Sprawozdanie

Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Projekt I

Data: 29.03.2021

Wykonał: Mateusz Malik, 249425

Prowadzący: Mgr inż. Marta Emirsajłow

Termin zajęć: Poniedziałek, 13:15 – 15:00

Wprowadzenie

Jednym z zagadnień nierozłącznie związanych z informatyką, a przede wszystkim z magazynowaniem danych jest sortowanie. Wszędzie tam gdzie zachodzi potrzeba przechowywania oraz przetwarzania danych, potrzebne jest ich usystematyzowane uporządkowanie według wybranych parametrów. Takimi parametrami mogą być: wielkość (w przypadku sortowania liczb), litera rozpoczynająca ciąg znaków, rozmiar, kategoria i tp. Dzięki sortowaniu przechowywanych informacji znacznie ułatwiamy późniejszy szybki dostęp do tej części z nich, której akurat potrzebujemy. Dlatego też jest to bardzo ważny zanieg w przypadku kiedy mamy do czynienia z naprawdę dużą ilością danych.

Istnieje wiele różnych algorytmów sortowania, których działanie i szybkość różni się w zależności od wielkości i stanu posortowania danych wejściowych. W celu ich klasyfikacji badane jest asymptotyczne tempo wzrostu czasu działania, lub zapotrzebowania na pamięć, które najczęściej opisuje się stosując notację dużego O. Polega ona na przedstawieniu złożoności obliczeniowej lub pamięciowej za pomocą prostej funkcji zależnej od rozmiaru tablicy wejściowej n, o najbardziej zbliżonym kształcie do funkcji opisującej rzeczywistą zależność. W notacji dużego O wyróżniamy takie klasy funkcji jak:

- $O(1)$ – stała
- $O(\log n)$ – logarytmiczna
- $O(n)$ – liniowa
- $O(n \log n)$ – logarytmiczno - liniowa
- $O(n^2)$ – kwadrarowa
- $O(n^3)$ – sześcienna
- $O(n^k)$ – wielomianowa
- $O(\alpha^n)$ – wykładnicza

W zależności od tego jak duży i uporządkowany jest zbiór danych, który chcemy uporządkować oraz od ilości dostępnej pamięci potrzebnej do wykonania sortowania, decydujemy jaki algorytm sortowania zastosujemy.

Opis badanych algorytmów

Algorytmy jakie wybrałem do przetestowania to Merge Sort, Quick Sort i Intro Sort.

Merge Sort, czyli sortowanie przez scalanie

Jest to algorytm sortowania, który należy do grupy algorytmów szybkich. Jest to zasługą zastosowania metody „dziel i zwyciężaj”, która opiera się na dzieleniu dużego problemu na małe pod-problemy prostsze do rozwiązania. Jest to także algorytm stabilny. Merge sort polega na rekurencyjnym dzieleniu sortowanej tablicy na 2 podtablice, aż do uzyskania tablic 1-elementowych, które są traktowane jako posortowane. Następnie małe tablice są scalane w taki sposób, aby tablica po złączeniu składała się z ich posortowanych

wyrazów. Scalanie odbywa się poprzez porównywanie ze sobą wyrazów z dwóch już posortowanych podtablic idąc od najmniejszych do największych i wstawianiu rezultatów do tablicy pomocniczej, aż do posortowania całości. Na sam koniec wszystkie wyrazy z tablicy pomocniczej są kopiowane do tablicy wejściowej.

Złożoność pamięciowa tego algorytmu to O(n), co jest spowodowane potrzebą wykorzystania do niego tablicy pomocniczej, natomiast złożoność czasowa zarówno dla przypadku średniego, jak i dla najgorszego to O(n log n).

Quick Sort – sortowanie szybkie

Sortowanie to także jest zaliczane do grupy algorytmów szybkich, do czego zobowiązuje jego nazwa. Mimo że również stosuje zasadę dziel i zwyciężaj, to jego sposób działania jest zupełnie inny niż w Merge Sorcie. Tu także dzielimy tablicę na mniejsze części, lecz w tym przypadku sortowanie dokonuje się już na tym etapie. Wybierany jest element osiowy, tzw. pivot, a następnie tablica jest dzielona na dwie podtablice zawierające elementy odpowiednio mniejsze i większe od niego. Pivot jest ustawiany między nimi i w tym momencie zapewnione jest, że jest on już na właściwym miejscu. Podział ten jest wywoływany rekurencyjnie aż do uzyskania podtablic jedno – wymiarowych, w tym momencie cała tablica jest już posortowana. Algorytm ten nie potrzebuje dodatkowej pamięci, zatem jego złożoność pamięciowa to O(1). Jeśli chodzi o złożoność czasową, to dla przypadku średniego jest to złożoność liniowo-logarytmiczna, natomiast dla wariantu niekorzystnego może nawet sięgać $O(n^2)$. To kiedy wystąpi wariant najmniej korzystny, zależy od sposobu wybierania elementu osiowego. Jeśli będzie to zawsze ostatni element tablicy, to złożoność kwadratową Quick Sort osiągnie w przypadku tablicy na wejściu już posortowanej. Jeśli będzie to pierwszy element, to najdłużej zajmie sortowanie tablicy odwrotnie posortowanej. W celu wyeliminowania problemu złożoności kwadratowej stosuje się losowanie pivota, lub liczenie mediany z 3 sąsiednich elementów.

Intro Sort, czyli sortowanie introspektywne

Algorytm ten jest udoskonaleniem sortowania szybkiego przez dodanie mechanizmu zapobiegającego zbyt dużej ilości wywołań rekurencyjnych dla małych podtablic. Polega to na badaniu głębokości rekurencji wywołań metody dzielącej tablice przez ustawianie elementów po dwóch stronach pivota. Jeśli przekroczyła ona liczbę 2 $log_2 n$, rekurencja zostaje zatrzymana i uruchamiane jest sortowanie przez kopcowanie. Na koniec dla całej tablicy uruchamiane jest także sortowanie przez wstawianie, które dla uporządkowanych danych osiąga złożoność O(n). Złożoność pamięciowa algorytmu to O(log n) ze względu na wielokrotne uruchamianie sortowania przez kopcowanie, natomiast czasowa to w każdym przypadku O(n log n).

Przebieg i wyniki eksperymentów

Eksperymenty zostały przeprowadzone za pomocą programu generującego tablice losowe i dokonującego ich sortowania. Testowane algorytmy porządkowały tablice liczb całkowitych o pięciu rozmiarach: 10 000, 50 000, 100 000, 500 000 i 1 000 000 elementów oraz 8 wariantach ułożenia elementów: nieposortowane, posortowane w 25%, posortowane w 50%, (…) 75%, 95%, 99%, 99,7%, czy posortowane w odwrotnej kolejności. Dla każdego algorytmu oraz wariantu, wielkości i posortowania został zmierzony czas porządkowania 100 takich tablic, z których każda była generowana na nowo, dla uzyskania 100-krotności statystycznego średniego wyniku. Aby zwiększyć rzetelność testów, każdy algorytm sortował zestaw tablic z tymi samymi wartościami. Program został skonstruowany tak, aby wykonać wszystkie testy w trakcie jednego uruchomienia, zajmując przy tym możliwie mało pamięci operacyjnej. Mimo to, prawdopodobnie przez ograniczenia sprzętowe, nie był w stanie wykonać się w takim wariancie do końca, zatem aby uzupełnić ostatnie wyniki musiał być on uruchamiany wielokrotnie przy „odcięciu" z kodu fragmentów dotyczących mniejszych tablic (które przeszły pomyślnie pierwszy test) oraz zmodyfikowaniu w celu zminimalizowania użycia RAM-u. Dla utrzymania wiarygodności wyników wszystkie kolejne uruchomienia odbywały się na tym samym sprzęcie, w tych samych warunkach (brak uruchomionych innych programów, ten sam priorytet programu z testami i tp.). Możliwe, że bez względu na optymalizację, ograniczenia sprzętowe i tak miały wpływ na niektóre wyniki, przekłamując ich wartość.

Tabela przedstawia wszystkie uzyskane wyniki z podziałem na wielkość i ułożenie tablicy oraz zastosowany algorytm:

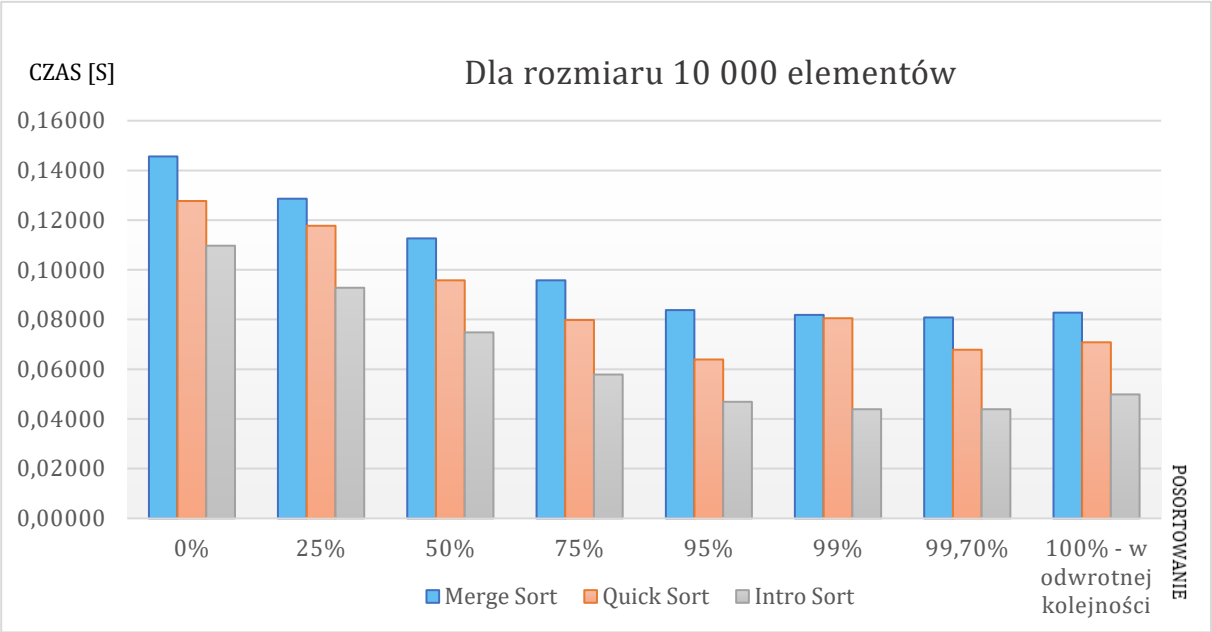
Tabela wyników czasu sortowania

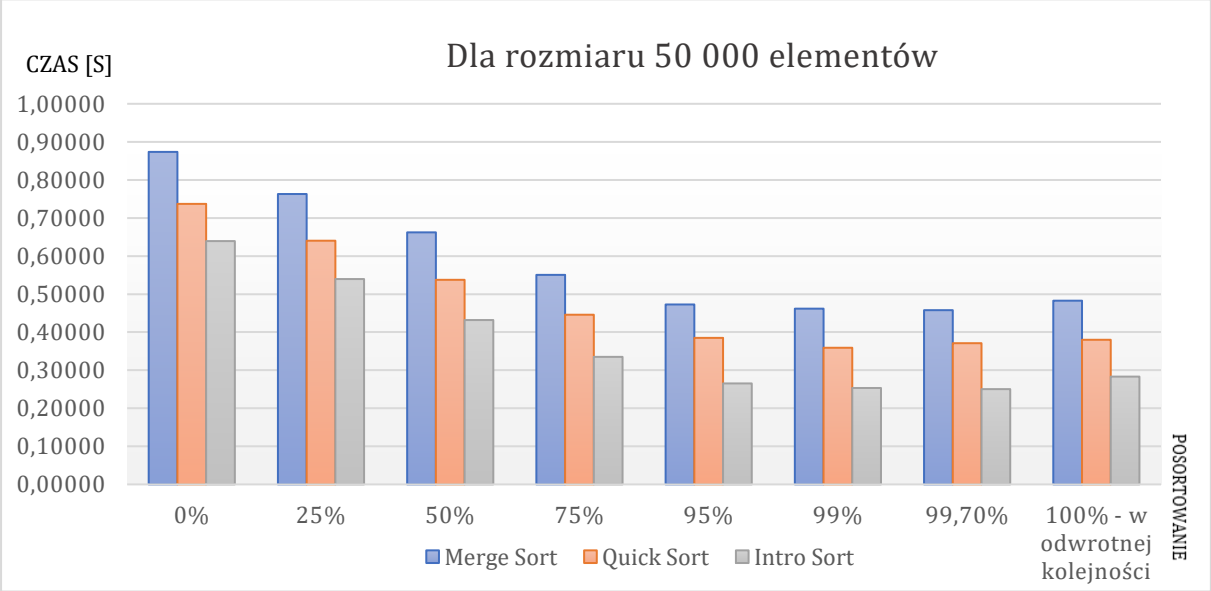
Wyniki przedstawiono w sekundach

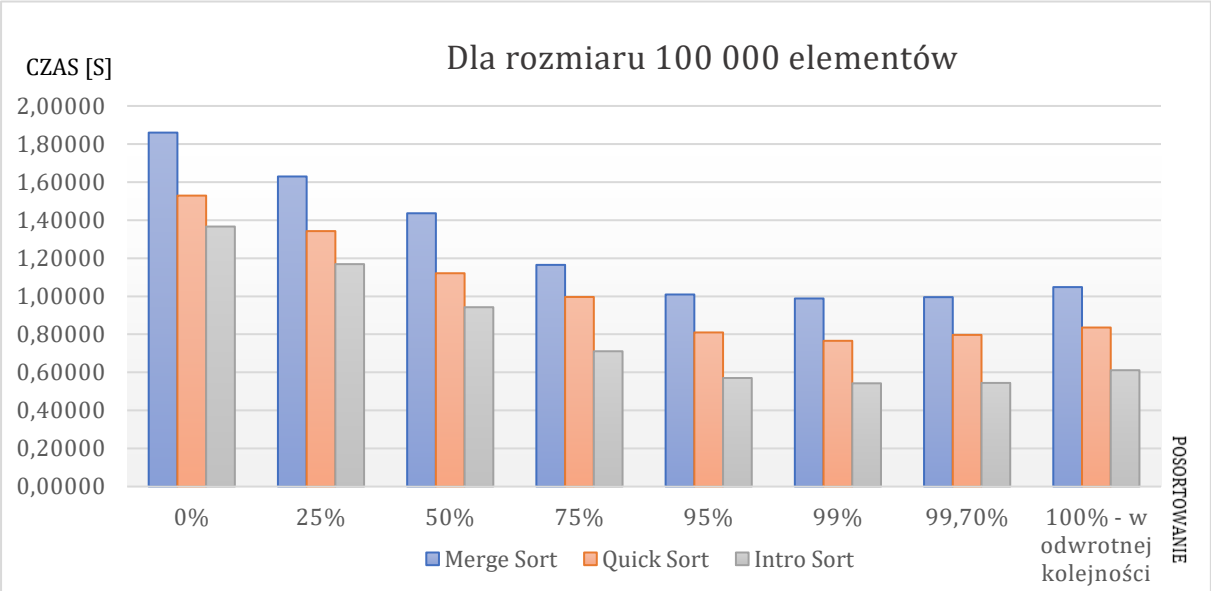
Rozmiar tablicy	Algorytm Sortowania	Wariant uporządkowania tablicy - posortowanie							
		0%	25%	50%	75%	95%	99%	99,70%	100% - w odwrotnej kolejności
10 000									
	Merge Sort	0,14561	0,12865	0,11270	0,09578	0,08381	0,08182	0,08082	0,08281
	Quick Sort	0,12769	0,11772	0,09578	0,07981	0,06393	0,08053	0,06785	0,07084
	Intro Sort	0,10971	0,09279	0,07483	0,05788	0,04691	0,04392	0,04392	0,04988
50 000									
	Merge Sort	0,87370	0,76299	0,66226	0,55056	0,47277	0,46180	0,45781	0,48274
	Quick Sort	0,73706	0,64033	0,53758	0,44584	0,38500	0,35904	0,37104	0,38002
	Intro Sort	0,63933	0,53959	0,43188	0,33514	0,26532	0,25336	0,25036	0,28324
100 000									
	Merge Sort	1,86006	1,62948	1,43620	1,16492	1,00933	0,98839	0,99537	1,04821
	Quick Sort	1,52894	1,34244	1,12100	0,99634	0,80983	0,76594	0,79690	0,83576
	Intro Sort	1,36639	1,16890	0,94251	0,71102	0,57051	0,54258	0,54458	0,61140
500 000									
	Merge Sort	11,12363	9,91429	8,55096	6,93346	5,68273	5,51628	5,76159	6,04878
	Quick Sort	8,94807	8,53714	7,60669	6,73701	5,91212	5,74566	5,62995	6,65381
	Intro Sort	8,02855	6,91776	6,85168	9,40033	6,25913	4,20077	4,04122	6,89373
1 000 000									
	Merge Sort	21,0058	18,3473	15,7852	13,2570	11,3516	11,0163	10,9609	21,0077
	Quick Sort	16,6969	16,8437	16,1712	14,9802	13,7198	13,3384	13,5319	16,8802
	Intro Sort	17,5316	20,2239	1077,57	33819,7	134,661	13,9122	9,3231	31811,7

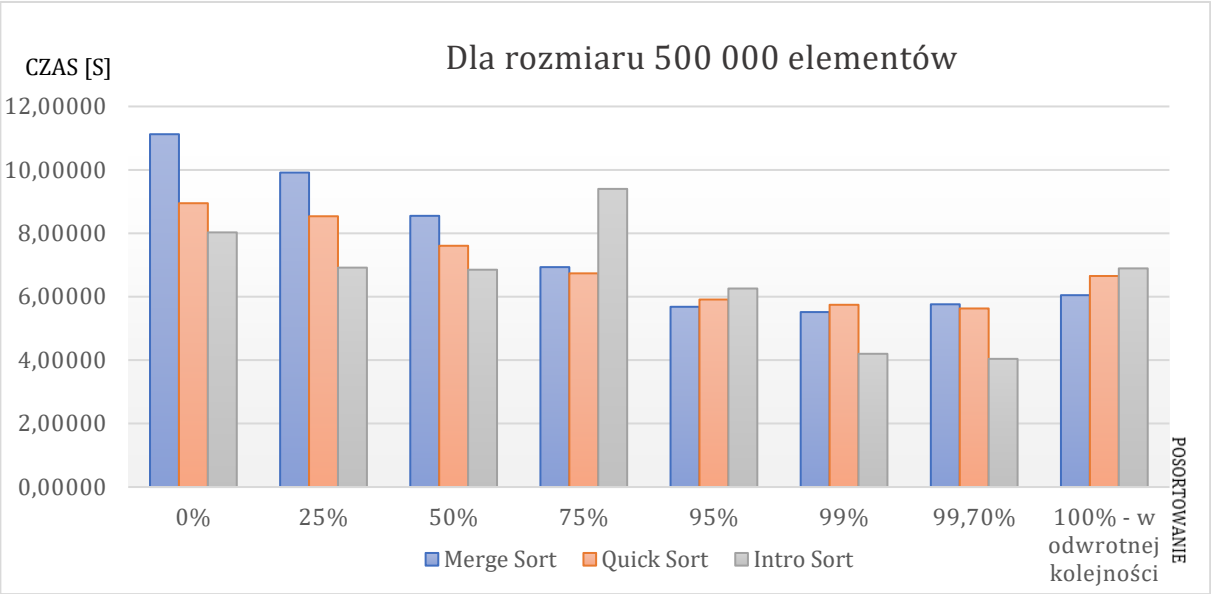
Wyniki odbiegające od normy

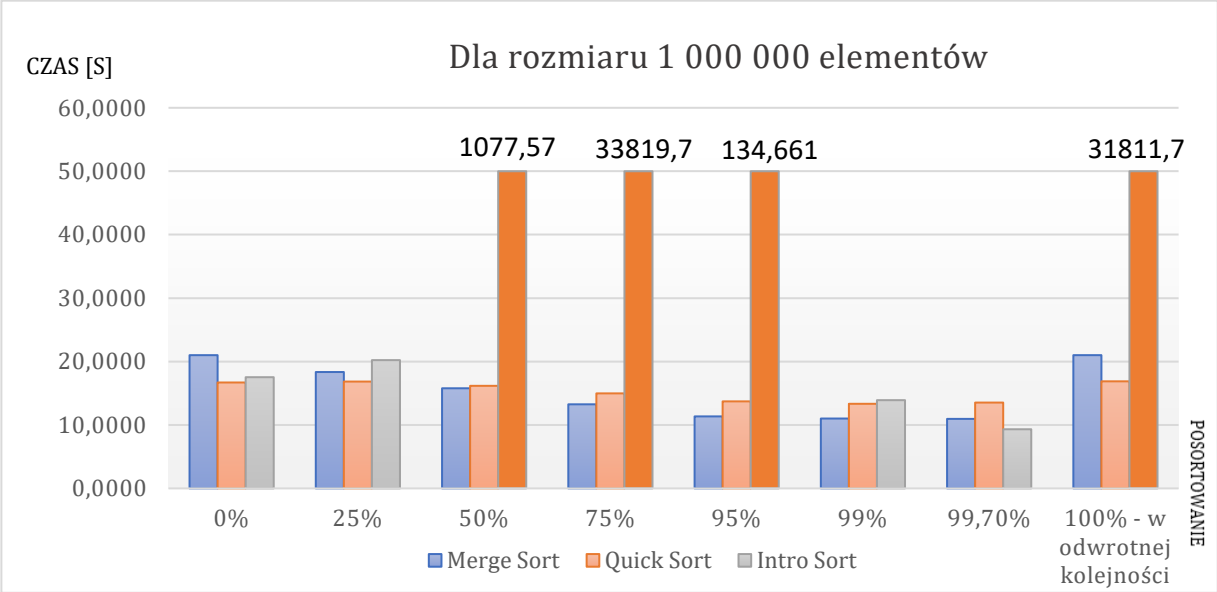
Wykresy zależności czasu sortowania od wariantu uporządkowania tablicy wejściowej:





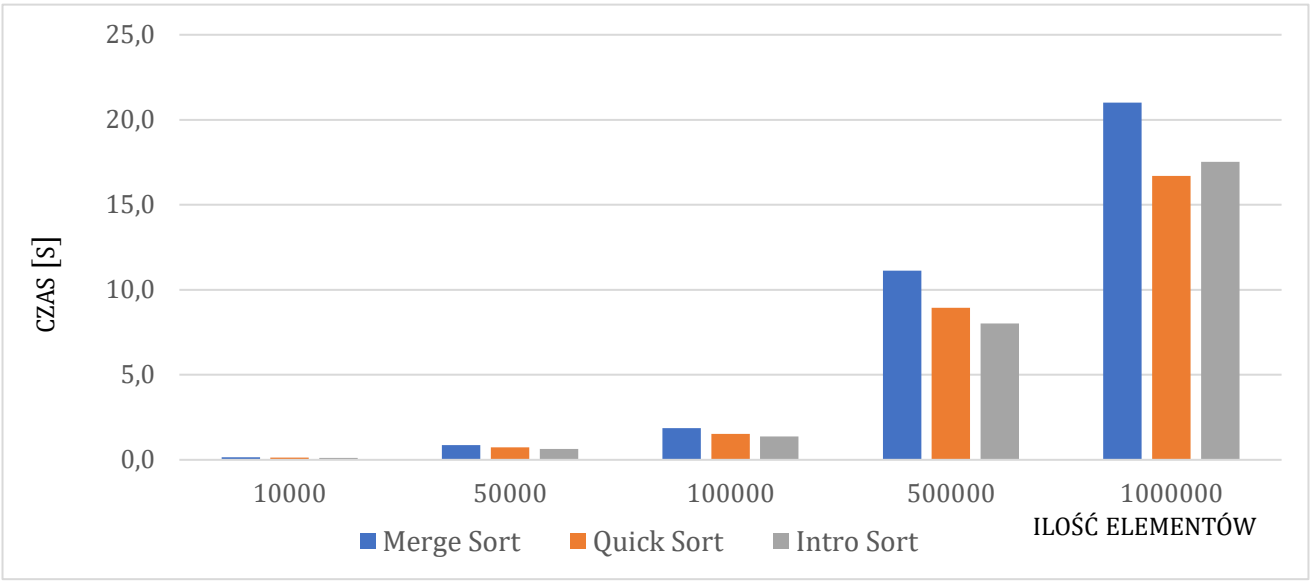


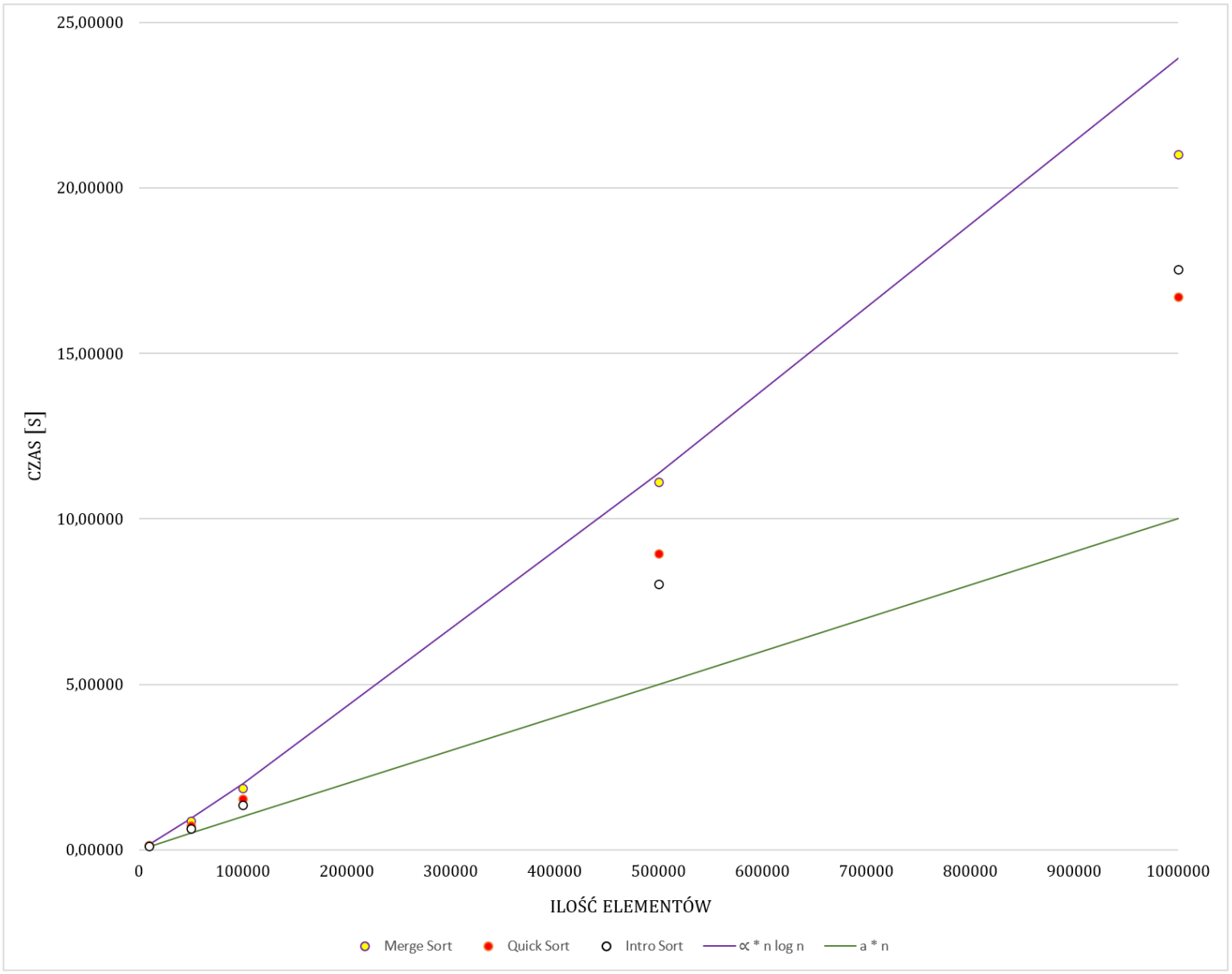




W związku z wykraczaniem wyników na ostatnim wykresie poza skalę, dla zachowania przejrzystości danych skala wyników odbiegających od normy nie jest zachowana, a ich rzeczywiste wartości znajdują się na etykietach słupków. W kolejnych wykresach wyniki te zostaną pominięte.

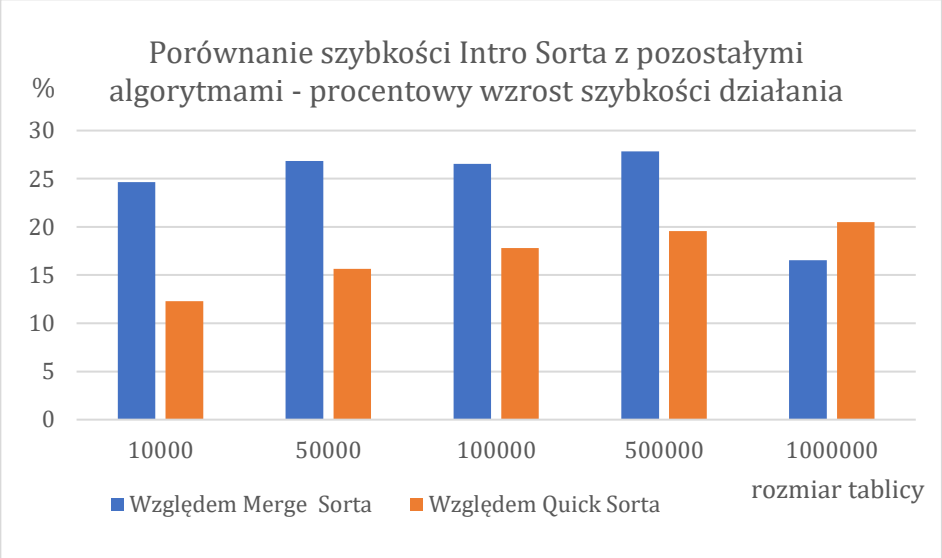
Wykresy zależności czasu sortowania od wielkości tablicy wejściowej dla wariantu nieuporządkowanego:





Podsumowanie, wnioski

Zaczynając od pierwszych wykresów przedstawiających zależności między czasem sortowania, a uporządkowaniem tablicy dla wybranego rozmiaru, łatwo zauważyć tendencję w wynikach uzyskanych przez badane algorytmy. Dla pierwszych trzech rozmiarów, czyli odpowiednio: 10 000, 50 000 i 100 000 elementów, zawsze najszybciej wykonywało się sortowanie introspektywne, następnie sortowanie szybkie, a najwolniej sortowanie przez scalanie, niezależnie od ułożenia tablicy. Dodatkowo różnice te nie pogłębiały się znacząco wraz ze wzrostem rozmiaru tablic, co może już w pewien sposób wskazywać na podobną klasę funkcji złożoności czasowej u każdego algorytmu. Od rozmiaru 500 000 w górę zaczęły występować niewyjaśnione wyjątki od tej reguły, gdzie sortowanie introspektywne nie dość, że wypadało najgorzej, to jeszcze osiągało wyniki kilka rzędów gorsze. Na poniższym wykresie przedstawiono wyniki - ile procent szybciej działał Intro Sort od pozostałych algorytmów, w przypadku tablicy z elementami całkowicie losowymi:



Średnio, sortowanie introspektywne działało 24,48 % szybciej od sortowania przez scalanie i 17,16 % szybciej niż sortowanie szybkie, przy czym widać, że z im większymi tablicami miało do czynienia sortowanie szybkie – tym gorzej wypadało na tle Intro Sorta.

Kolejna kwestia to zależność między wstępnym uporządkowaniem tablic, a czasem sortowania. Poza wykresem dotyczącym rozmiaru 1 000 000, zachowana jest zależność, że im bardziej posortowana tablica na wejściu, tym szybsza praca algorytmu. Między czasem sortowania tablic uporządkowanych w 99%, a tymi w 99,7%, nie było wielkiej różnicy, przy czym w niektórych przypadkach wynik był minimalnie szybszy dla tablicy mniej uporządkowanej. Czas sortowania tablic o odwróconej kolejności elementów, oscylował między wynikiem dla 75, a 95 – procentowego uporządkowania, poza tablicą z milionem elementów, w przypadku której zajmował on nawet więcej niż posortowanie tablicy nieuporządkowanej. Warto zauważyć, że mimo, że dla wielu implementacji sortowania szybkiego jest to najbardziej pesymistyczny scenariusz, w którym algorytm ten osiąga złożoność czasową $O(n^2)$, w tym przypadku nie miało to miejsca. Jest to zasługą udoskonalenia implementacji poprzez losowanie pivota, zamiast wybierania na jego miejsce pierwszego, bądź ostatniego elementu tablicy.

Diametralne odbieganie od normy wyników testów jedno milionowej tablicy, najprawdopodobniej jest spowodowane ograniczeniami sprzętowymi, lub systemowymi maszyny, na której były one uruchamiane. Stopień w jakim nie można ich porównywać do reszty rezultatów, wskazuje, że jest to bardziej wina czynnika zewnętrznego niż regularnego statystycznie spowolnienia algorytmu. Istnieje także możliwość, że w którymś fragmencie programu wystąpił niezidentyfikowany błąd znacznie wydłużający jego wykonywanie, jednak wcześniejsze rezultaty i wstępne przetestowanie działania algorytmów obniżają prawdopodobieństwo takiego przypadku.

Ostatni wykres, na którym widzimy zależność czasu potrzebnego na sortowanie, od wielkości całkowicie losowej tablicy wejściowej, służy do potwierdzenia liniowo – logarytmicznej złożoności czasowej wszystkich trzech algorytmów. Wyniki są graficznie ograniczone dwiema funkcjami: od góry jest to $1{,}2 \cdot 10^{-6} \cdot n \log_2 n$, natomiast od dołu: $n \cdot 10^{-5}$, zatem widać, że złożoność jest większa od liniowej, natomiast ogranicza ją funkcja liniowo – logarytmiczna, co potwierdza wstępne założenie.

Literatura

- http://lukasz.jelen.staff.iiar.pwr.edu.pl/styled-2/page-2/index.php
- http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html
- https://pl.wikipedia.org/wiki/Sortowanie
- https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
- https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie
- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
- https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://www.youtube.com/watch?v=2s717IFZLuU&t=3s
- https://www.youtube.com/watch?v=82XxdhRCMbI&t=88s
- https://www.youtube.com/watch?v=8RkE7MbqVl8&t=29s
- https://www.youtube.com/watch?v=cKJx2nrAmx0
- https://www.youtube.com/watch?v=iJyUFvvdfUg
- https://www.youtube.com/watch?v=M2bKENbdnI4&t=639s