

POLITECHNIKA WROCŁAWSKA

PROJEKTOWANIE ALGORYTMÓW I METODY  
SZTUCZNEJ INTELIGENCJI

PROJEKT  
TERMIN: ŚR. 11:15

---

# Projekt 1:

## Algorytmy sortowania

---

*Autor:*  
Michał KUZEMCZAK

*Prowadzący:*  
dr inż. Łukasz JELEŃ

3 kwietnia 2019



Politechnika  
Wrocławska

# 1 Wprowadzenie

Sprawozdanie zawiera omówienie trzech algorytmów sortowania: przez scalanie, quicksort, introspektywne. Przeprowadzono na nich szereg testów mających na celu sprawdzenie ich wydajności, także ich mocnych i słabych stron dla różnego rodzaju danych wejściowych.

Każdy czas jest podany jako średnia ze 100 sortowań, w milisekundach. Istnieją dwa warianty wstępnego posortowania elementów:

- Posortowane elementy są na swoich miejscach - raczej bezużyteczny sposób, niewiele wnosi do testów. Wstępne posortowanie zawsze skraca czas.
- Posortowane elementy nie są na swoich miejscach - przy tym sposobie zachowanie algorytmów nie jest jednoznaczne, co podczas testów jest bardzo na rękę.

Oczywiście wybrano drugi wariant.

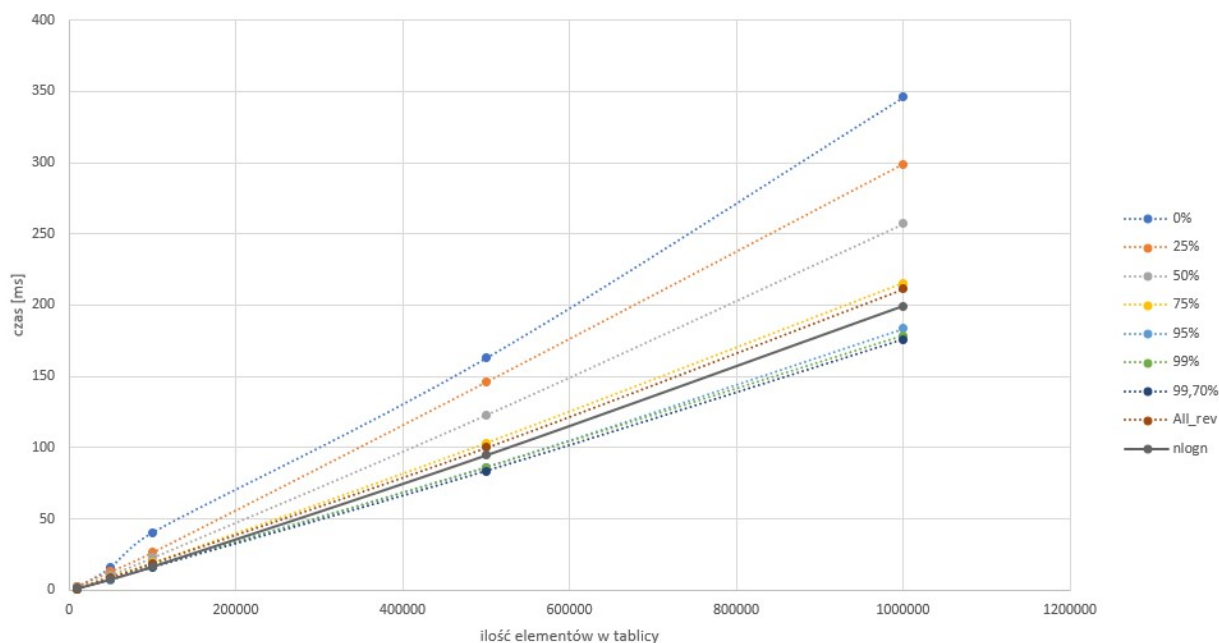
## 2 Sortowanie przez scalanie (merge sort)

Sortowanie przez scalanie jest algorytmem rekurencyjnym. W toku jego działania tablica o wielkości  $n$  zostaje podzielona na  $n$  jednoelementowych podzbiorów, siłą rzeczy posortowanych. W następnym kroku podzbiory są łączone w co raz większe, z zachowaniem sortowania. Maksymalna głębokość rekurencji jest równa wysokości drzewa binarnego, ponadto w każdym elemencie wykonywane jest porównanie i scalenie dwóch podzbiorów, ztem złożoność obliczeniowa algorytmu wynosi  $O(n \cdot \log n)$ . Złożoność ta jest stała, ponieważ dla różnych przypadków zmienia się jedynie liczba porównań podczas scalania.

W **Tabeli 1** przedstawiono wyniki pomiaru czasu sortowania tablic o różnej wielkości i różnym procentowym poziomie wcześniejszego posortowania.

	10000	50000	100000	500000	1000000
0%	2,08	14,25	30,99	171,26	350,48
25%	2,05	12,39	27,37	145,24	301,07
50%	1,06	11,3	22,48	123,74	252,7
75%	1	9,52	19,57	101,21	214,21
95%	1,05	7,73	16,78	85,6	184,52
99%	1,03	7,26	16,71	89,98	179,76
99,7%	1,07	7,95	16,26	84,32	186,55
All rev	1,41	9,39	20,45	103,25	216,69

**Tabela 1:** Średnie czasy z testów algorytmu mergesort, podane w milisekundach



**Wykres 1:** Sortowanie przez scalanie, wizualizacja wyników testów.

Z powyższych danych można zauważyć, że częściowe, czy też pełne odwrotne posortowanie ma jedynie pozytywny wpływ na czas wykonywania algorytmu. Potwierdza się tu wcześniejsze założenie, że liczba operacji jest w tych przypadkach podobna i zachowane jest  $n \log n$ .

### 3 Quicksort

Sortowanie szybkie to algorytm rekurencyjny, który polega na rozbiciu zbioru o wielkości  $n$  na  $n$  podzbiorów o długości 1, przy czym, przy każdym podzieleniu na dwa, wybiera się tzw. pivota, a następnie po jego lewej stronie odkłada się elementy mniejsze od niego, a po prawej większe.

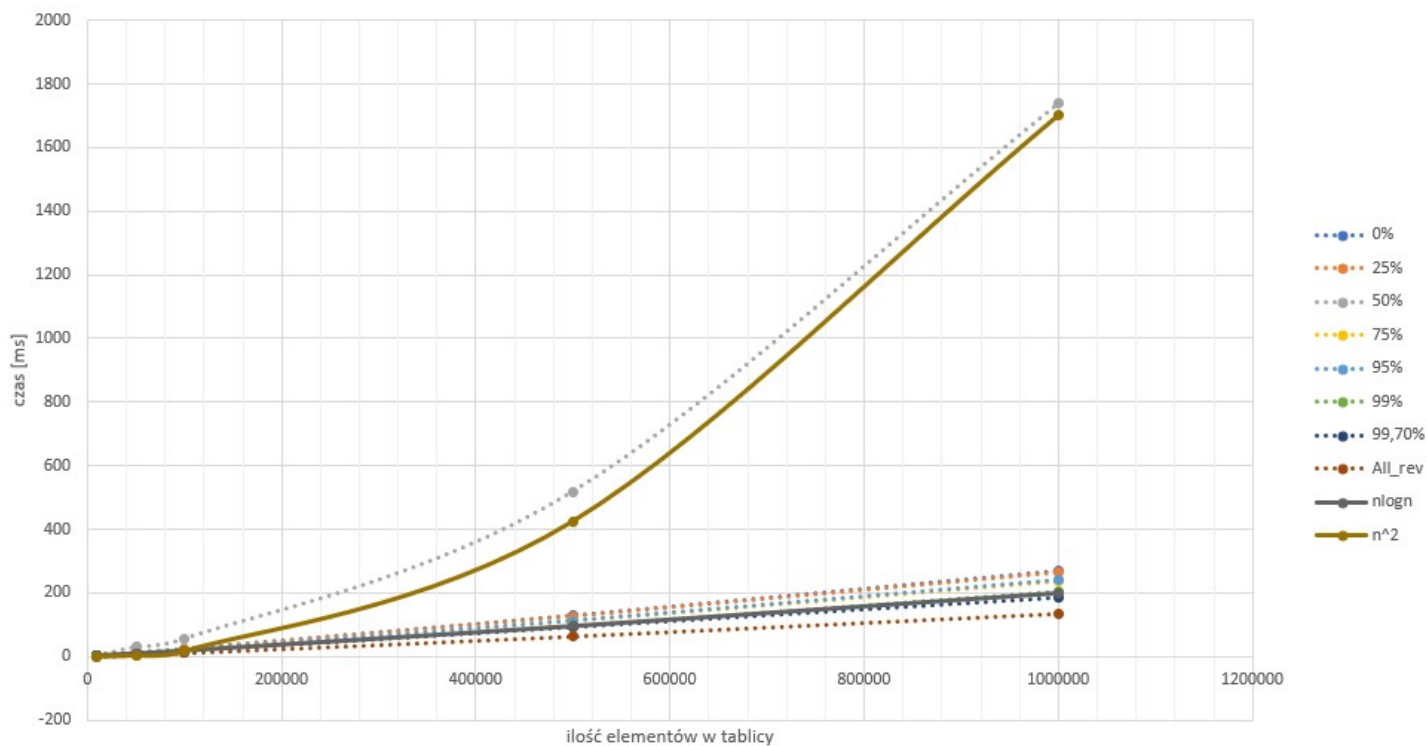
Kluczowy jest tutaj sposób, w jaki wybierzemy pivota. Jedną z możliwości jest wybór pierwszego elementu zbioru, lecz szybko okazuje się, że jeśli często będzie to element większy od pozostałych, to dostajemy najgorszy przypadek złożoności obliczeniowej algorytmu, czyli  $O(n^2)$ .

Statystycznie dobrym sposobem okazuje się wybór środkowego elementu jako pivota, ponieważ przy równomiernym rozkładzie losowania liczb, tutaj będziemy potrzebować najmniej operacji zamiany miejscami. Złożoność wynosi wtedy  $O(n \log n)$ .

Poniżej wyniki testów dla wersji z wyborem pivota w środku zbioru.

	10000	50000	100000	500000	1000000
0%	1,44	10,34	22,15	127,06	267,03
25%	1,08	10,01	21,8	125,97	264,05
50%	3,02	29,72	55,22	518,25	1737,31
75%	1,01	8,35	18,74	110,07	235,65
95%	1,01	8,16	18,57	112,81	241,98
99%	1,02	7,12	15,41	94,51	205,12
99,7%	1	6,21	14,47	91,88	186,65
All rev	0,36	5,17	11,22	63,06	133,24

**Tabela 2:** Średnie czasy z testów algorytmu quicksort, podane w milisekundach



**Wykres 2:** Sortowanie szybkie, wizualizacja wyników testów.

Wśród uzyskanych danych rzuca się w oczy słaba wydajność algorytmu przy wprowadzeniu zbioru posortowanego w 50%, aproksymowana  $n^2$ . Po dodaniu podczas testów zmiennej kontrolującej maksymalną głębokość rekurencji okazuje się, że wacha się ona od 70 do nawet 2000 (dla tablicy 500000 el.). Dla porównania, przy 0% posortowania maksymalna głębokość rekurencji wacha się między 40 a 50.

Na **Wykresie 3** możemy zauważyć, jak wybór pivota psuje się przy posortowaniu 50%. Średnia dległość od środka dochodzi wtedy nawet do 50%, czyli pivot łąduje przy końcu lub początku podzbioru, co jest najgorszym przypadkiem. Oznacza to w praktyce, że po wyborze pivota w środku algorytm musiał wykonać ilość zamian wynoszącą ponad połowę wielkości podzbioru, skoro raz jest przesuwany w lewo, raz w prawo, a i tak dotarł do krawędzi.

Duża ilość zamian w połączeniu z dużą głębokością rekurencji są powodem takiego pogorszenia czasu wykonywania algorytmu.

Sytuacja zupełnie przeciwna, czyli najlepszy przypadek, aproksymowany  $n \log n$  ma miejsce przy pełnym odwrotnym posortowaniu. Pivot wybierany ze środka dzieli podzbiory zawsze na pół, przez co głębokość rekurencji jest najmniejsza. Ponadto ilość zamian dla podzbioru o długości  $n$  zawsze wynosi dokładnie  $\frac{n-1}{2}$ . Nie jest to najlepszy przypadek ogólnie, ale wśród analizowanych, to połączenie małej głębokości rekurencji i średniej ilości zamian okazało się najlepsze.



**Wykres 3:** Jedna kropka to jedno sortowanie, dla 0% i 50% wstępnego posortowania przeprowadzono po 100 sortowań.

Użyty wzór przedstawiono poniżej. Oblicza on średnią odległość pivota od środka dzielonego podzbioru, po podzieleniu na mniejsze i większe, dla całego sortowania.

$$d = \frac{1}{\sum_{i=0}^{M-1} \frac{n_i}{100}} \cdot \sum_{i=0}^{M-1} \frac{n_i}{100} \cdot \text{abs} \left( \frac{\text{end}_i - j_i}{n_i} - 0,5 \right)$$

gdzie:

$d$  - średnia odległość,

$M$  - ilość wywołań funkcji wyboru pivota w sortowaniu,

$n_i$  - wielkość podzbioru  $i$ ,

$\frac{n_i}{100}$  - waga jednej próbki odległości pivota od środka zbioru  $i$ ,

$\text{end}_i$  - ostatni indeks podzbioru  $i$ ,

$j_i$  - indeks pivota w podzbiorze  $i$  po przestawieniu mniejszych na lewo i większych na prawo.

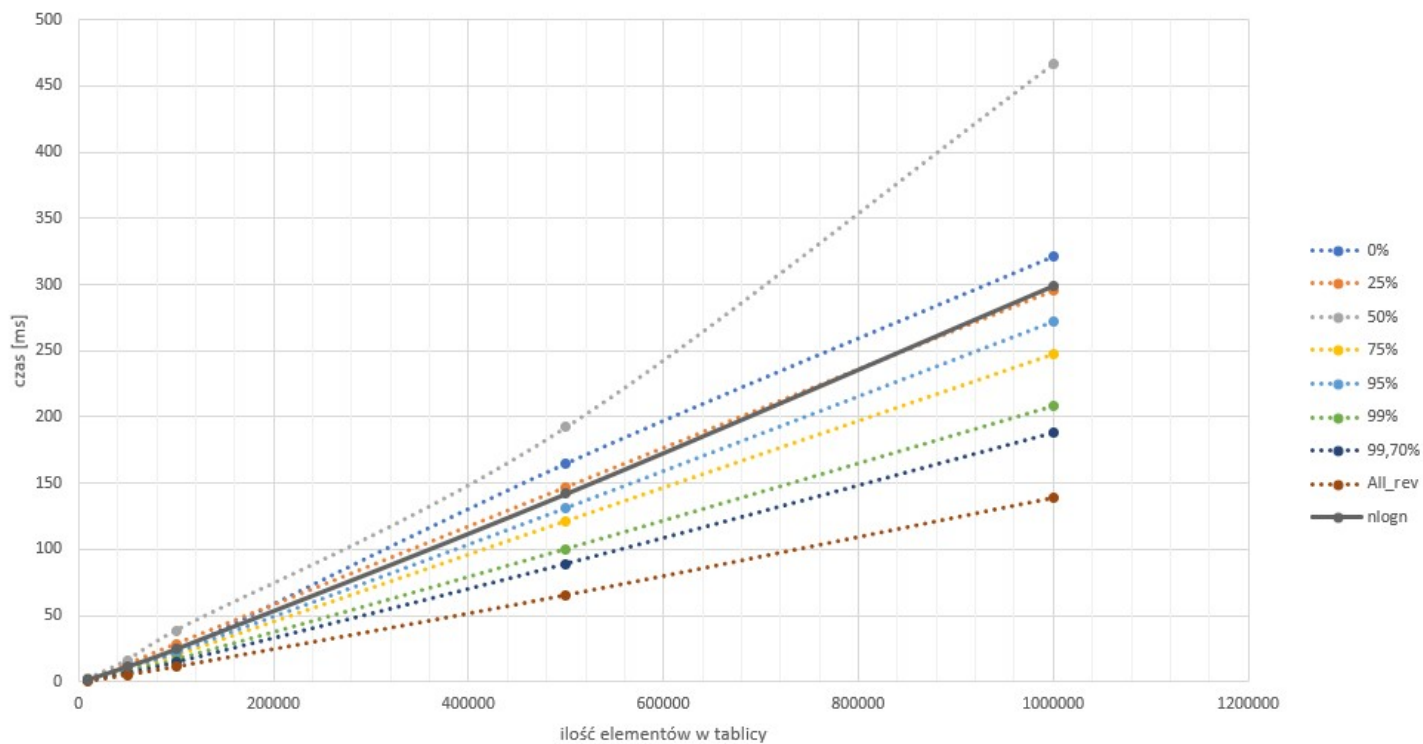
## 4 Sortowanie introspektywne

Introsort to hybrydowy algorytm sortujący, który bazuje na dwóch algorytmach: quicksort i heap sort. Introsort działa jak quicksort dla lepszych przypadków, korzystając z jego prędkości, lecz dla gorszych przypadków, gdzie rekurencja zachodzi zbyt głęboko, wywoływany jest heap sort. Pozwala to na zachowanie złożoności  $O(n \log n)$  dla średnich i najgorszych przypadków.

Wyniki testów:

	10000	50000	100000	500000	1000000
0%	1,32	10,93	22,59	164,9	321,77
25%	1,34	13,24	28,51	147,16	296,04
50%	2,44	16,19	38,68	192,17	467,08
75%	1,01	8,77	19,64	121	247,07
95%	1,08	9,89	21,78	131,15	272,27
99%	1,03	7,73	16,45	100,03	208,03
99,7%	1	6,17	14,32	89,16	188,65
All rev	0,09	5,1	11,13	65,19	138,59

**Tabela 3:** Średnie czasy z testów algorytmu introsort, podane w milisekundach



**Wykres 4:** Sortowanie introspektywne, wizualizacja wyników testów.

Jak widać, dla naszego najgorszego przypadku (dla quicksorta z wyborem pivotu w środku), przy 50% wstępnie posortowanych danych, nastąpiła znaczna poprawa czasu sortowania w stosunku do samego quicksorta. Zdziało ograniczenie głębokości rekurencji i złożoność została zredukowana do  $n \log n$ .



## 5 Źródła

- [wikipedia.org](https://wikipedia.org) - mergesort, quicksort, introsort;
- [4programmers.net](https://4programmers.net) - o złożoności obliczeniowej;
- [algorytm.org](https://algorytm.org) - mergesort;
- [eduinf.waw.pl](https://eduinf.waw.pl) - mergesort, heap sort;
- [researchgate.net](https://researchgate.net) - introsort;
- [algorytm.edu.pl](https://algorytm.edu.pl) - mergesort;
- [gist.github.com/inneme/540829265469e673d045](https://gist.github.com/inneme/540829265469e673d045) - randutils.hpp;
- [youtube.com](https://youtube.com) - wizualizacje dla weny.