

# Projektowanie Algorytmów i Metody Sztucznej Inteligencji

## Projekt 1

Mgr inż. Marcin Ochman  
Grupa: Wtorek 15<sup>15</sup>-16<sup>55</sup>

### 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  $O(n \log n)$ , 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  $O(n \log n)$ . 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  $O(n \log n)$ .

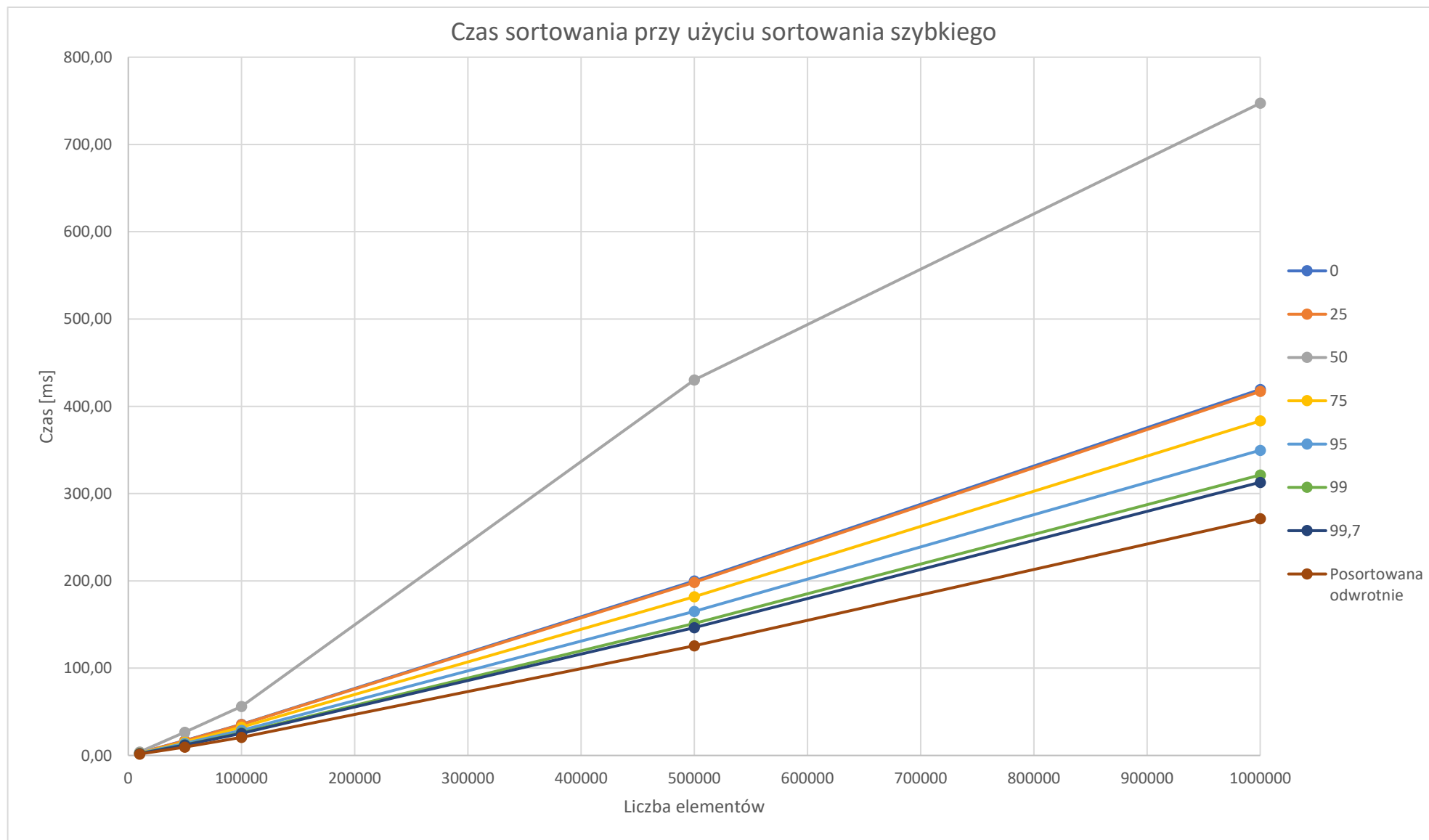
### 4. Sortowanie introspektywne

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

### a. Sortowanie szybkie

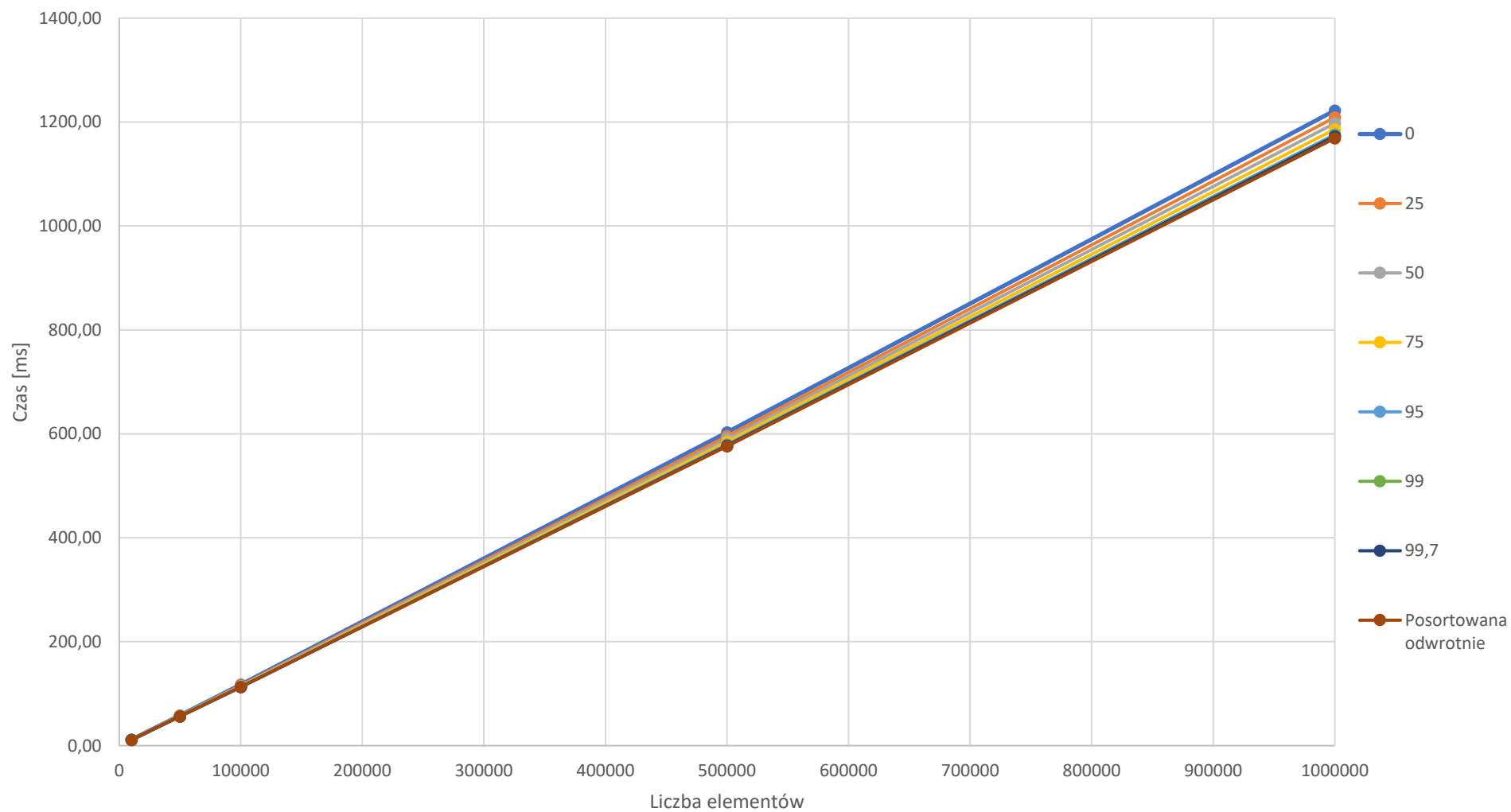
Posortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	2,95	16,71	35,26	197,77	415,81
	Średni	2,98	17,04	35,64	199,83	419,31
	Największy	3,05	20,09	37,34	207,83	429,37
25	Najmniejszy	2,94	16,57	34,90	195,19	411,42
	Średni	2,98	16,79	35,37	198,33	417,10
	Największy	3,04	17,40	36,97	202,50	432,49
50	Najmniejszy	2,94	16,06	34,66	194,97	409,24
	Średni	4,02	26,53	56,11	430,37	747,24
	Największy	7,62	55,41	178,94	1786,83	2838,08
75	Najmniejszy	2,64	15,03	31,82	179,74	379,02
	Średni	2,68	15,25	32,51	181,75	383,29
	Największy	3,08	15,85	35,08	186,91	399,35
95	Najmniejszy	2,36	13,52	28,68	164,00	347,02
	Średni	2,41	13,80	28,89	165,16	349,68
	Największy	2,74	14,67	29,35	169,90	363,36
99	Najmniejszy	2,12	12,19	25,93	150,19	320,21
	Średni	2,15	12,30	26,11	151,13	321,31
	Największy	2,20	12,70	27,02	156,44	333,06
99,7	Najmniejszy	2,01	11,68	25,04	145,94	311,74
	Średni	2,09	11,88	25,23	146,41	312,97
	Największy	2,15	12,21	26,01	147,23	320,39
Posortowana odwrotnie	Najmniejszy	1,67	9,61	20,72	125,15	270,41
	Średni	1,68	9,64	20,80	125,67	271,34
	Największy	1,72	9,87	21,24	129,57	277,66



## b. Sortowanie przez scalanie

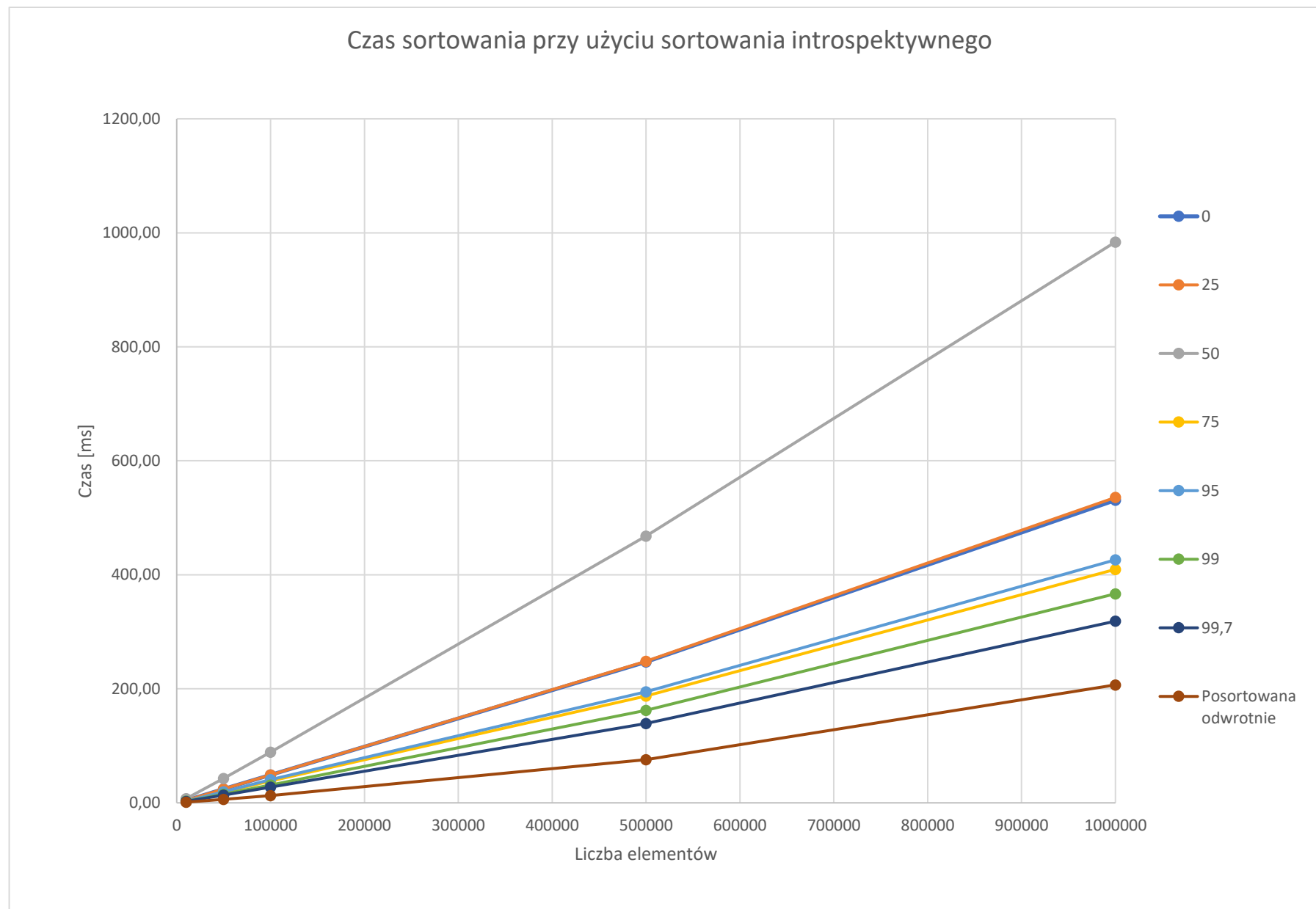
Posortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	11,21	57,69	116,40	595,66	1212,63
	Średni	11,52	58,32	117,62	602,68	1222,20
	Największy	15,03	60,81	119,49	612,42	1251,49
25	Najmniejszy	11,20	57,06	114,73	587,91	1197,63
	Średni	11,31	57,76	116,06	595,49	1208,78
	Największy	12,34	58,62	118,57	605,69	1221,52
50	Najmniejszy	11,06	56,50	114,10	582,86	1187,88
	Średni	11,18	57,22	114,93	589,96	1197,49
	Największy	11,32	58,76	116,69	598,59	1211,26
75	Najmniejszy	10,98	55,83	112,13	576,53	1176,37
	Średni	11,10	56,53	113,60	584,62	1186,12
	Największy	11,31	57,63	116,03	596,64	1213,92
95	Najmniejszy	10,91	55,44	111,67	571,14	1158,43
	Średni	10,99	56,10	112,91	579,24	1176,02
	Największy	11,10	57,40	115,43	593,64	1188,88
99	Najmniejszy	10,90	55,33	111,96	569,85	1157,26
	Średni	11,00	56,15	112,86	577,48	1173,53
	Największy	11,07	57,80	114,72	587,00	1191,08
99,7	Najmniejszy	10,87	55,49	111,60	569,65	1162,41
	Średni	10,97	55,99	112,57	577,35	1172,65
	Największy	11,10	56,65	114,65	586,60	1184,79
Posortowana odwrotnie	Najmniejszy	10,89	55,32	111,96	568,52	1159,11
	Średni	11,04	56,07	112,85	576,19	1168,66
	Największy	11,25	57,12	114,62	582,23	1199,19

### Czas sortowania przy użyciu sortowania przez scalanie



### c. Sortowanie introspektywne

Posortowane w %	Czas [ms]	Liczba elementów				
		10000	50000	100000	500000	1000000
0	Najmniejszy	3,69	21,83	44,90	225,58	485,86
	Średni	4,11	24,39	48,98	247,12	530,93
	Największy	4,80	29,82	53,98	294,61	634,94
25	Najmniejszy	3,64	21,69	44,07	218,10	477,28
	Średni	4,07	24,08	48,94	248,41	535,76
	Największy	4,90	28,87	62,52	289,12	625,94
50	Najmniejszy	3,84	24,57	45,67	225,29	499,78
	Średni	7,28	42,87	89,03	467,92	983,96
	Największy	10,30	60,88	129,21	773,74	1652,08
75	Najmniejszy	2,93	17,68	35,74	178,64	389,59
	Średni	3,13	18,92	38,18	187,28	409,64
	Największy	3,46	20,92	41,23	198,79	446,97
95	Najmniejszy	2,76	17,65	37,04	184,56	402,22
	Średni	3,06	19,38	40,58	194,71	426,31
	Największy	3,33	21,05	43,46	206,68	452,88
99	Najmniejszy	2,50	14,01	29,20	158,23	358,66
	Średni	2,66	15,18	31,37	162,35	366,78
	Największy	2,80	15,98	34,19	168,96	377,22
99,7	Najmniejszy	2,30	13,19	26,62	134,66	310,41
	Średni	2,55	13,57	27,52	139,23	318,70
	Największy	2,74	14,19	28,41	142,70	325,10
Posortowana odwrotnie	Najmniejszy	1,17	6,17	12,45	75,15	206,02
	Średni	1,18	6,20	12,53	75,60	206,91
	Największy	1,20	6,26	13,77	76,60	209,51



## 6. Wnioski:

W przeprowadzonym przeze mnie eksperymencie sortowanie szybkie traci na wydajności dla tablicy posortowanej w 50% dzieje się tak ponieważ w mojej implementacji oś wybierana jest jako środkowy element tablicy. Kiedy tablica jest w połowie posortowana to element środkowy jest najprawdopodobniej elementem największym lub bliskim największemu.

Można zauważyć, że sortowanie przez scalanie osiągnęło czasy bardzo zbliżone, niezależnie od stopnia w jakim posortowana została tablica.

Sortowanie introspektywne swoje działanie opiera w większości na działaniu sortowania szybkiego, dlatego otrzymane wyniki są zbliżone do wyników sortowania szybkiego.

Głównym błędem jaki popełniłem w czasie tworzenia implementacji sortowania introspektywnego mogło być obiektowe podejście do problemu, powodem czego za każdym razem kiedy sortowanie introspektywne „przełącza” się na sortowanie przez wstawianie lub sortowanie przez kopcowanie określona ilość czasu może zostać poświęcona na utworzenie i dekonstrukcję obiektu.

Biorąc pod uwagę podobieństwa w implementacji tych trzech sposobów sortowania można zoptymalizować kod rezygnując z tworzenia klas.

Działanie wewnątrz stworzonej przeze mnie struktury 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/Mersenne_Twister)  
[https://pl.wikipedia.org/wiki/Sortowanie\\_szybkie](https://pl.wikipedia.org/wiki/Sortowanie_szybkie)  
[https://pl.wikipedia.org/wiki/Sortowanie\\_przez\\_kopcowanie](https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie)  
[https://pl.wikipedia.org/wiki/Sortowanie\\_przez\\_scalanie](https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie)  
[https://pl.wikipedia.org/wiki/Sortowanie\\_przez\\_wstawianie](https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie)  
[https://pl.wikipedia.org/wiki/Sortowanie\\_introspektywne](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](https://pl.wikipedia.org/wiki/Generator_liczb_pseudolosowych)