	1,16389	1,0208	0,929086	0,913454	0,908401	0,927981
100tys	2,87858	2,62283	2,42521	2,18082	1,97606	1,94623	1,94495	1,96489
500tys	15,47	14,3985	13,1354	11,8367	10,9899	10,648	10,6197	10,7816
milion	31,5043	29,0999	26,5382	24,1984	22,3463	21,9542	21,8727	21,9277



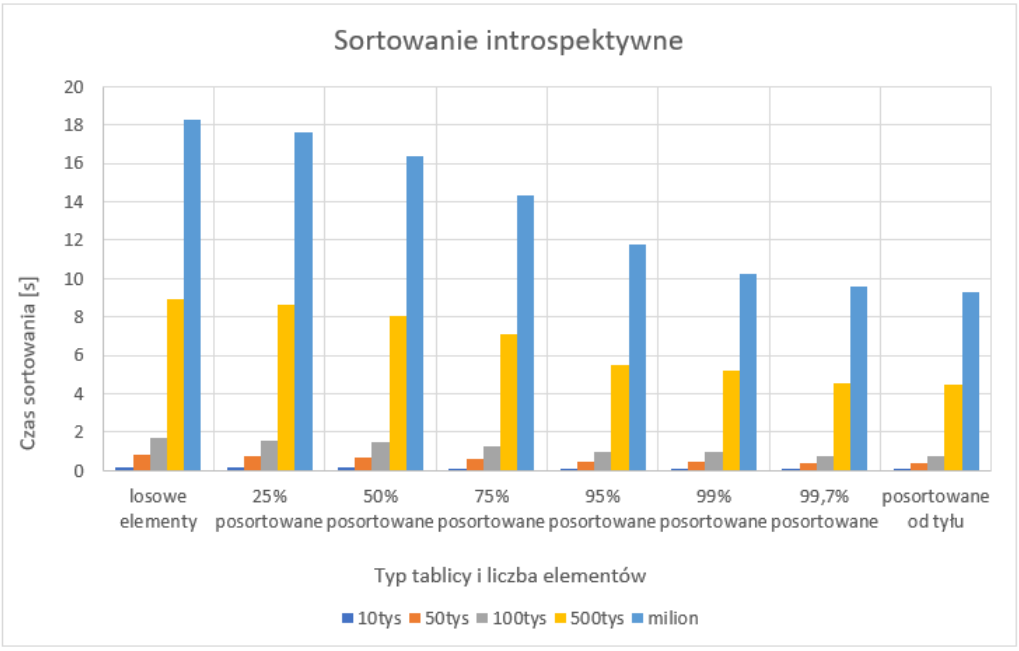
3.2 Sortowanie szybkie

rodzaj tablicy	losowe	25% posortowane	50% posortowane	75% posortowane	95% posortowane	99% posortowane	99,7% posortowane	posortowane od tyłu
10tys	0,177933	0,169048	0,210123	0,132689	0,103729	0,08934	0,082116	0,074978
50tys	0,937215	0,909269	1,24397	0,719476	0,562947	0,491216	0,459762	0,433817
100tys	1,93284	1,83574	2,34791	1,453	1,18711	1,03396	0,957612	0,936606
500tys	10,2358	9,89077	12,9654	7,97521	6,67525	5,93785	5,59146	5,4097
milion	21,0866	20,2354	27,7329	16,2595	13,7061	12,3433	11,6271	11,202



3.3 Sortowanie introspektywne

rodzaj tablicy	losowe	25% posortowane	50% posortowane	75% posortowane	95% posortowane	99% posortowane	99,7% posortowane	posortowane od tyłu
10tys	0,157275	0,146324	0,136222	0,118905	0,087109	0,081119	0,07002	0,060128
50tys	0,781033	0,754801	0,705481	0,616319	0,461222	0,444819	0,38817	0,362662
100tys	1,65845	1,57895	1,47391	1,24508	0,984444	0,935916	0,780082	0,776018
500tys	8,89187	8,60835	8,07103	7,07018	5,46732	5,17172	4,551	4,45756
milion	18,2605	17,5757	16,3385	14,3087	11,7816	10,2266	9,54933	9,28184



4 Wnioski

- Sortowanie przez scalanie czasem może być szybsze niz sortowanie szybkie, jednak jest to rzadko spotykany przypadek.
- Implementacja dynamicznej tablicy std::vector do przechowywania liczb; spowalnia proces sortowania, natomiast ułatwia napisanie programu.
- Sortowanie introspektywne wyklucza najgorszy przypadek sortowania szybkiego z powodzeniem. Odpowiednie ustalenie ilości elementów tablicy przeznaczanej do sortowania przez wstawianie, ustalenie maksymalnej głębokości rekurencji, po której zaczyna działać sortowanie przez kopcowanie oraz inteligentny dobór piwota przez funkcję w algorytmie, sprawiają, że jest on najszybszy i najefektywniejszy spośród testowanych.
- Teoretycznie najwolniejszemu sortowaniu przez scalanie najwięcej czasu zajmuje posortowanie losowej tabeli. Wraz z ilością posortowanych elementów w tablicy wejściowej, rośnie wydajność algorytmu.
- Zdecydowanie największe problemy sortowaniu szybkiemu sprawia tablica posortowana do połowy. Jest to głównie skutek wyboru piwota w środku tablicy.

5 Bibliografia

- `https://pl.wikipedia.org/wiki/Sortowanie_szybkie`
- `https://www.geeksforgeeks.org/know-your-sorting-algorithm`
- `https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie`
- `https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie`