Projekt nr 1: Algorytmy sortowania

Krystian Mirek

April 2020

1 Wprowadzenie

1.1 Opis

Projekt miał na celu przetestowanie trzech algorytmów sortowania pod względem czasu wykonania zadanych mu operacji.
W programie przeanalizowano następujące algorytmy:

- Mergesort
- QuicSort
- IntroSort

Program bada efektywność algorytmów na różnych rozmiarach tablic, te rozmiary to:

- 10 000
- 50 000
- 100 000
- 500 000
- 1 000 000

Program również bada efektywność dla różnego poziomu wstępnego posortowania elementów, te wartości to:

- Wszystkie elementy losowe
- 25%
- 50%
- 75%
- 95%
- 99%
- 99,7%
- Wszystkie elementy posortowane ale w odwrotnej kolejności

1.2 Opis algorytmów

1.2.1 MergeSort

Algorytm stosujący metodę divide and conquer. Działanie MergeSort'a można opisać w dwóch prostych krokach:

- Dziel rekurencyjnie na połówki cały zestaw danych aż do osiągnięcia jednoelementowych podproblemów
- Scalaj z powrotem kolejne podproblemy wraz z ich sortowaniem aż do osiągnięcia posortowanego zestawu danych

Złożoność obliczeniowa MergeSort'a wygląda następująco:

- Najlepszy przypadek $O(nlogn)$
- Typowy przypadek $O(nlogn)$
- Najgorszy przypadek $O(nlogn)$

Złożonośc pamięciowa: $O(nlogn)$

MergeSort jest algorytmem stabilnym, co oznacza, że jeśli przed sortowaniem mamy elementy o takiej samej wartości, które są względem siebie ułożone w konkretnej kolejności to po sortowaniu te elementy będą nadal w takiej samej kolejności.
Złożoność pamięciowa MergeSort'a wynika z potrzeby posiadania dodatkowej tymczasowej struktury danych. Jest to fakt który decyduje o tym, że HeapSort (algorytm sortujący „w miejscu") jest preferowanym algorytmem nad MergeSort'em.

1.2.2 QuickSort

Algorytm, który również stosujący metodę divide and conquer. Działanie QuickSort'a można opisać w dwóch prostych krokach:

- Wybierz na zasadzie określonych zasad element rozdzielający (pivot), który rozdziela zestaw danych na dwie części. Ważne jest, że elementy większe od pivota przenoszone są do części po jego prawej stronie a mniejsze do części po lewej
- Wykonuj rekursywnie powyższą czynność dla kolejnych fragmentów zestawu danych aż do osiągnięcia jednoelementowego zbioru który jest uważany za posortowany.

Złożoność obliczeniowa QuickSort'a wygląda następująco:

- Najlepszy przypadek $O(nlogn)$
- Typowy przypadek $O(nlogn)$
- Najgorszy przypadek $O(n^2)$

Złożoność pamięciowa: $O(logn)|O(n)$

Quick nie potrzebuje dodatkowej struktury danych żeby sortować, tzn. sortuje on „w miejscu". Złożoność pamięciowa O(n) wynika z występowania pesymistycznego przypadku quicka. QuickSort jest niestabilny. Z uwagi na możliwość manipulacji wyboru pivota, algorytm ten może działać z bardzo rożną wydajnością.

1.2.3 IntroSort

Algorytm ten jest hybrydowy, tzn. że składa się on z kilku innych algorytmów sortowania. Na IntroSort'a składają się: QuickSort, HepSort, InsertionSort. Działanie IntroSort'a można opisać w kilku prostych krokach:

- Partycjonuj dane w taki sam sposób jak QuickSort, czyli wykorzystując element rozdzielający.
- Jeżeli głębokość rekurencji (dozwolona głębokość wywołań rekurencyjnych określana jako współczynnik $2 * logn$ jest równa 0 przełącz się na HeapSort'a.
- Jeżeli pozostała ilość danych w określonym fragmencie jest mniejsza od 16 to należy przełączyć się na InsertionSort'a i posortować fragment do końca.

Złożoność obliczeniowa IntroSort'a wygląda następująco:

- Najlepszy przypadek $O(nlogn)$
- Typowy przypadek $O(nlogn)$
- Najgorszy przypadek $O(nlogn)$

Złożoność pamięciowa: $O(logn)$

Intro również nie potrzebuje dodatkowej struktury danych żeby sortować oraz

jest niestabilny. Algorytm ten został stworzony aby wyeliminować najgorszy przypadek złożoności obliczeniowej Quick'a. Warto wspomnieć że jest to algorytm którego używa język programowania c++ pod instrukcją std::sort() w nagłówku $< algorithm >$.

1.3 Porównanie

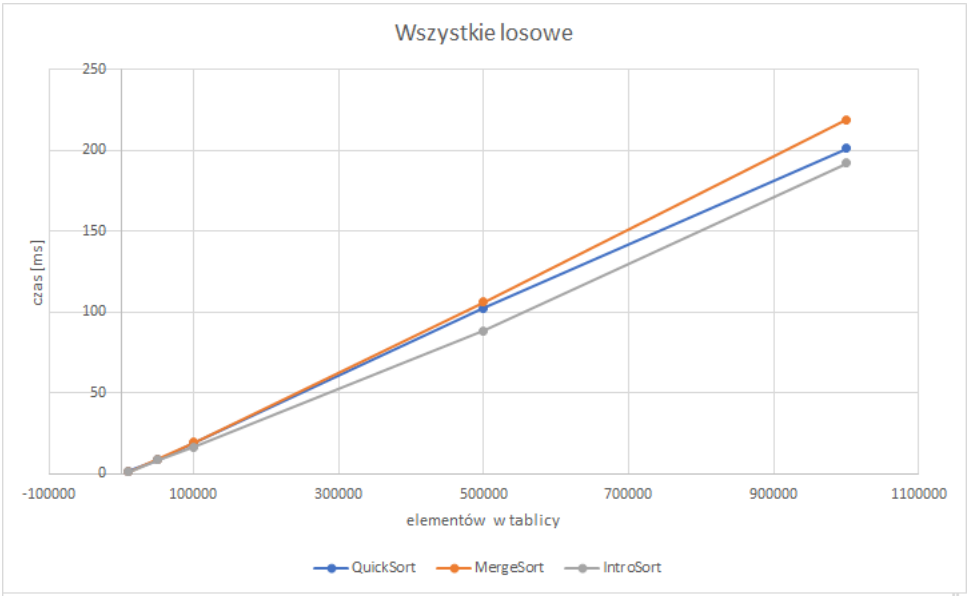
1.3.1 Merge vs Quick vs Intro

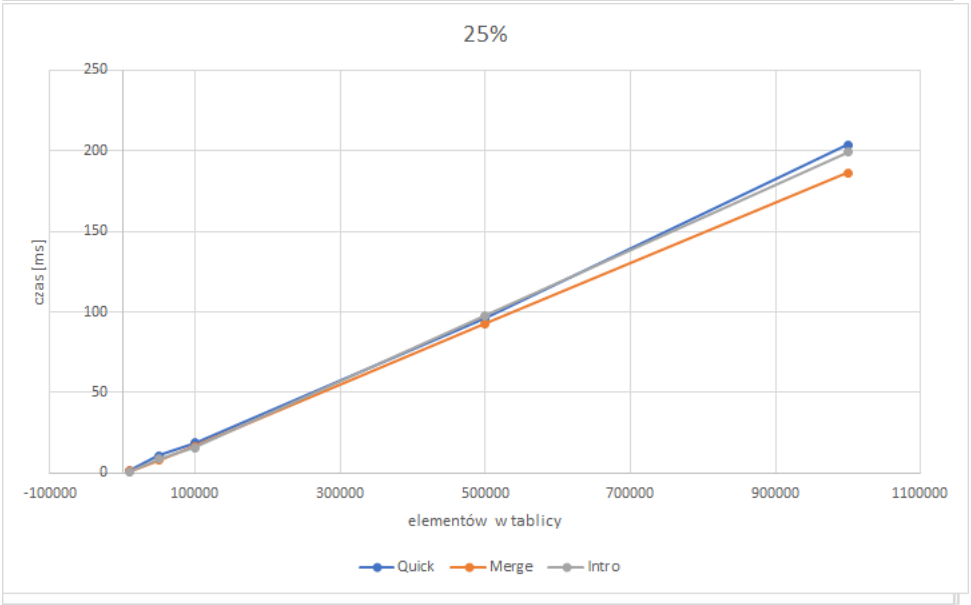
	Merge	**Quick**	**Intro**
Złożoność pamięciowa	$O(n)$	$O(logn)\|O(n)$	$O(logn)$
Najgorszy przypadek złożoności czasowej	$O(nlogn)$	$O(n^2)$	$O(nlogn)$
Stabilność	stabilny	niestabilny	niestabilny
Lepsza wydajność	równa dla każdej wielkości zestawu	mniejsze zestawy danych	dobra dla każdej wielkości zestawu
Kompatybilność z pamięcią podręczną procesora	słaba	dobra	dobra

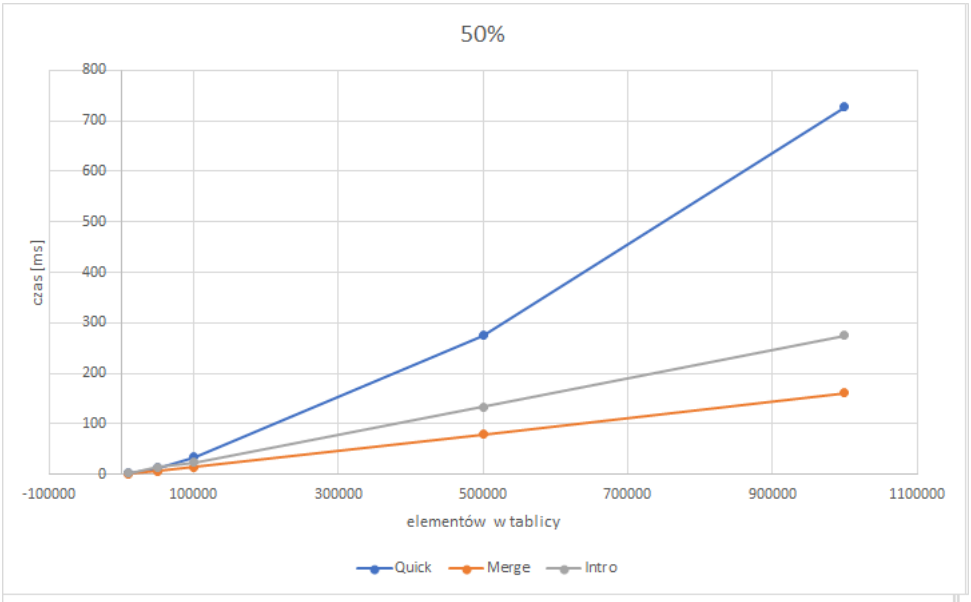
1.4 Przewidywane wyniki

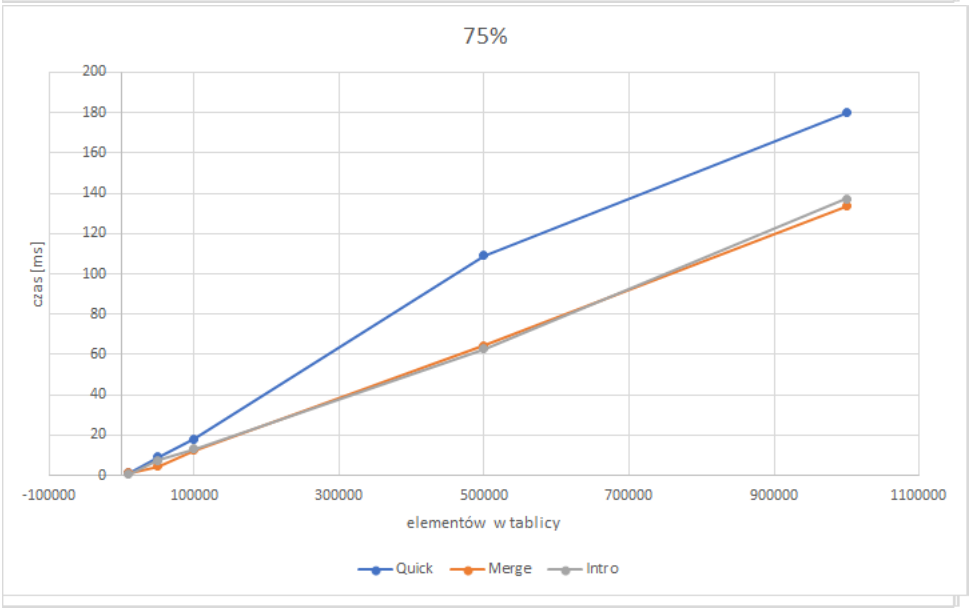
Analizując algorytmy można stwierdzić że najszybszy powinien być IntroSort a najwolniejszy MergeSort. Jednak w przypadku dużych ilości danych, MergeSort powinien się lepiej sprawdzić od QuickSorta; również dla większego stopnia wstępnego posortowania Merge może się zachowywać lepiej niż Quick z uwagi na dużą głębokość rekurencji jaką będzie Quick musiał pokonać. Intro z uwagi na swoja wielozadaniowość oraz przystosowanie do różnych warunków pracy powinien być najlepszym ze wszystkich algorytmów. Taka zależność powinna wystąpić w przypadku użycia tradycyjnej implementacji algorytmów, w projekcie natomiast użyta jest niekonwencjonalna metoda implementacji MergeSorta. Polega ona na stworzeniu tej dodatkowej struktury danych której potrzebuje MergeSort zupełnie niezależnie od niego, tzn. nie będzie ona tworzona za każdą rekurencją, co powinno drastycznie przyspieszyć jego działanie. Taką operacją sztucznie wymuszamy żeby MergeSort sortował „w miejscu". Dzięki takiemu działaniu usuwamy zupełnie czynnik niekompatybilności z pamięcią Cache co stawia Merge w pozycji konkurencyjnej do Quicka a nawet do Intro.

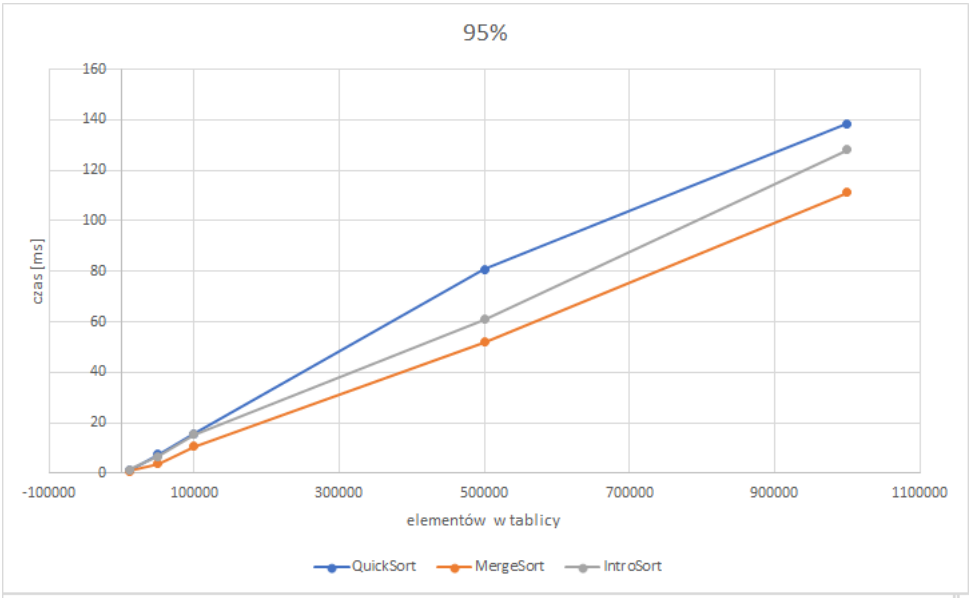
2 Testy

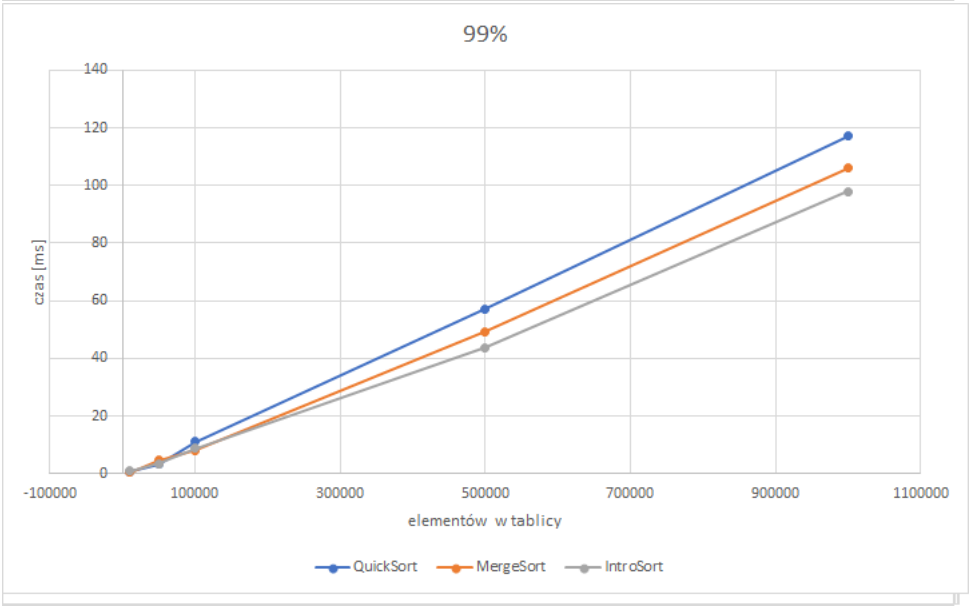

Wszystkie losowe
250
200
150
100
50
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
QuickSort
MergeSort
IntroSort

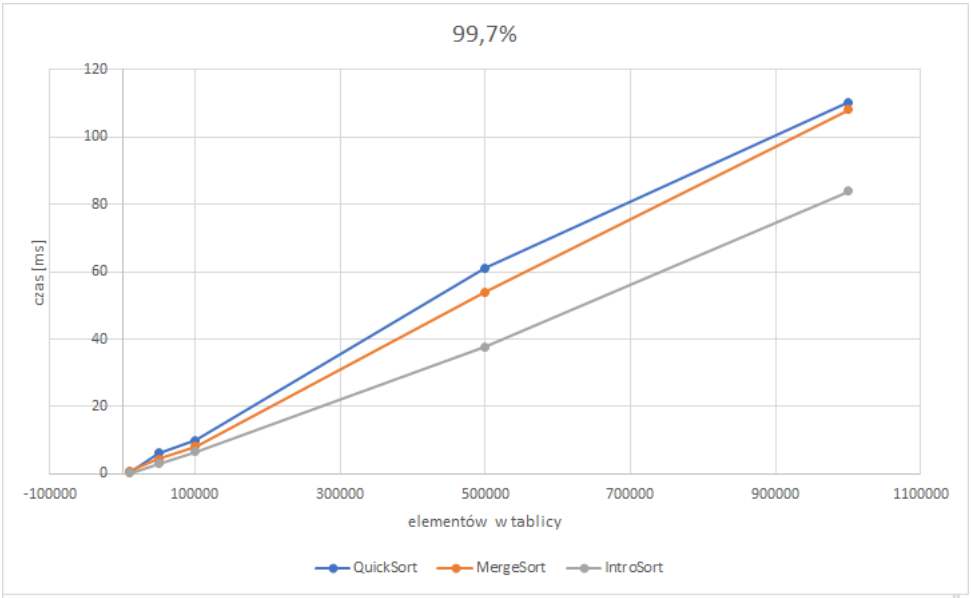

25%
250
200
150
100
50
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
Quick
Merge
Intro

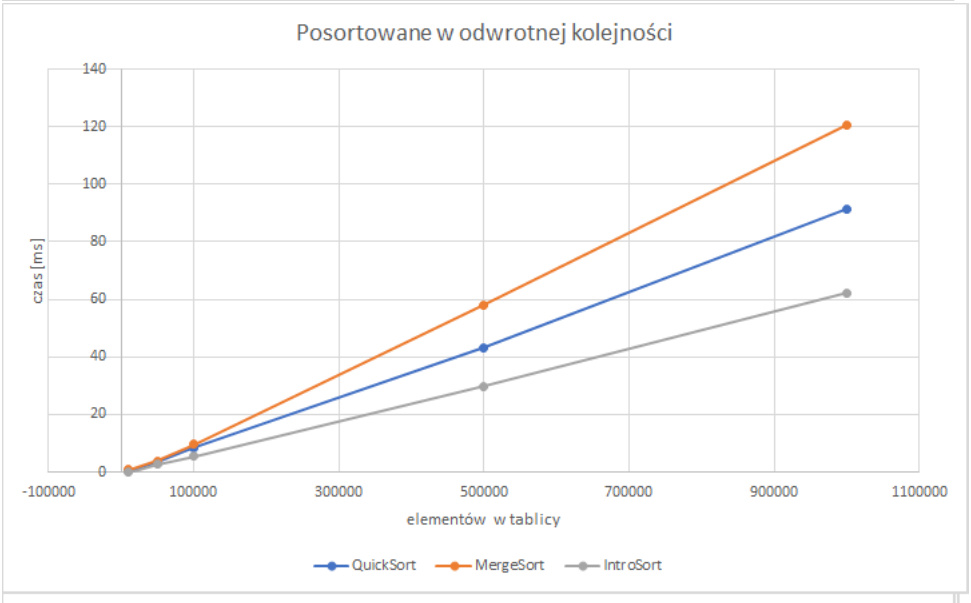

50%
800
700
600
500
400
300
200
100
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
Quick
Merge
Intro


75%
200
180
160
140
120
100
80
60
40
20
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
Quick
Merge
Intro


95%
160
140
120
100
80
60
40
20
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
QuickSort
MergeSort
IntroSort


99%
140
120
100
80
60
40
20
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
QuickSort
MergeSort
IntroSort


99,7%
120
100
80
60
40
20
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
QuickSort
MergeSort
IntroSort


Posortowane w odwrotnej kolejności
140
120
100
80
60
40
20
0
czas [ms]
-100000
100000
300000
500000
700000
900000
1100000
elementów w tablicy
QuickSort
MergeSort
IntroSort

2.1 średnie czasy dla sortowania 100 tablic

	rozmiartablicy	**quick[ms]**	**merge[ms]**	**intro[ms]**
0%	10000	1,38057	1,00311	1,23999
	50000	8,57811	8,93583	8,61797
	100000	18,7447	19,2756	16,396
	500000	102,228	106,293	88,3176
	1000000	201,45	219,076	192,29
25%	10000	1,90864	1,31559	1,15978
	50000	10,9668	7,85355	9,04602
	100000	18,5781	16,6033	15,8911
	500000	95,9245	92,6137	97,9066
	1000000	203,771	186,223	198,872
50%	10000	2,53998	1,09388	1,91571
	50000	13,4705	6,46149	14,0182
	100000	33,4894	14,641	24,1716
	500000	274,611	78,5234	133,929
	1000000	727,297	160,569	274,671
75%	10000	1,17477	1,31565	0,756306
	50000	9,05087	4,67589	7,65779
	100000	18,013	12,6278	13,1228
	500000	109,047	64,3538	62,8069
	1000000	179,771	133,439	137,051
95%	10000	1,07939	0,852412	1,40646
	50000	7,44666	3,69927	6,737
	100000	15,7726	10,6844	15,3255
	500000	80,7894	51,9928	60,8456
	1000000	138,487	111,247	128,202
99%	10000	0,690329	0,533808	0,781357
	50000	3,257	4,50167	3,38689
	100000	10,9865	8,10134	8,6487
	500000	57,205	49,2055	43,8128
	1000000	117,14	105,913	97,8576
99,7%	10000	0,690128	0,781856	0,301884
	50000	6,19958	4,30935	3,07468
	100000	9,84791	8,09845	6,52351
	500000	61,195	53,9326	37,7114
	1000000	110,414	108,138	83,9946
odwrotnie	10000	0,377964	0,846361	0
	50000	3,28588	3,82963	2,63187
	100000	8,52097	9,71981	5,36684
	500000	43,3024	57,9932	29,7866
	1000000	91,532	120,62	62,2017

3 Podsumowanie

- Największym odstępstwem w czasach algorytmów wykazuje się wykres dla tablicy posortowanej w 50%. Może być to spowodowane podobną zasadą.
- MergeSort stał się konkurencyjny dla pozostałych algorytmów. Można zauważyć zależność, że pobija on QuickSort'a ale przegrywa w niektórych sytuacjach z IntroSort'em; dzieje się tak w przypadku kiedy dane nie wymagają za dużo porównań, tj. kiedy dane są już w jakimś stopniu posortowane. Kiedy natomiast dane są zupełnie nieuporządkowane a już w szczególności kiedy są posortowane w odwrotnej kolejności, Merge staje się gorszy również od Qucik'a.
- Zgodnie z założeniami Quick jest generalnie wolniejszy od Intro.
- Przewidywania wyników wysunięte na podstawie założeń teoretycznych okazały się mniej lub więcej odzwierciedlać realne wyniki testów. Oznacza to, że algorytmy zostały zaimplementowane w poprawny sposób oraz, że testy zostały przeprowadzone w sposób kontrolowany.