Michał Więcek
248956
03.04.2020

Projektowanie Algorytmów i Metody Sztucznej Inteligencji
Projekt 1

Mgr inż. Marcin Ochman
Grupa: Wtorek 15^{15}-16^{55}

1. Wstęp

Celem projektu jest porównanie różnych sposobów sortowania oraz sprawdzenie jaki wpływ na czas sortowania ma ich złożoność obliczeniowa. Zaimplementowane algorytmy: sortowanie szybkie, sortowanie przez scalanie, sortowanie introspektywne. Złożoność obliczeniowa tych algorytmów to O ($n\ logn$), gdzie dla sortowania szybkiego trzeba uwzględnić pesymistyczny przypadek O(n^2). Sortowanymi obiektami były dynamicznie zaalokowane tablice wypełnione pseudolosowo przez funkcję `rand()`.

2. Sortowanie szybkie

Sortowanie szybkie jest algorytmem rekurencyjnym. Dla tablicy elementów wybiera jeden element zwany osią i tworzy dwie tablice. Jedna z elementami mniejszymi od elementu osiowego i drugą z elementami większymi. Algorytm następnie wywołuje się rekurencyjnie dla utworzonych tablic doprowadzając do ich uporządkowania. Jego złożoność obliczeniowa dla przypadku średniego to O($n\ logn$). W sytuacji kiedy jako oś wybrany zostanie element największy lub najmniejszy złożoność obliczeniowa wynosi O(n^2).

3. Sortowanie przez scalanie

Sortowanie przez scalanie jest algorytmem rekurencyjnym. Dzieli tablice na dwie mniejsze tablice do momentu uzyskania tablic jednoelementowych. Kiedy cały zestaw danych jest podzielony na pojedyncze elementy zaczyna scalać i sortować elementy aż do uzyskania całego zestawu danych. Algorytm ma złożoność obliczeniową O($n\ logn$).

4. Sortowanie introspektywne

Sortowanie introspektywne jest hybrydowym algorytmem sortowania. Znaczy to, że składa się on z kilku innych algorytmów sortowania. Główną zaletą tego typu algorytmu jest wyeliminowanie pesymistycznego przypadku sortowania szybkiego. Zaczyna on sortowanie za pomocą sortowania szybkiego. W trakcie działania sortowania badana jest maksymalna głębokość wywołań rekurencyjnych. Jeśli wywołania te osiągną 0, to wywołania rekurencyjne kończą się, a dla danego podproblemu jest wykorzystywane sortowanie przez kopcowanie. Jeśli uzyskano w trakcie sortowania tablice o rozmiarze mniejszym od 10 to sortowane są one przy pomocy sortowania przez wstawianie który pomimo swojej złożoności obliczeniowej radzi sobie lepiej z małymi zestawami danych.

5. Przebieg eksperymentów

Michał Więcek
248956

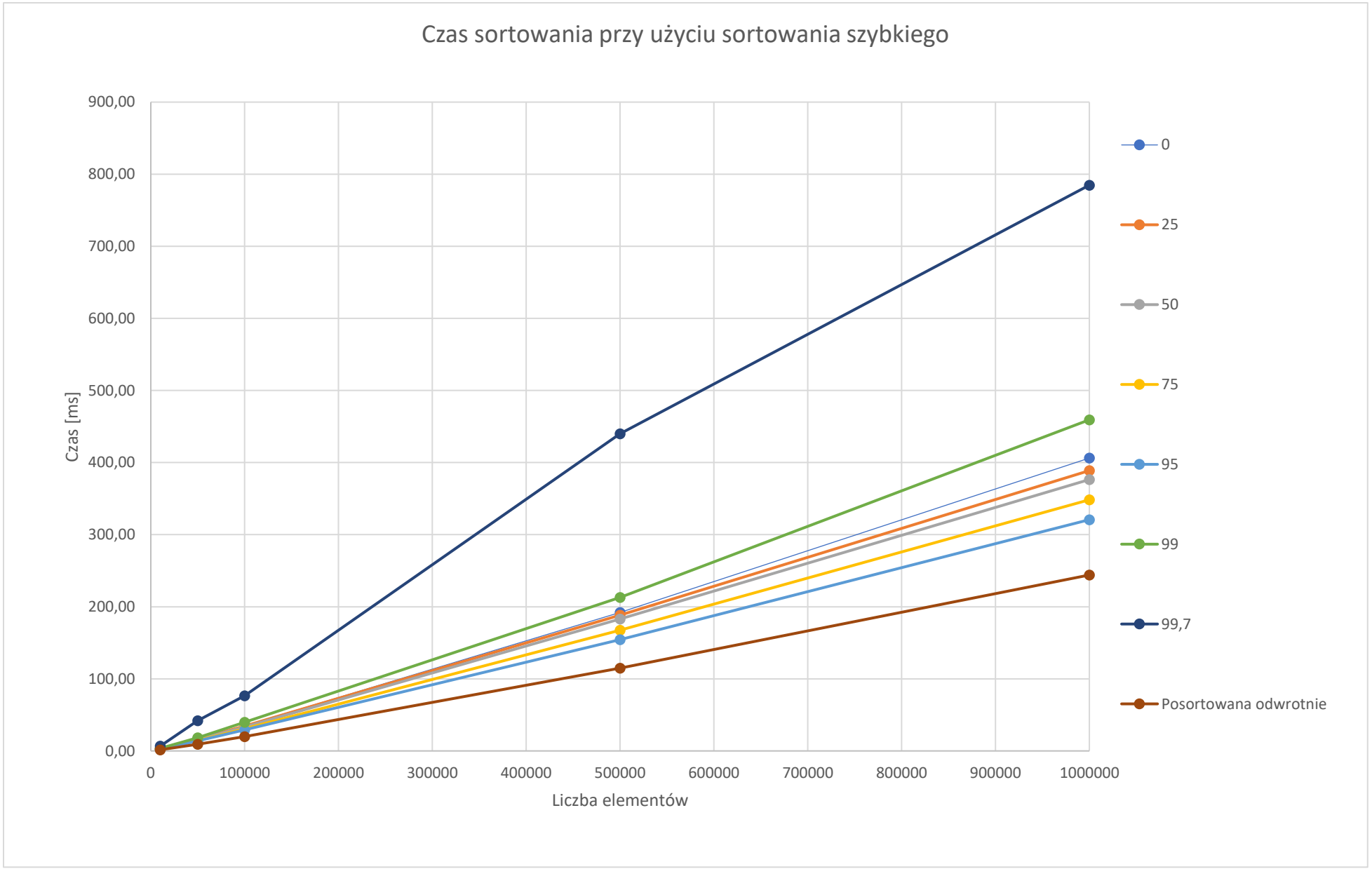
03.04.2020

a. Sortowanie szybkie

Posortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	2,96	16,46	34,48	188,80	390,19
	Średni	3,01	16,79	34,84	192,03	406,24
	Największy	3,28	20,26	37,44	222,62	603,37
25	Najmniejszy	2,93	16,28	33,85	186,55	385,03
	Średni	3,01	16,59	34,44	188,48	388,86
	Największy	4,20	20,07	37,25	200,93	454,77
50	Najmniejszy	2,80	15,56	32,54	177,54	367,32
	Średni	2,87	15,95	33,23	182,93	376,44
	Największy	3,23	16,94	34,19	234,42	436,84
75	Najmniejszy	2,59	14,32	29,78	163,88	338,10
	Średni	2,67	14,70	30,66	167,54	348,38
	Największy	3,73	17,39	32,04	178,04	442,56
95	Najmniejszy	2,23	12,43	25,94	140,81	293,02
	Średni	2,51	13,78	29,13	154,54	320,77
	Największy	3,69	23,62	62,33	217,71	484,08
99	Najmniejszy	2,17	12,51	26,48	138,40	285,76
	Średni	3,61	18,32	39,89	212,80	459,35
	Największy	11,15	52,69	105,52	483,20	1255,56
99,7	Najmniejszy	2,48	15,94	30,14	178,57	350,75
	Średni	6,67	41,88	76,54	440,12	784,81
	Największy	20,13	146,01	298,99	2373,49	2273,24
Posortowana odwrotnie	Najmniejszy	1,57	9,20	19,69	114,04	243,00
	Średni	1,58	9,28	19,88	114,74	244,02
	Największy	1,66	9,66	20,71	123,63	245,45

Michał Więcek
248956

03.04.2020



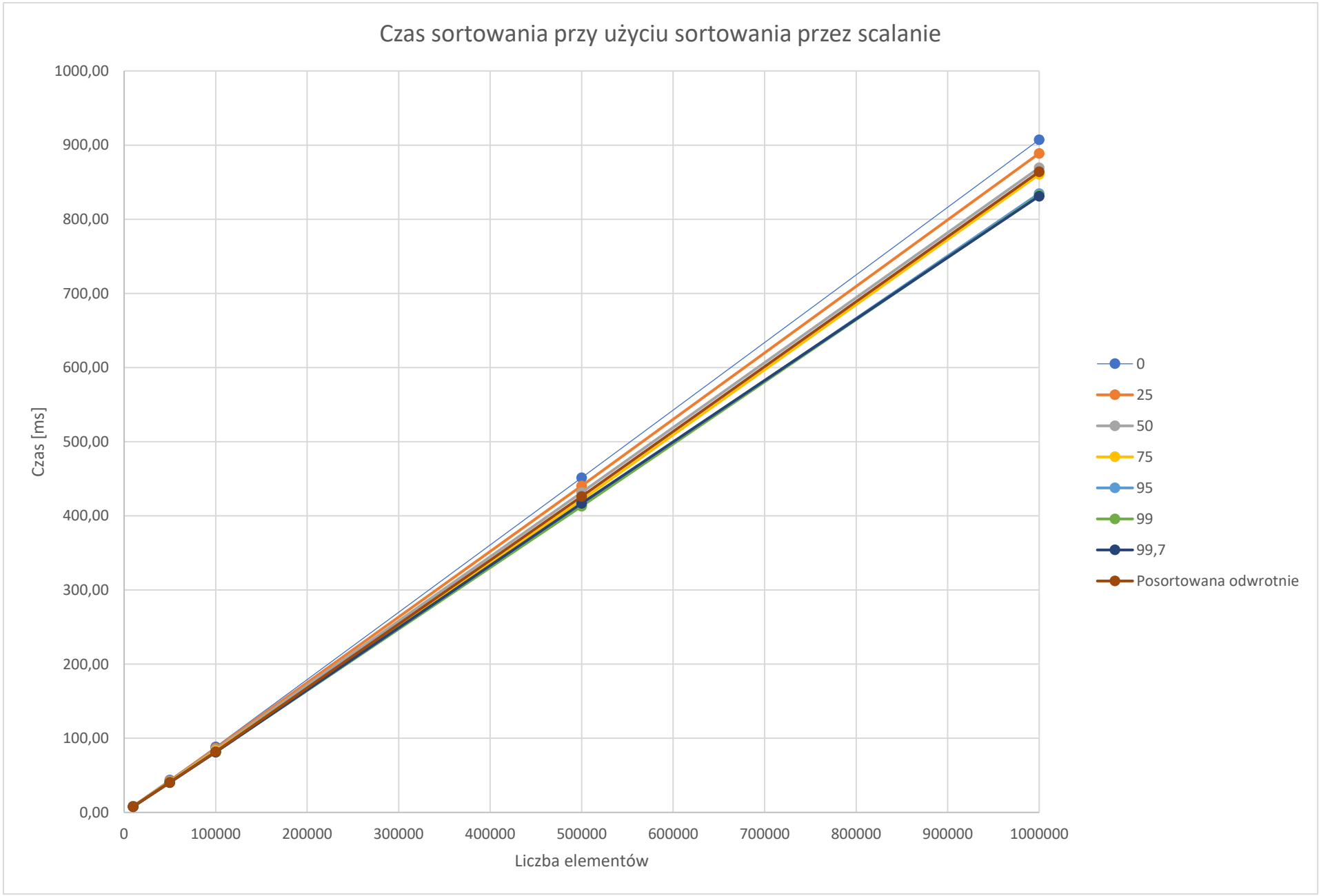
Michał Więcek
248956

b. Sortowanie przez scalanie

Posortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	8,43	43,46	87,85	448,14	904,48
	Średni	8,54	43,97	88,34	451,41	907,30
	Największy	9,40	56,09	89,86	526,29	924,29
25	Najmniejszy	8,29	42,70	86,18	438,93	885,82
	Średni	8,37	42,95	86,75	440,56	888,84
	Największy	8,67	43,77	88,55	444,15	939,39
50	Najmniejszy	8,15	41,89	84,43	429,53	866,90
	Średni	8,23	42,21	85,02	431,44	869,41
	Największy	8,82	42,76	88,79	438,20	875,70
75	Najmniejszy	7,99	40,99	82,55	419,51	847,70
	Średni	8,05	41,29	83,15	421,43	860,50
	Największy	8,29	42,42	85,49	428,49	1193,35
95	Najmniejszy	7,87	40,27	81,14	411,94	831,56
	Średni	7,96	40,58	81,60	413,64	834,74
	Największy	8,39	41,68	82,99	418,91	843,40
99	Najmniejszy	7,86	40,19	80,87	411,13	829,57
	Średni	7,92	40,92	81,65	413,03	832,30
	Największy	8,18	54,22	87,18	419,51	839,45
99,7	Najmniejszy	7,85	40,12	80,77	409,66	827,93
	Średni	7,94	40,40	81,30	416,83	831,24
	Największy	8,88	41,70	83,26	504,90	838,44
Posortowana odwrotnie	Najmniejszy	7,84	40,35	81,31	413,74	831,78
	Średni	7,91	40,63	81,87	426,08	864,12
	Największy	8,58	41,64	84,50	700,37	1690,38

Michał Więcek
248956

03.04.2020



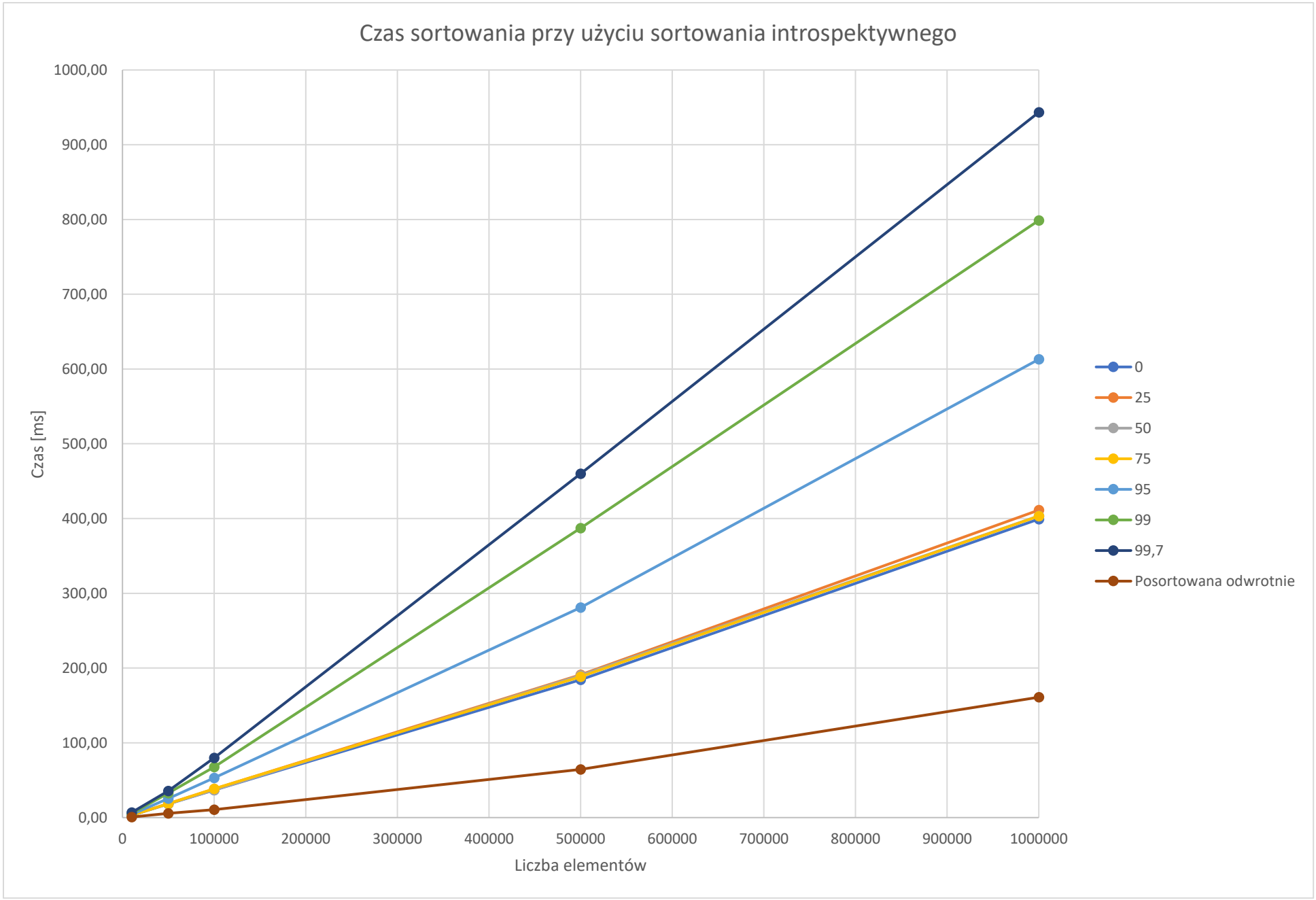
Michał Więcek
248956

c. Sortowanie introspektywne

osortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	2,68	16,76	34,12	168,36	357,94
	Średni	2,99	18,13	36,66	184,41	399,23
	Największy	3,52	20,26	42,61	213,26	585,79
25	Najmniejszy	2,77	16,48	34,07	171,33	360,93
	Średni	3,10	18,83	38,12	191,14	411,27
	Największy	4,46	21,65	51,40	232,60	539,72
50	Najmniejszy	2,36	16,30	32,13	162,10	347,83
	Średni	3,11	18,49	36,90	190,06	403,47
	Największy	5,09	23,67	48,05	308,08	652,37
75	Najmniejszy	2,29	12,90	28,08	149,74	313,76
	Średni	3,00	18,49	38,40	188,33	403,29
	Największy	7,94	48,17	100,21	489,10	1101,11
95	Najmniejszy	2,11	13,51	26,86	127,27	267,51
	Średni	4,14	25,39	53,22	280,87	612,95
	Największy	8,10	49,00	107,76	605,58	1283,41
99	Najmniejszy	2,45	13,91	26,94	130,76	291,33
	Średni	5,32	32,58	67,63	387,13	798,73
	Największy	8,50	50,00	107,72	602,74	1270,45
99,7	Najmniejszy	2,47	17,08	38,83	174,64	398,82
	Średni	6,80	35,69	79,93	459,85	943,38
	Największy	15,88	57,36	196,76	806,20	2056,51
Posortowana odwrotnie	Najmniejszy	0,70	5,48	10,46	64,16	160,00
	Średni	0,71	5,54	10,56	64,52	161,00
	Największy	0,82	6,37	10,85	66,16	169,14

Michał Więcek
248956

03.04.2020



6. Wnioski:

Z wykresów wynika, że złożoność każdego z algorytmów zawiera się w założonej, czyli $O(n \log n)$. W przypadku sortowania szybkiego wydajniejsze okazało się wybieranie pivota jako mediany z trzech liczb. Dzięki temu udało się wyeliminować pesymistyczny przypadek dla tablicy posortowanej w 50%. Sortowanie introspektywne swoje działanie opiera w większości na działaniu sortowania szybkiego, dlatego otrzymane wyniki są zbliżone do wyników sortowania szybkiego.

Można zauważyć, że sortowanie przez scalanie osiągnęło czasy bardzo zbliżone, niezależnie od stopnia w jakim posortowana została tablica. Wynika to z faktu, że dane wejściowe nie mają wpływu na ten algorytm.

Działanie na stworzonej przeze mnie strukturze danych `PaczkaTablic<T>` mimo, że wydawało mi się wygodne mogło zwiększać czas działania algorytmów poprzez potrzebę kilkokrotnego wywołania operatora dereferencji za każdym razem kiedy algorytm sortowania miał zacząć operować na nowej tablicy.

Ze względu na to, że funkcja `rand()` jest dość mocno ograniczona ponieważ jej maksymalna wartość może wynosić `32767` chciałem wypełniać tablice przy użyciu ciekawszego algorytmu Marsenne Twister. Jednak zaimplementowanie go zwiększyło czas generowania całego zestawu tablic z około 15 sekund na około 2 minuty więc zrezygnowałem z jego implementacji. Uznałem, że zwiększenie „losowości” liczb nie jest aż tak ważnym czynnikiem.

7. Literatura:

https://pl.wikipedia.org/wiki/Mersenne_Twister
https://pl.wikipedia.org/wiki/Sortowanie_szybkie
https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie
https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie
https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie
https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
http://www.cplusplus.com/reference/
https://www.geeksforgeeks.org/heap-sort/
https://www.geeksforgeeks.org/quick-sort/
https://www.youtube.com/watch?v=kPRA0W1kECg
https://pl.wikipedia.org/wiki/Generator_liczb_pseudolosowych