

3.04.2019

SPRAWOZDANIE  
PAMSI  
PROJEKT 1  
ALGORYTMY SORTOWANIA

**Prowadzący:**

Dr Łukasz Jeleń

**Dane studenta:**

Kacper Starościak 241581

**Termin zajęć:**

Środa 11:15

## 1. Zadanie

Celem zadania było zaimplementowanie wybranych algorytmów sortowania. Wybrano następujące algorytmy:

- Merge Sort;
- Quick Sort;
- Intro Sort;

Oprócz implementacji przeprowadzono również badania algorytmów, czasu ich działania dla różnych rozmiarów tabel i różnych stopni ich posortowania.

## 2. Opis algorytmów

### a) Merge Sort (sortowanie przez scalanie)

Sposób działania algorytmu sortowania przez scalanie wykorzystuje zasadę „dziel i zwyciężaj”. Dzieli on problem na mniejsze podproblemy i wywołuje się rekurencyjnie również dla nich. Rekurencja trwa, aż podzielone części tablicy będą miały rozmiar 2. Wtedy zaczynamy proces scalania (Merge). Scalanie odbywa się na dwóch mniejszych, ale już posortowanych ciągach i tworzy jeden większy, również posortowany.

Dzielenie tablicy o długości  $n$  na połowy może odbyć się  $\log_2 n$  razy. Scalanie na każdym poziomie zajmuje  $O(n)$  operacji. Wynika z tego, że całkowita złożoność obliczeniowa algorytmu Merge Sort to  $O(n \log n)$ .

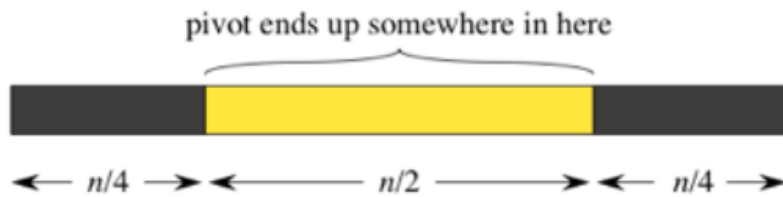
Sortowanie przez scalanie posiada taką samą złożoność obliczeniową zarówno dla najlepszego, średniego jak i najgorszego przypadku.

### b) Quick Sort (sortowanie szybkie)

Sortowanie szybkie polega na wyborze elementu rozdzielającego, według którego uporządkujemy pozostałe elementy. Wszystkie na lewo od pivotu będą od niego mniejsze, a te na prawo-większe. Następnie korzystamy z rekurencji, wywołując sortowanie szybkie dla lewej i prawej strony, pivotu zostawiając na swoim miejscu.

W przypadku najgorszym element rozdzielający znajduje się na początku lub końcu tablicy, przez co jedna z partycji ma wielkość 0, a druga jest tylko o jeden mniejsza od wyjściowej tablicy. Suma czasu potrzebnego na wykonanie tych operacji to  $c(n + (n-1) + (n-2) + (n-3) + \dots + 1)$ . A zatem wykorzystując wzór na sumę ciągu arytmetycznego otrzymujemy  $(n + 1) * \frac{n}{2}$ . Złożoność obliczeniowa jest zatem równa  $O(n^2)$ .

W przypadku średnim mamy do czynienia z czymś pomiędzy najlepszym (czyli podziałem na połowy), a najgorszym przypadkiem. Gdyby pivot lądował zawsze w takim obszarze:



To znaczy, że w najgorszym przypadku podział nastąpiłby na partycje o  $n/4$  i  $3n/4$  elementach. Aby obliczyć ile wywołań rekurencji wystąpi dla dłuższej partycji należy przeanalizować problem następująco: idąc od dołu (gdzie rozmiar to 1) każdy wyższy poziom będzie miał rozmiar  $4/3$  raza większy (bo niższy poziom to  $3/4$  wyższego); zatem ile razy należy powiększać  $4/3$  raza wartość 1 aby otrzymać  $n$ ? Lub inaczej:

$(\frac{4}{3})^x = n$ ; zatem  $x = \log_{\frac{4}{3}} n$ . Analogiczne rozumowanie można przeprowadzić dla krótszej partycji, jednakże nie jest to niezbędne.

Ponadto:

$$\log_{\frac{4}{3}} n = \frac{\log_2 n}{\log_2 \frac{4}{3}}$$

Na każdym z poziomów następuje  $n$  operacji, zatem złożoność obliczeniowa wynosi  $O(n \log n)$ .

#### c) Intro Sort (sortowanie introspektywne)

Jest to algorytm hybrydowy, połączenie dwóch lub trzech algorytmów. Przedstawione później wyniki są wynikami działania algorytmu złożonego z trzech innych: QuickSorta, HeapSorta oraz InsertSorta.

Celem hybrydyzacji algorytmu jest wykluczenie najgorszego przypadku dla algorytmu sortowania szybkiego. Wprowadza się licznik, który liczy głębokość wywołań rekurencyjnych algorytmu. Jeśli głębokość jest zbyt duża, zamiast kolejnego QuickSorta wykorzystuje się HeapSorta (sortowanie przez kopcowanie). Ponadto dla partycji o wielkości mniejszej niż 16 wykorzystuje się sortowanie przez wstawianie (InsertSort).

Złożoność obliczeniowa algorytmu zarówno w najgorszym jak i średnim przypadku wynosi  $O(n \log n)$ .

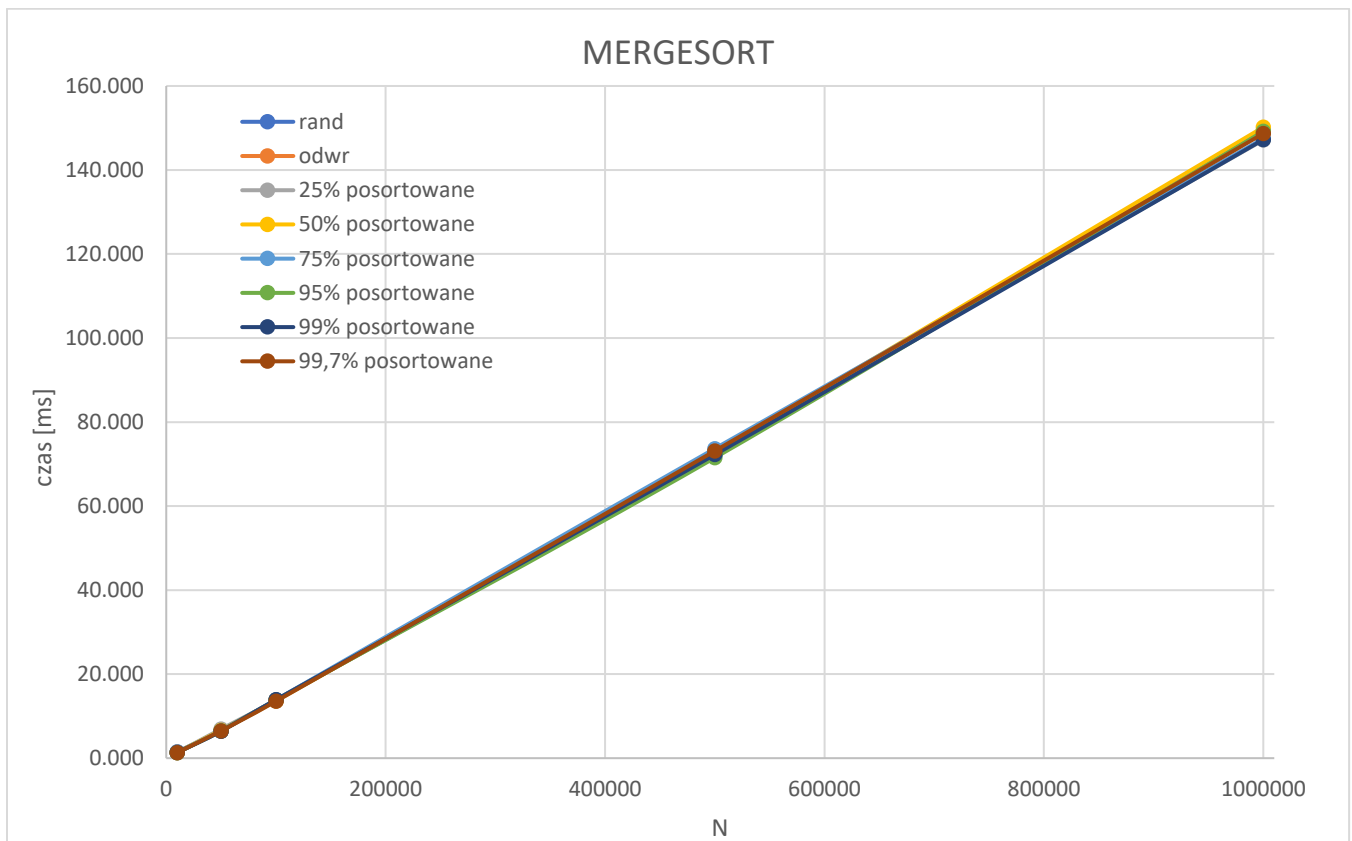
### 3. Przebieg eksperymentu

Do implementacji algorytmów w języku C++ wykorzystano program Microsoft Visual Studio 2017. Zgodnie z instrukcją, pomiary czasu działania algorytmów wykonano dla różnych typów tabel oraz różnych ich rozmiarów. Poniżej przedstawiono tabele wyników (czas podano w milisekundach) oraz wykresy sporządzone na ich podstawie.

#### a) MergeSort

Tabela 1. Wyniki pomiaru czasu działania algorytmu sortowania przez scalanie

n\rodzaj tabeli	random	odwr	25%	50%	75%	95%	99%	99,7%
10000	1.516	1.331	1.332	1.390	1.338	1.322	1.338	1.327
50000	6.553	6.632	6.944	6.646	6.500	6.490	6.424	6.503
100000	13.776	13.835	13.848	13.627	13.888	13.843	13.902	13.556
500000	72.029	72.866	72.988	72.901	73.679	71.506	72.253	73.116
1000000	149.300	148.930	147.123	150.187	147.470	149.178	147.232	148.675

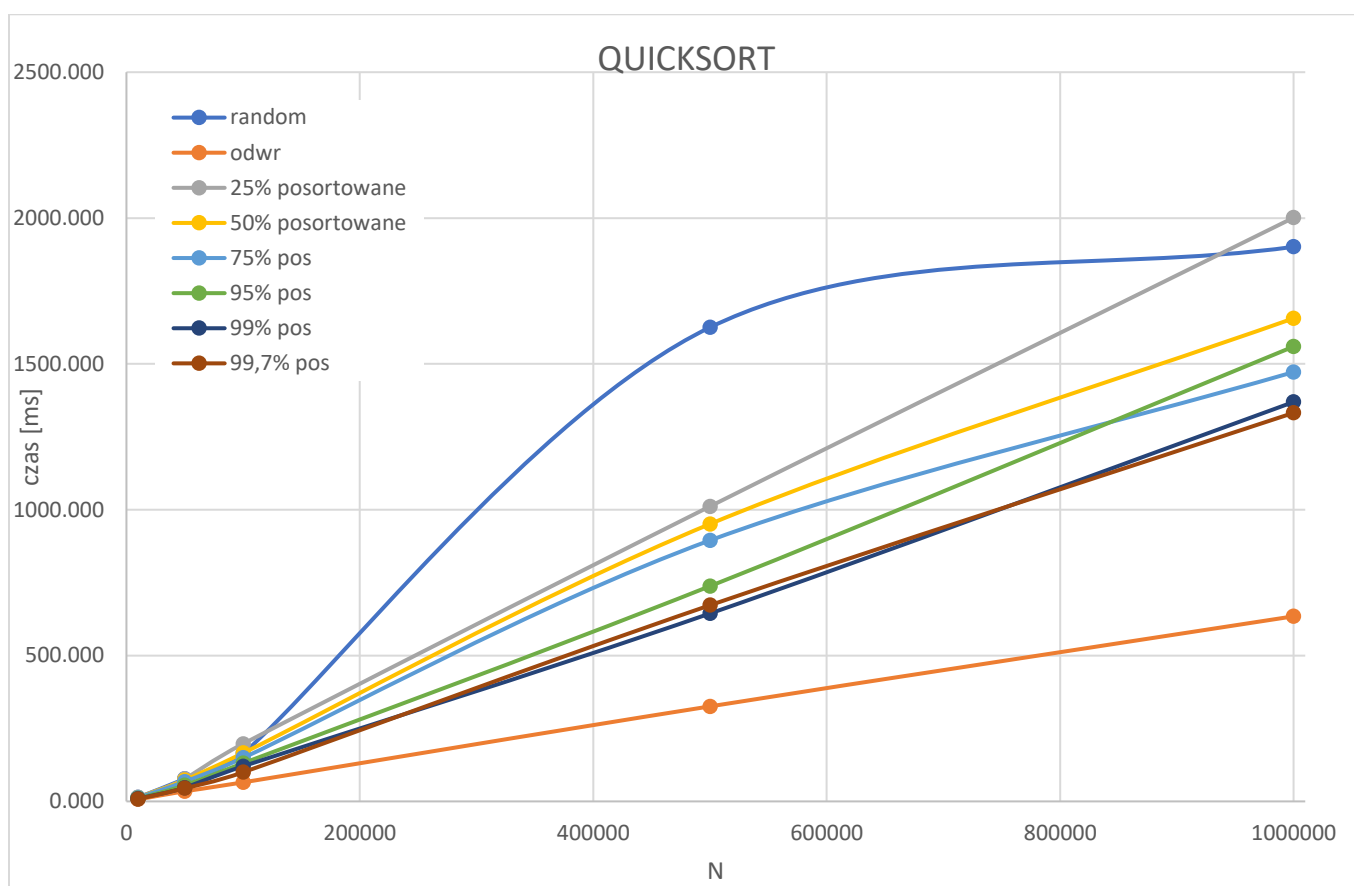


Wykres 1. Wyniki pomiarów czasu dla sortowania przez scalanie

## b) QuickSort

Tabela 2. Wyniki pomiaru czasu działania algorytmu sortowania szybkiego

n\rodzaj tabeli	random	odwr	25%	50%	75%	95%	99%	99,7%
10000	13.519	6.685	13.206	12.795	12.280	9.931	9.255	8.071
50000	77.211	34.786	76.944	72.843	67.917	54.874	46.870	45.114
100000	164.122	65.193	197.413	165.756	149.261	130.941	120.537	99.894
500000	1625.185	325.718	1011.421	950.989	894.883	737.925	644.229	672.524
1000000	1901.367	634.480	2001.135	1656.111	1471.820	1559.320	1369.217	1331.928



Wykres 2. Wyniki pomiarów czasu dla sortowania szybkiego

c) IntroSort

Tabela 3. Wyniki pomiaru czasu działania algorytmu sortowania introspektywnego

n\rodzaj tabeli	random	odwr	25%	50%	75%	95%	99%	99,7%
10000	55.65681	121.6304	53.93675	41.94579	25.27428	5.57878	1.17629	0.38419
50000	1436.099	2818.669	1322.945	1100.453	627.8276	139.5665	29.1201	8.85135
100000	5653.4	11150.71	5189.55	4148.37	2423.98	540.9	110.3	32.13
500000								
1000000								

#### 4. Wnioski

Zastanawiające jest, że algorytmy, które teoretycznie powinny działać szybciej (QuickSort, IntroSort) działają wolniej niż MergeSort. Może być to spowodowane błędami w implementacji lub niewydajnością komputera.

Można zauważyć, że sortowanie przez scalanie jest odporne na rodzaj tablicy, dla każdego z rodzajów wykres wygląda niemal identycznie.