Tomasz Jankowiak 249006 27.03.2020

Projektowanie algorytmów i metody sztucznej inteligencji

Projekt 1 – Algorytmy sortowania

Prowadzący:	mgr inż. Marta Emirsajłow
Termin zajęć:	Piątek, 9.15
Numer grupy projektowej:	E08-06i

1. Wprowadzenie

Sortowanie to proces ustawiania zbioru obiektów w określonym porządku. Stosuje się w celu ułatwienia późniejszego wyszukiwania elementów sortowanego zbioru. Jest spotykane powszechnie. Obiekty są posortowane na listach obecności, w książkach telefonicznych, w spisach treści, w bibliotekach, w słownikach, magazynach i w wielu innych miejscach, gdzie potrzebne jest przeszukiwanie i odnajdowanie obiektów składowych. Istotnym problemem w dziedzinie sortowania danych jest ogromna różnorodność algorytmów wykonujących to zadanie.

Celem projektu było zaimplementowanie i przeprowadzenie analizy efektywności trzech wybranych algorytmów sortowania. Wybrano następujące algorytmy: sortowanie przez scalanie, sortowanie szybkie oraz sortowanie introspektywne.

Program napisano w języku C++ i umieszczono w nim wyjaśniające komentarze do niektórych fragmentów kodu. Liczbę elementów do posortowania zdefiniowano za pomocą stałej symbolicznej „WYM1" i odpowiednio zmieniano dla kolejnych testów. Przeprowadzono także wstępną weryfikację poprawności zaimplementowanych algorytmów, wykorzystując małe tablice o małych liczbach, co pozwoliło sprawdzić wizualnie poprawność działania.

Testy efektywności przeprowadzono dla 100 tablic z elementami typu całkowitoliczbowego o rozmiarach 10 000, 50 000, 100 000, 500 000 i 1 000 000 dla różnych stopni posortowania początkowych elementów: 0%, 25%, 50%, 75%, 95%, 99%, 99,7%, oraz dla elementów posortowanych w odwrotnej kolejności. Testy przeprowadzono na tym samym sprzęcie.

2. Opis algorytmów

a) Sortowanie przez scalanie (ang. merge sort)

Jest to rekurencyjny algorytm sortowania danych, wykorzystujący metodę „dziel i zwyciężaj". Został odkryty przez węgierskiego matematyka Johna von Neumanna w 1945 roku. Ideę algorytmu można przedstawić następująco:

- podzielenie n-elementowego zbioru danych na dwa podzbiory po n/2 elementów każdy,
- posortowanie za pomocą metody przez scalanie każdego z podzbiorów osobno dopóki nie zostanie tylko jeden element,
- połączenie posortowanych podciągów w jeden posortowany ciąg.

W programie funkcja „przez_scalanie" rekurencyjnie wywołuje samą siebie oraz wykorzystuje funkcję „scal". Algorytm jest wydajny oraz stabilny. Złożoność obliczeniowa dla każdego przypadku wynosi **O(n log n)**. Praktycznie jedyną wadą algorytmu jest złożoność pamięciowa, ponieważ podczas scalania potrzebny jest dodatkowy obszar pamięci na tablicę pomocniczą, przechowującą kopie podtablic do scalenia. Metoda sortowania przez scalanie jest szybka i sprawdza się nawet dla bardzo dużych danych.

b) Sortowanie szybkie (ang. quicksort)

Jest to jeden z najpopularniejszych algorytmów sortowania, a jego implementacje znajdują się w bibliotekach standardowych wielu środowisk programowania. Działa rekurencyjnie na zasadzie „dziel i zwyciężaj". Został wynaleziony w 1962 roku przez brytyjskiego informatyka C.A.R. Hoare'a.

Procedura sortowania szybkiego polega na następujących czynnościach: należy wybrać element służący za oś podziału (ang. pivot), następnie należy ustawić elementy nie większe na lewo tej wartości, natomiast nie mniejsze na prawo. Na koniec wystarczy każdą z tych podtablic posortować osobno według tego samego schematu. Rekurencja zapewnia poskładanie wyników cząstkowych i w konsekwencji posortowanie całej tablicy.

Algorytm jest niestabilny. Wybór elementu rozdzielającego jest losowy, a zarazem kluczowy, gdyż od niego zależy złożoność algorytmu. Algorytm działa najgorzej przy wyborze najmniejszego (lub największego) elementu z tablicy podawanej sortowaniu, co prowadzi do wywołania wielu instrukcji porównywania, ewentualnych zamian i kolejnych wywołań rekurencyjnych. W pesymistycznym przypadku algorytm w każdym kroku może wybierać taki element, co doprowadzi do głębokości rekurencji równej n, więc całkowita złożoność czasowa będzie wynosić **$O(n^2)$**. W tym przypadku otrzymuje się też olbrzymią, liniową złożoność pamięciową. Zatem, jeśli podziały są zrównoważone

algorytm jest tak szybki jak sortowanie przez scalanie – O(n log n), w przeciwnym przypadku może działać tak wolno jak sortowanie przez wstawianie – $O(n^2)$.

Mimo pesymistycznej złożoności czasowej $O(n^2)$ algorytm w średnim przypadku jest niezmiernie efektywny. Średni czas działania przy losowym wyborze elementu rozdzielającego, dorównuje przypadkowi optymistycznemu. Jego średnia złożoność obliczeniowa jest rzędu **O(n log n)**. Ponadto algorytm nie potrzebuje dodatkowej tablicy do posortowania, jak to jest w przypadku sortowania przez scalanie.

c) Sortowanie introspektywne (ang. introspective sort lub introsort)

Jest sortowaniem hybrydowym, łączącym sortowanie przez kopcowanie, przez wstawianie i szybkie. Rozwiązuje problem złożoności O(n2), występującej w najgorszym przypadku algorytmu sortowania szybkiego. Przyjęto, że małe tablice będą sortowane jednym z elementarnych algorytmów sortowania, który mimo kwadratowej złożoności obliczeniowej, dla zbiorów o niewielkim rozmiarze działa relatywnie szybko. Sortowanie przez kopcowanie zostaje wywołane jako sortowanie pomocnicze tylko wówczas, gdy podziały będą zdegenerowane – badana jest głębokość rekurencji. Dzięki temu eliminujemy najgorszy przypadek algorytmu sortowania szybkiego, tak że zapewniona jest logarytmiczno-liniowa złożoność obliczeniowa: **O(n log n)** zarówno dla średniego, jak i dla najgorszego przypadku.

3. Opracowanie wyników

Na podstawie testów efektywności opisanych we wprowadzeniu (pkt.1) policzono średni czas sortowania jednej tablicy daną metodą. Wyniki przedstawiono w tabelach i na wykresach. Czas podano w milisekundach (1 ms = 10^{-3} s). Na wykresach kolorami zaznaczono początkowe uporządkowanie tablic.

a) Sortowanie przez scalanie (ang. merge sort)

Liczba elementów \ Posortowane elementy	losowo	25%	50%	75%	95%	99%	99,7%	odwr.
10 000	1,46	1,33	1,16	0,95	0,80	0,76	0,79	0,75
50 000	8,55	8,09	6,61	5,44	4,55	4,50	4,51	4,36
100 000	18,1	16,4	14,0	11,7	9,59	9,29	9,19	9,41
500 000	103	92,1	78,9	64,2	56,0	51,8	52,1	51,6
1 000 000	215	191	165	136	113	110	109	109

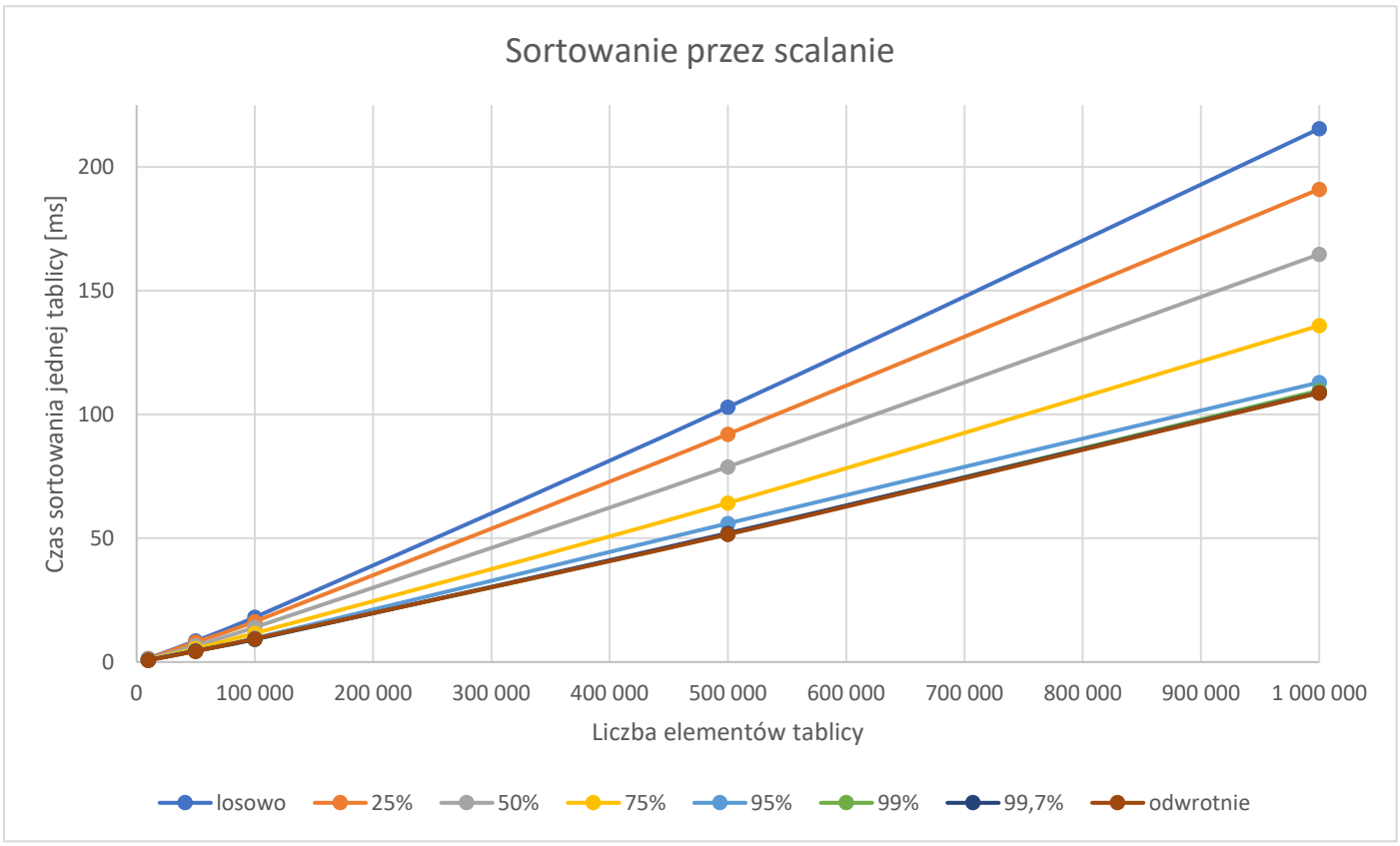
b) Sortowanie szybkie (ang. quicksort)

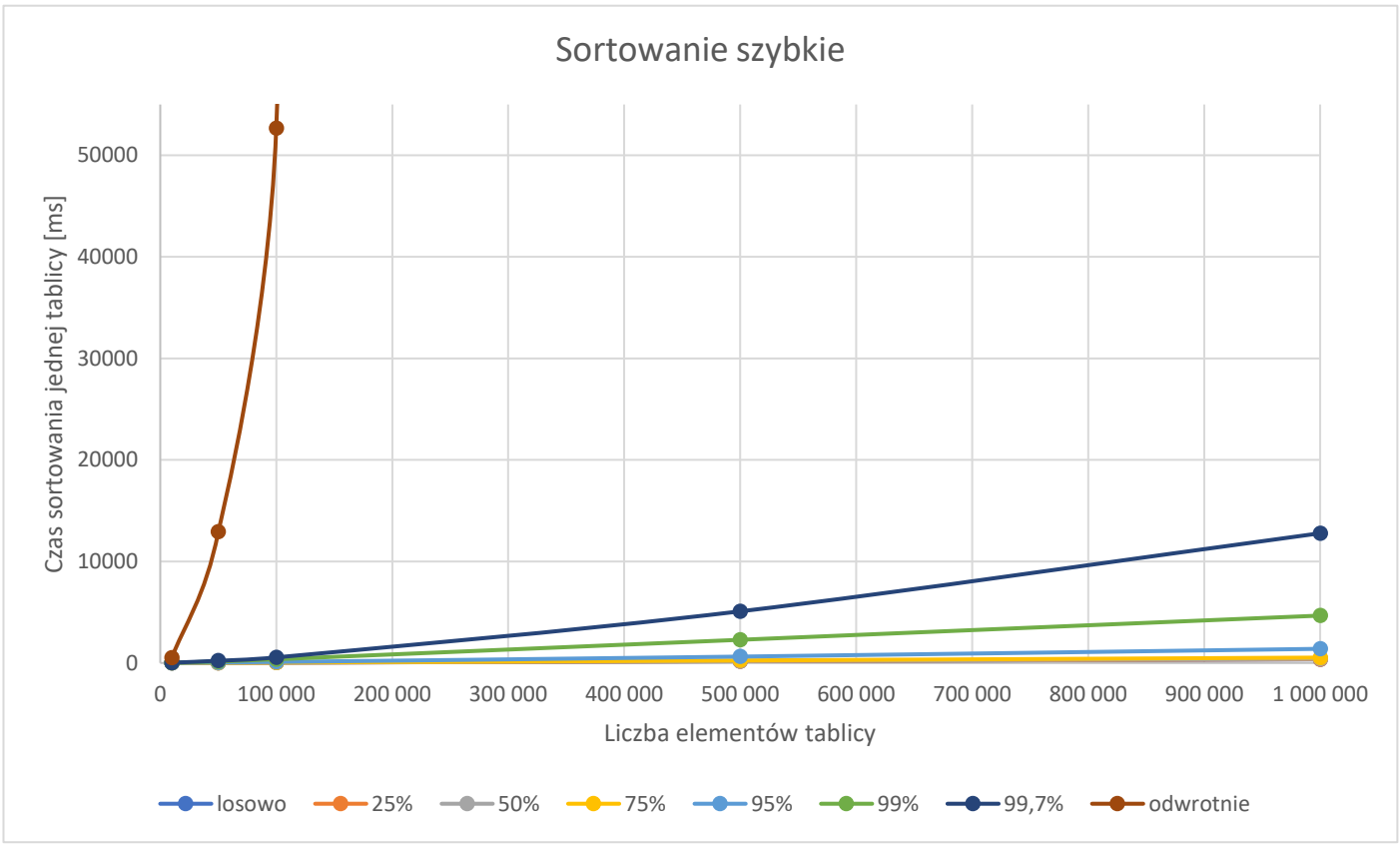
Liczba elementów \ Posortowane elementy	losowo	25%	50%	75%	95%	99%	99,7%	odwr.
10 000	2,39	2,38	2,55	3,15	8,04	14,1	24,1	517
50 000	13,8	14,1	15,9	19,9	46,8	143	217	12906
100 000	29,7	30,2	33,1	41,5	101	361	549	52676
500 000	169	175	191	245	626	2280	5084	
1 000 000	356	369	405	527	1392	4665	12773	

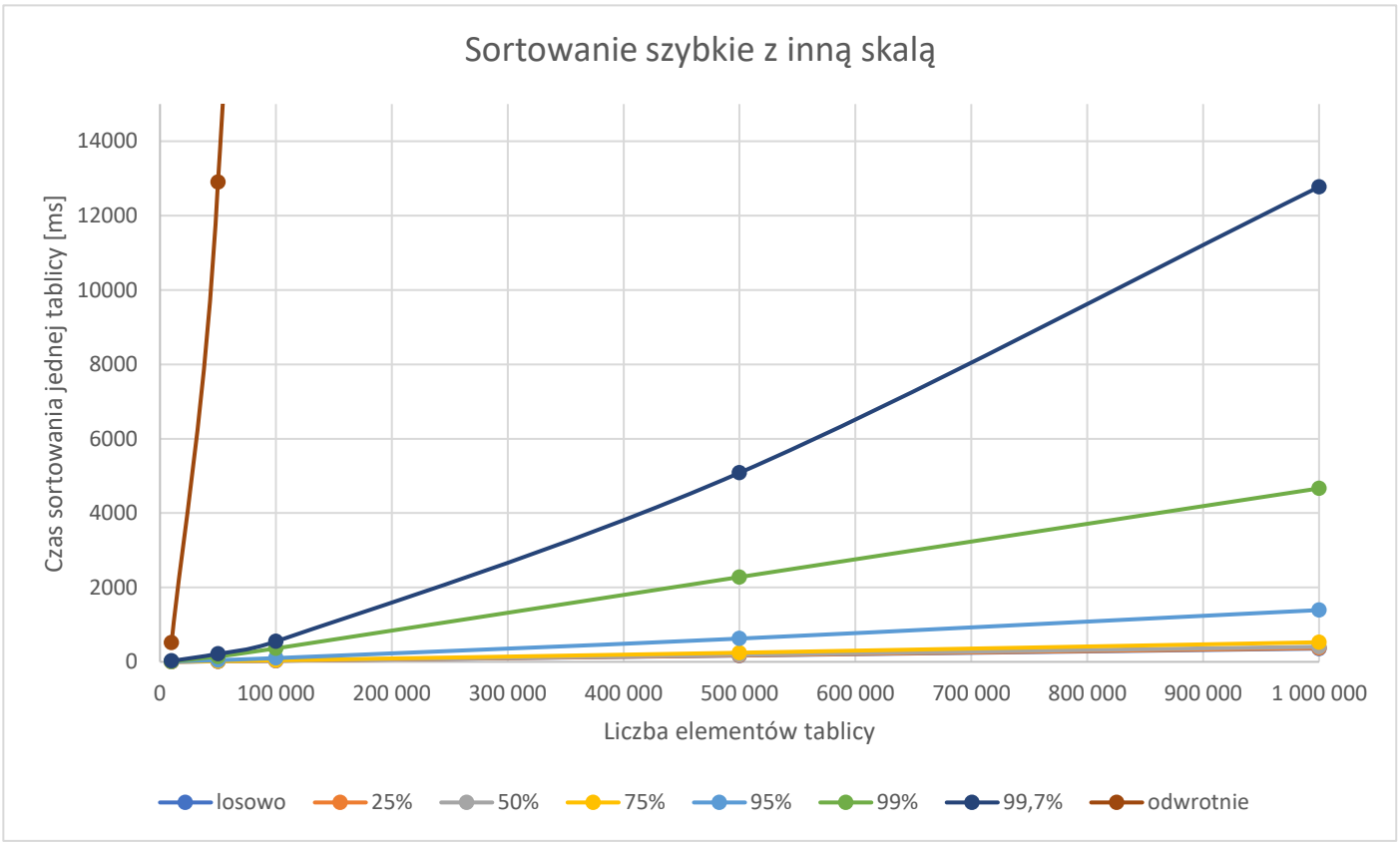
W implementacji algorytmu sortowania szybkiego za element rozdzielający (pivot) wybrano pierwszy element tablicy. Gdy tablica jest uporządkowana w odwrotnej kolejności, algorytm za każdym razem wybiera element największy. Prowadzi to do głębokości rekurencji równej n, więc całkowita złożoność czasowa wynosi $O(n^2)$. W tym przypadku występuje także olbrzymia, liniowa złożoność pamięciową. Średnia wartość dla tablicy 100 000 - elementowej została oszacowana na podstawie sortowania 10 tablic zamiast 100. Z kolei dla tablicy z 500 000 elementów po pewnym czasie wystąpił błąd ochrony pamięci (segmentation fault) nawet przy próbie posortowania tylko jednej tablicy. Dzięki wiedzy, że złożoność obliczeniowa jest kwadratowa można oszacować, że gdybyśmy operowali nieskończoną pamięcią to czas sortowania jednej półmilionowej tablicy w przybliżeniu byłby równy 22 minuty, a czas sortowania jednej milionowej tablicy wynosiłby około 1,5 godziny. Na szczęście eksperyment w porę przerwano i udało się tego uniknąć, do tragedii nie doszło.

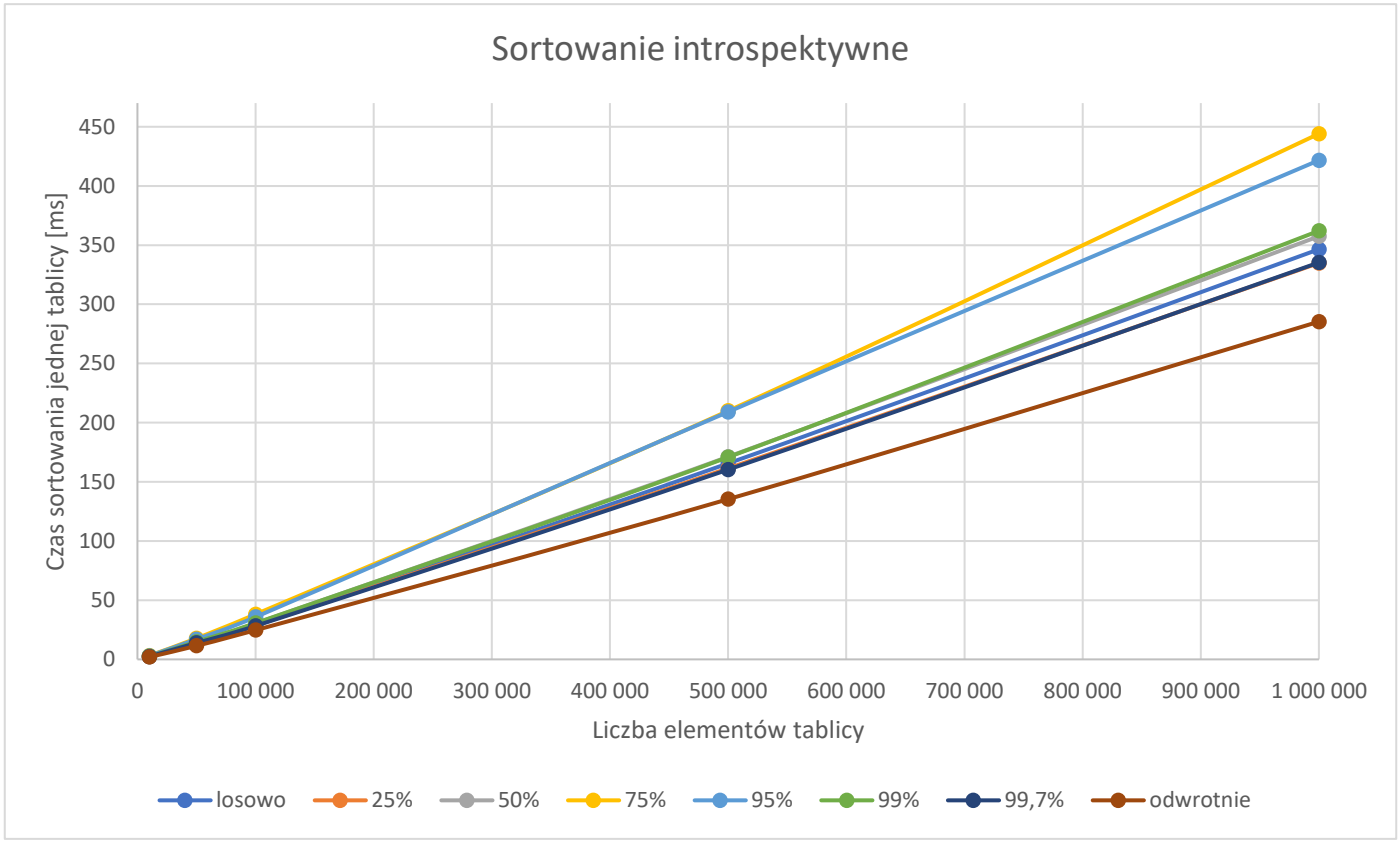
c) Sortowanie introspektywne (ang. introspective sort lub introsort)

Liczba elementów \ Posortowane elementy	losowo	25%	50%	75%	95%	99%	99,7%	odwrotnie
10 000	2,35	2,34	2,51	3,12	2,98	2,51	2,42	1,98
50 000	13,8	14,2	14,3	17,8	17,0	14,9	13,9	11,5
100 000	29,1	28,7	30,5	38,1	35,8	30,7	28,3	24,7
500 000	166	161	171	210	209	171	160	135
1 000 000	346	335	357	444	422	362	335	285






Sortowanie szybkie z inną skalą
Czas sortowania jednej tablicy [ms]
14000
12000
10000
8000
6000
4000
2000
0
0
100 000
200 000
300 000
400 000
500 000
600 000
700 000
800 000
900 000
1 000 000
Liczba elementów tablicy
losowo
25%
50%
75%
95%
99%
99,7%
odwrotnie


Sortowanie introspektywne
Czas sortowania jednej tablicy [ms]
450
400
350
300
250
200
150
100
50
0
0
100 000
200 000
300 000
400 000
500 000
600 000
700 000
800 000
900 000
1 000 000
Liczba elementów tablicy
losowo
25%
50%
75%
95%
99%
99,7%
odwrotnie

4. Wnioski

Wszystkie trzy przedstawione algorytmy sortowania są szybkie i wydajne. Każda metoda ma swoje wady i zalety, które zostały szczegółowo omówione w opisie algorytmów (pkt.2). Niektóre z nich udało się wykazać podczas testów efektywności w programie. Krzywe na wykresach w przybliżeniu pokrywają się ze złożonościami obliczeniowymi danych algorytmów. Najbardziej uniwersalny i najszybszy wydaje się być algorytm sortowania przez scalanie. Jest on stabilny. Sortowanie szybkie może być zaimplementowane na wiele sposobów. Quicksort sprawdza się jedynie dla danych ułożonych zgodnie z rozkładem normalnym prawdopodobieństwa i może napotkać problem jeśli tablica nie jest losowa. Wybór elementu osiowego wpływa na równomierność podziału na podtablice. Wybór pierwszego elementu tablicy nie sprawdza się w przypadku, gdy tablica jest już prawie uporządkowana, a w przypadku odwrotnego posortowania tablicy działa on po prostu fatalnie. Problem rozwiązuje hybrydowe sortowanie introspektywne, które jest szybsze od quicksorta nawet dla zwykłych losowych danych. Jest ono warte uwagi.

Otrzymane wyniki są zgodne z przewidywaniami. Potwierdzają zatem teorię oraz poprawność zaimplementowania algorytmów i przeprowadzenia testów.

5. Literatura

- P. Wróblewski – „Algorytmy, struktury danych i techniki programowania”, wydanie V, Helion 2015
- T.H. Cormen, C.E. Leiserson, R.L. Rivest – „Wprowadzenie do algorytmów”, wyd. IV, WNT, Warszawa 2001
- http://cpp0x.pl/dokumentacja/standard-C/rand/580
- http://cpp0x.pl/kursy/Kurs-C++/Poziom-2/Pseudolosowe-liczby-calkowite/290
- http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-scalanie.html
- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
- https://pl.wikipedia.org/wiki/John_von_Neumann
- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://pl.wikipedia.org/wiki/C.A.R._Hoare
- http://cpp0x.pl/kursy/Kurs-C++/Biblioteka-time-h/321
- https://4programmers.net/Forum/Newbie/198832-funkcja_time_w_milisekundach
- http://www.cplusplus.com/reference/ctime/clock/
- http://cpp0x.pl/kursy/Kurs-C++/Dynamiczne-zarzadzanie-pamiecia-new-i-delete/307
- http://cpp0x.pl/kursy/Kurs-C++/Poziom-5/Zarzadzanie-pamiecia-new-delete/581
- https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
- https://en.wikipedia.org/wiki/Introsort
- https://www.geeksforgeeks.org/know-your-sorting-algorithm-set-2-introsort-cs-sorting-weapon/
- http://www.cplusplus.com/reference/cmath/log2/
- N. Wirth – „Algorytmy + struktury danych = programy”, wydanie VI, WNT, Warszawa 2002