Sprawozdanie z Sortowania Liczb

Michał Bizoń

05 IV 2014

1 Założenia i implementacja metod

Naszym zadaniem była implementacja algorytmów sortowania liczb - zaimplementowane zostały przeze mnie algorytmy: **QuickSort , MergeSort , IntroSort , HeapSort (wymagany do Introsorta)**, a następnie przetestowanie ich pod kątem szybkości oraz przypadków użycia - który jest najbardziej skuteczny przy danym zestawie danych.

1.1 MergeSort - Sortowanie przez Scalanie

Rekurencyjny algorytm sortowania danych, stosujący metodę ***Dziel i zwyciężaj***. Odkrycie algorytmu przypisuje się ***Johnowi von Neumannowi***. Czas tego sortowania wynosi ***O(n*log[n])***.

Algorytm ten:

- Dzieli dane na dwie równe części
- Sortuje każdą część osobno
- Scala posortowane elementy w jeden ciąg

1.2 QuickSort - Sortowanie Szybkie

Jeden z popularnych algorytmów sortowania działających na zasadzie ***Dziel i zwyciężaj***. Sortowanie Szybkie zostało wynalezione w 1962 przez C.A.R. Hoare'a. Algorytm sortowania szybkiego jest wydajny: jego średnia złożoność obliczeniowa jest rzędu ***O(n*log[n])***. Ze względu na szybkość i prostotę implementacji jest powszechnie używany.

Algorytm ten:

- Wybiera z tablicy element rozdzielający (tzw. **Piwot**) i przenosi elementy mniejsze od niego na lewo, a większe na prawo
- Po zakończonym przenoszeniu algorytm wywołuje się rekurencyjnie dla lewego podzbioru dopóki wszystkie dane w lewym podzbiorze nie zostaną posortowane. To samo dzieje się z prawym podzbiorem

Piwot jest zawsze wybierany na początku wywołania funkcji.

1.3 IntroSort - Sortowanie Introspektywne

Sortowanie to jest z rodziny hybrydowych, czyli do sortowania wykorzystuje inne algorytmy, które razem działają naprawdę szybko. Wykorzystuje m.in. sortowanie szybkie, sortowanie przez kopcowanie oraz sortowanie przez wstawianie. Na początku Algorytm dzieli tablicę na mniejsze części (wielkość ustalona odgórnie, np. 64 elementy) za pomocą procedury podziału znaną z sortowania szybkiego. Kiedy cała tablica zostanie wstępnie posortowana, każda z nich jest sortowana poprzez kopcowanie. Całość dopełnia i zamyka sortowanie przez wstawianie, które działa bardzo dobrze dla prawie posortowanych danych.

Algorytm ten:

- dzieli tablicę na mniejsze części (wielkość ustalona odgórnie, np. 64 elementy) za pomocą procedury podziału znaną z sortowania szybkiego
- Kiedy cała tablica zostanie wstępnie posortowana, każda z nich jest sortowana poprzez kopcowanie
- Całość dopełnia i zamyka sortowanie przez wstawianie, które działa bardzo dobrze dla prawie posortowanych danych.

2 Sposób Implementacji

Napisany przeze mnie program operuje na **2 tablicach** o wymiarach **[100][n]**, gdzie **n** jest aktualnie wybraną przez użytkownika wielkością. Przy każdym starcie symulacji tworzona jest **TABELA**, do której przekazywane są dane parametry:

- ilość elementów w każdej tablicy (rozmiar **n**)
- ilość wstępnie posortowanych elementów
- zakres losowanych liczb
- parametr „**i**” (od 0 do 100) – numer wiersza tablicy – wypełniając tablicę funkcją RAND() zdarzyło się, że generował dwa wiersze o identycznych wartościach, dodanie „**i**” do SRAND rozwiązało ten problem

Drugą tabelą jest **TABELA_SORTOWAN**, wypełniana początkowo wartościami *NULL* o takich samych rozmiarach jak **TABELA**. Przed każdym sortowaniem następuje przypisanie wartości z **TABELA** do **TABELA_SORTOWAN** (która jest poddawana sortowaniu, dzięki czemu przy każdym sortowaniu operujemy tym samym zestawem danych. Czasy są zapisywane do osobnej tablicy, a następnie zapisywane są do plików ***.csv** - ich nazwa jest uzależniona od wybranego algorytmu oraz opcji wybranych przed rozpoczęciem symulacji. Po każdym sortowaniu sprawdzana jest kolejność elementów – dla całej tablicy sprawdza czy element n <= n+1.

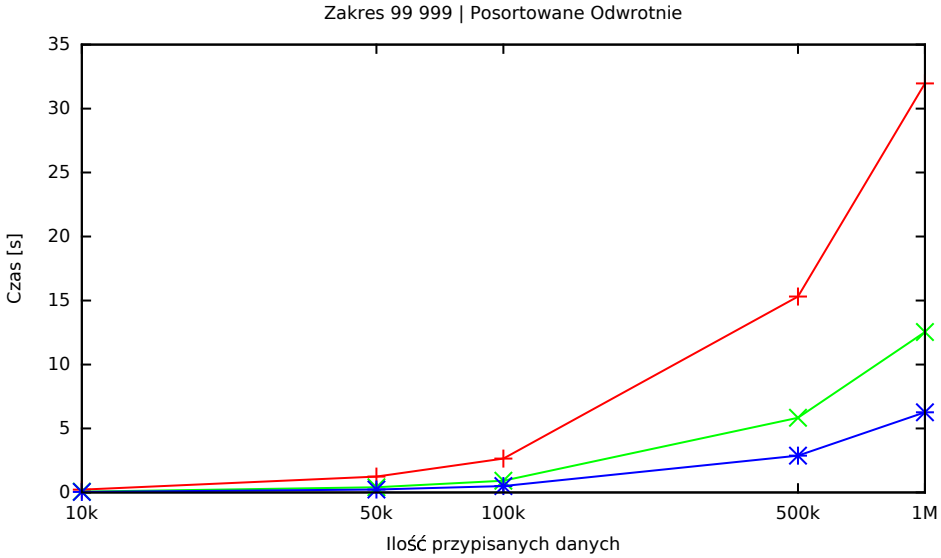
3 Wyniki oraz wykresy

Wykresy porównujące ogólne czasy przypisań tych algorytmów - dla zestawów danych kolejno:

[10 000 ; 50 000 ; 100 000 ; 500 000 ; 1 000 000]
a także zakresu losowanych liczb - [0 ; 99 999]
MergeSort ; QuickSort ; IntroSort

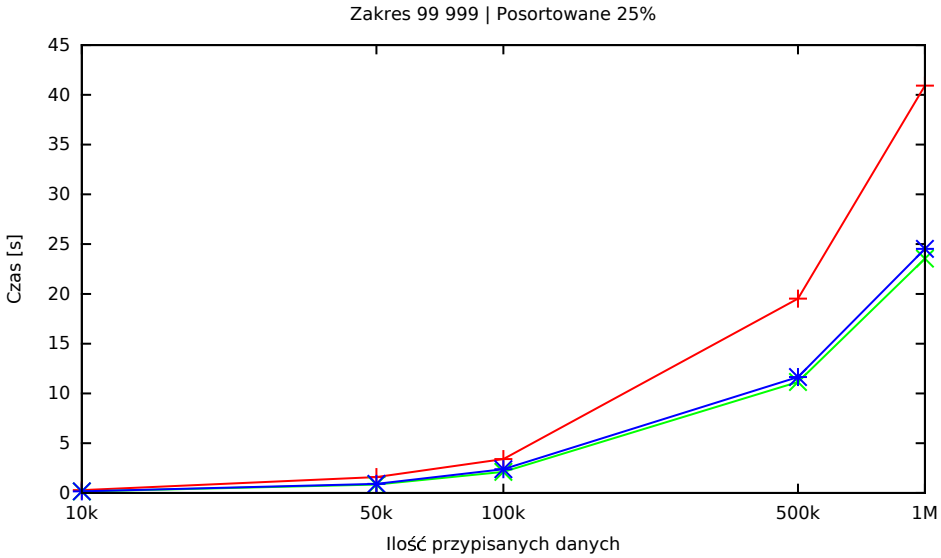
3.1 Posortowane Odwrotnie

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.217866	0.066815	0.041242
50 000	1.24108	0.405296	0.232374
100 000	2.65195	0.915541	0.498457
500 000	15.3149	5.83867	2.87433
1 000 000	31.9639	12.5349	6.25958



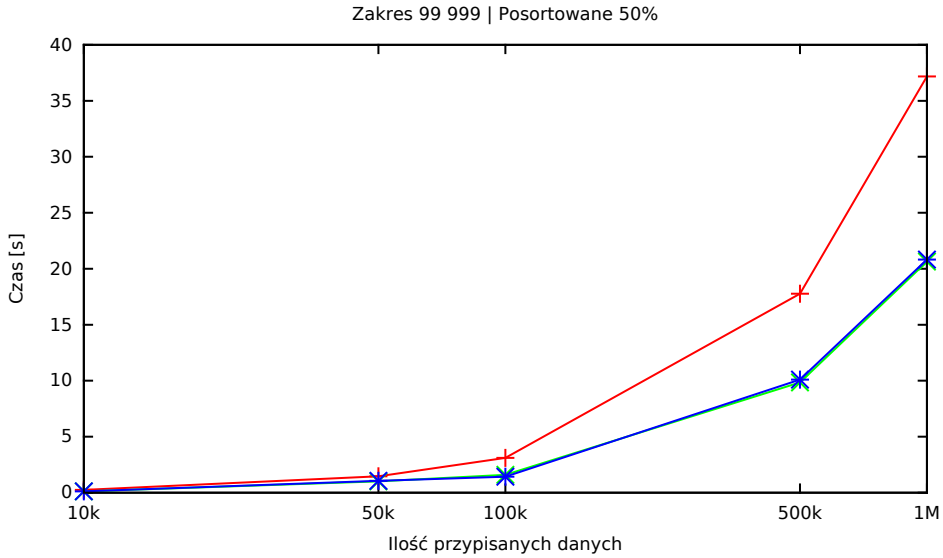
3.2 Posortowane 25% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.279653	0.145696	0.161251
50 000	1.59893	0.846128	0.905473
100 000	3.40157	2.11195	2.38316
500 000	19.5257	11.1548	11.6455
1 000 000	40.9295	23.551	24.5284



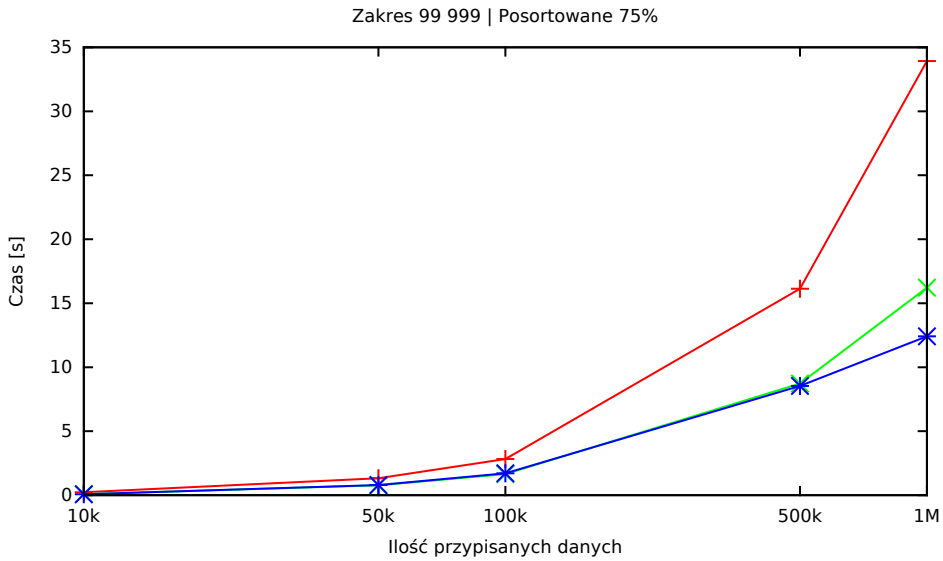
3.3 Posortowane 50% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.254603	0.115298	0.116302
50 000	1.47066	1.00844	1.06264
100 000	3.10443	1.59201	1.43375
500 000	17.7761	9.85518	10.1041
1 000 000	37.1798	20.64	20.8247



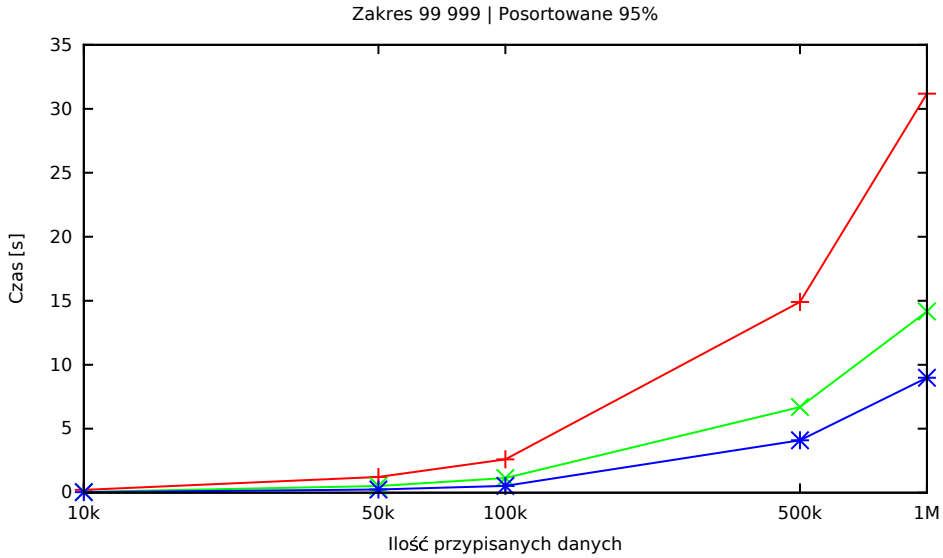
3.4 Posortowane 75% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.231321	0.086716	0.073935
50 000	1.32414	0.772884	0.794843
100 000	2.82	1.64857	1.71199
500 000	16.1368	8.72012	8.53956
1 000 000	33.9285	16.2207	12.4165



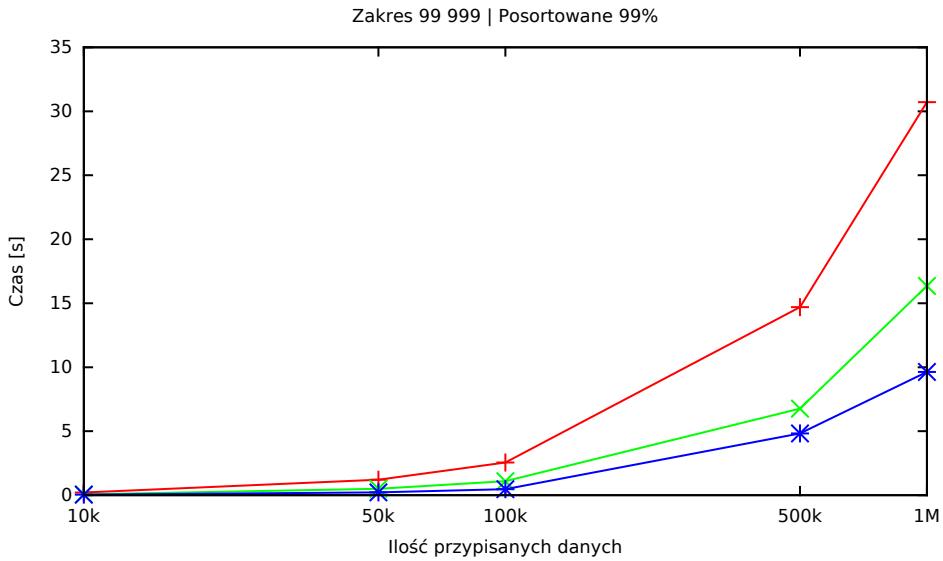
3.5 Posortowane 95% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.214803	0.066131	0.042678
50 000	1.22505	0.52572	0.246116
100 000	2.60647	1.14297	0.535877
500 000	14.9096	6.69184	4.09729
1 000 000	14.9096	14.1687	8.98775



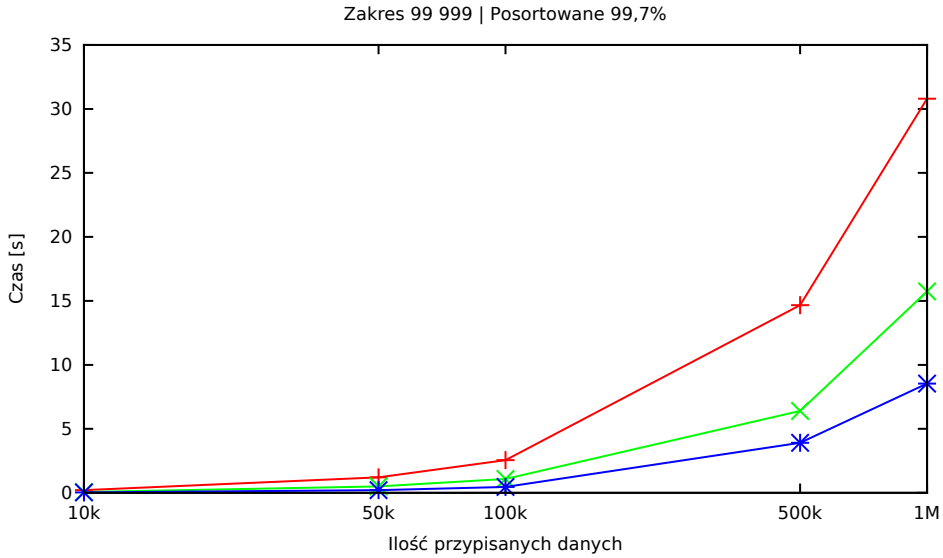
3.6 Posortowane 99% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.211942	0.062384	0.037232
50 000	1.20931	0.500322	0.215102
100 000	2.55391	1.08974	0.472521
500 000	14.6977	6.75924	4.8308
1 000 000	30.7118	16.355	9.62904



3.7 Posortowane 99,7% elementów

Il. Elementów	MergeSort	QuickSort	IntroSort
10 000	0.212263	0.062026	0.035904
50 000	1.20672	0.496701	0.210869
100 000	2.54969	1.08056	0.458917
500 000	14.6631	6.39672	3.90337
1 000 000	30.8038	15.7354	8.53766



4 Wnioski

1. Sortowania QuickSort oraz IntroSort działają prawie dwukrotnie szybciej niż MergeSort
2. Decydując się na wybór QuickSorta nie opłaca się wybierać **Piwotu** (El. Odniesienia) funkcją napisaną dla introsorta (NAJLEPSZY_PIWOT), ponieważ w każdym przypadku czas operacji sortowania się wydłuża (nieznacznie)
3. Za uniwersalny można przyjąć algorytm Quicksort, działa bardzo szybko dla każdego rodzaju i zakresu danych
4. Z moich doświadczeń wynikło iż zakres losowań nie wpływa znacząco na szybkość algorytmów
5. Wyniki zaprezentowane w sprawozdaniu zostały przeprowadzne na Systemie operacyjnym **UBUNTU**, program został zbudowany przy pomocy kompilatora **GCC**, wykorzystując 1 rdzeń procesora
6. Testy zostały również przeprowadzone w systemie **Windows 7 x64** - kompilowany za pomocą **Visual C++ 2010** z dodatkową komendą **/MP - MultiProcessor** - w przypadku użycia dwóch rdzeni IntroSort działa bardzo szybko - czas sortowania tablicy **1 000 000 elementów** oraz posortowaną w **99%** wynosił niecałe 2 [s] (w przypadku **25%** czas wynosił średnio **10,8 [s]**)

Testy zostały wykonane na komputerze o parametrach:
Windows 7 x64 / Ubuntu 13.10, 4GB RAM, Intel Celeron(R) Dual-Core CPU T3500 @ 2.10GHz