Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Filip Kaśkos 249009

14 kwietnia 2020

Prowadzący: Mgr inż. Marcin Ochman

Termin: wtorek 15.15

1.Wprowadzenie

Zadanie polegało na zaimplementowaniu i przetestowaniu trzech algorytmów sortowania. Algorytmy zostały zaimplementowane jako szablony. Przeprowadzenie testów polegało na zmierzeniu czasu sortowania w poszczególnych przypadkach. Testom zostały poddane tablice zawierające liczby całkowite. Algorytmy były użyte do posortowania 100 tablic danego rozmiaru(10000,50000,100000,500000,1000000). Dla każdego przypadku należało uwzględnić:

- wszystkie elementy tablicy losowe,

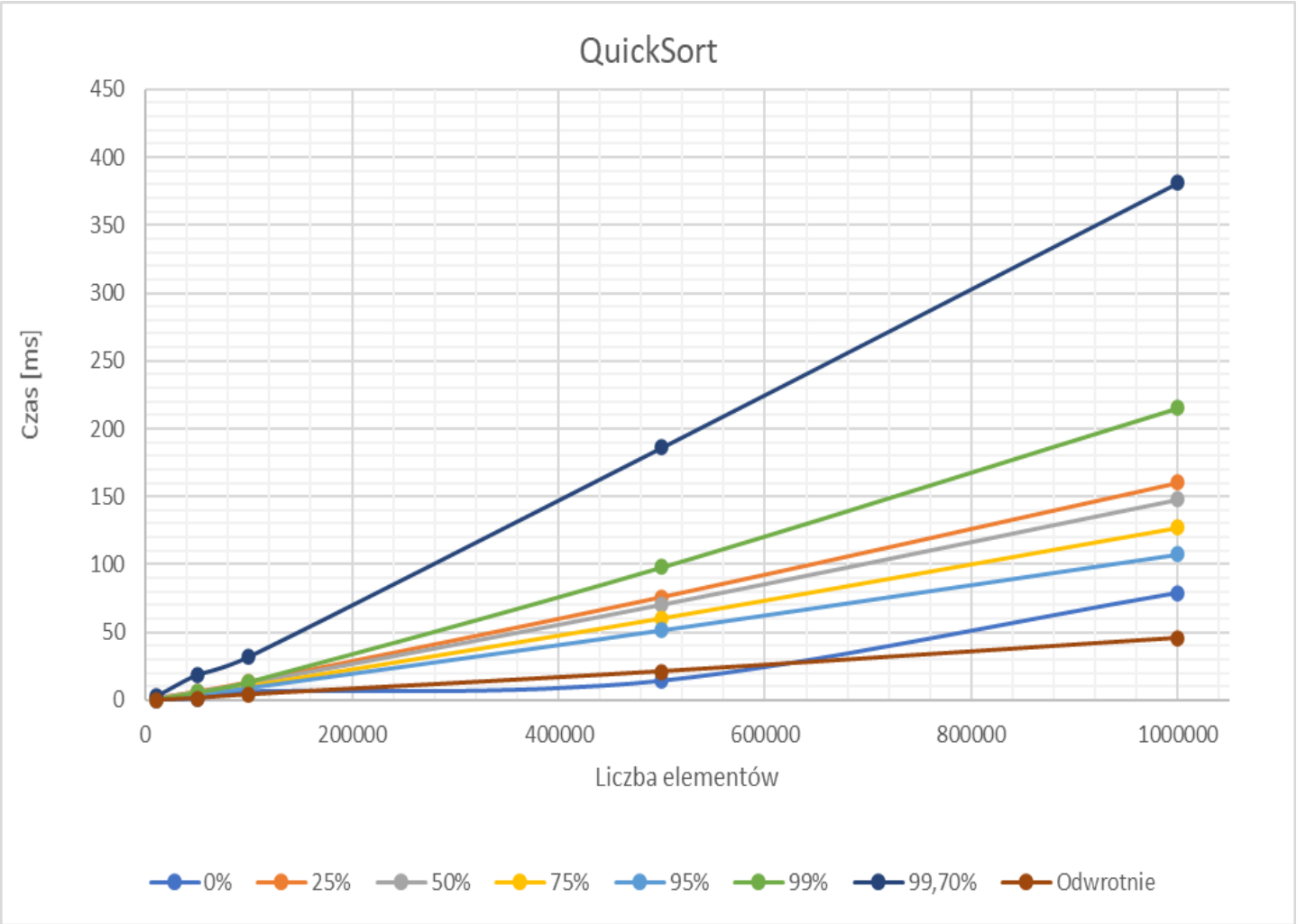
- elementy posortowane w pewnej części (0%,25%,50%,75%,95%,99%,99.7%)

- elementy posortowane odwrotnie(od największego do najmniejszego)

Wybrane przeze mnie algorytmy są algorytmami z grupy szybkich. Ich złożoność obliczeniowa w większości przypadków wynosi O(n*logn).

2.QuickSort-sortowanie szybkie

Jest to algorytm który wykorzystuje technikę „Dziel i zwyciężaj”. Jest to algorytm działający rekurencyjnie. Zasada jego działania opiera się na wybraniu elementu rozdzielającego tablicę na dwie strony. W lewej stronie znajdują się elementy mniejsze od rozdzielającego, a z prawej strony większe od niego. Następnie każdą z nich sortujemy osobno tym samym schematem. Dzięki tym operacją uzyskujemy afekt końcowy jakim jest w pełni posortowana tablica. Złożoność tego algorytmu dla przypadku średniego to O(n*logn) jednak istnieje przypadek najgorszy, w którym algorytm ten osiąga złożoność $O(n^2)$.



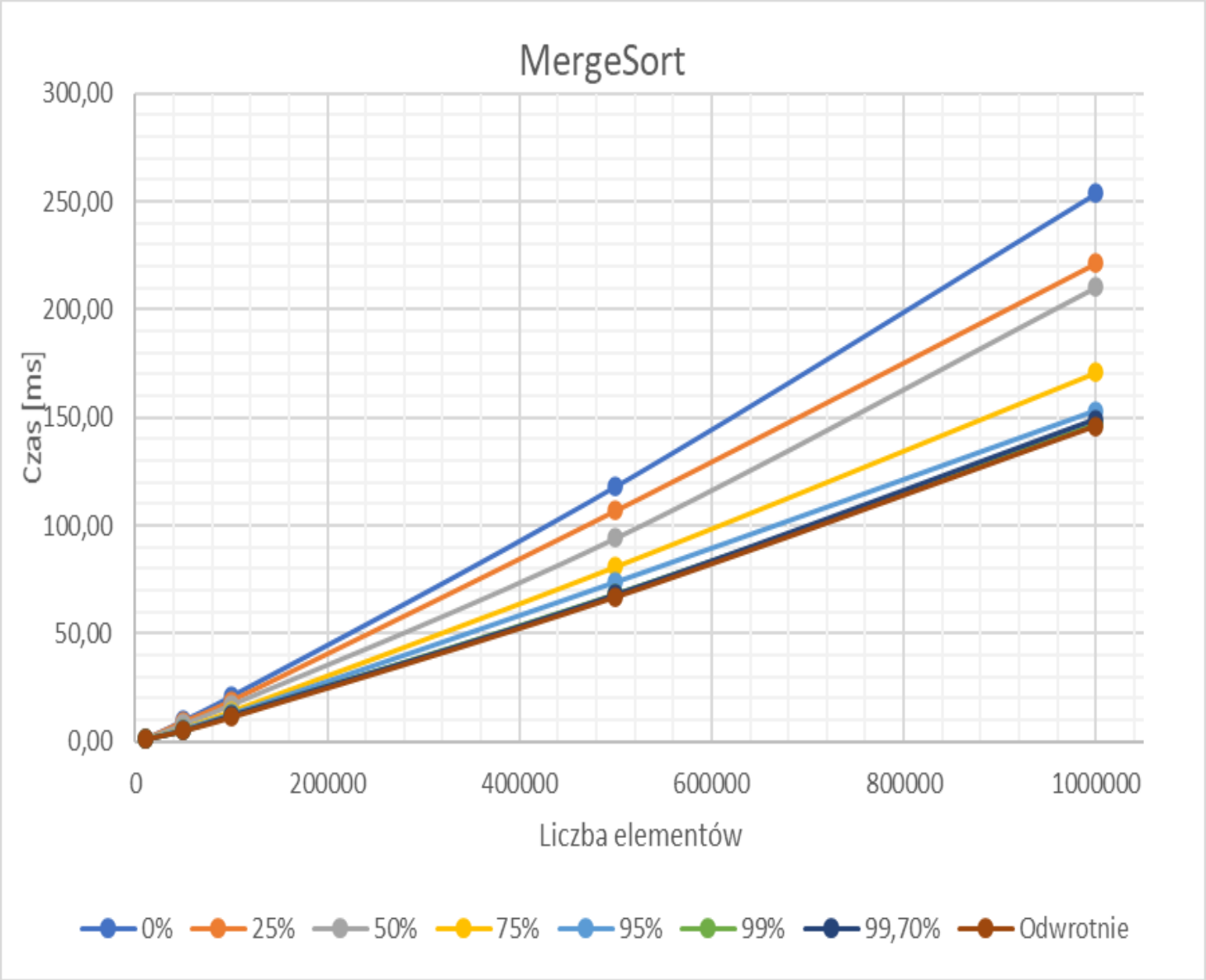
Wykres 1.Wykres przedstawiający zależność czasu od ilości elementów oraz % ich posortowania

Tabela 1.Wyniki przeprowadzanych testów dla Quicksorta

Procent posortowania	Liczba Elementów	10000	50000	100000	500000	1000000
0	Min [ms]	1,00	6,00	13,00	78,00	164,00
	Max [ms]	1,00	6,00	14,00	82,00	169,00
	Średni [ms]	1,00	6,00	13,98	78,76	165,31
25	Min [ms]	1,00	6,00	13,00	74,00	157,00
	Max [ms]	1,00	6,00	13,00	78,00	175,00
	Średni [ms]	1,00	6,00	13,00	76,00	160,52
50	Min [ms]	1,00	5,00	12,00	69,00	145,00
	Max [ms]	1,00	7,00	13,00	73,00	152,00
	Średni [ms]	1,00	5,79	12,04	70,34	148,06
75	Min [ms]	0,00	4,00	10,00	59,00	123,00
	Max [ms]	1,00	5,00	14,00	68,00	146,00
	Średni [ms]	0,03	4,99	10,16	60,32	127,28
95	Min [ms]	0,00	3,00	7,00	44,00	94,00
	Max [ms]	1,00	6,00	12,00	81,00	165,00
	Średni [ms]	0,02	3,87	8,67	51,40	107,54
99	Min [ms]	0,00	3,00	8,00	60,00	134,00
	Max [ms]	4,00	22,00	28,00	296,00	714,00
	Średni [ms]	0,67	5,97	13,61	98,14	215,46
99.7	Min [ms]	1,00	6,00	10,00	77,00	142,00
	Max [ms]	10,00	90,00	230,00	502,00	1416,00
	Średni [ms]	2,52	18,62	31,91	186,07	381,41
Odwrotnie	Min [ms]	0,00	1,00	3,00	20,00	43,00
	Max [ms]	0,00	2,00	4,00	22,00	54,00
	Średni [ms]	0,00	1,33	3,99	20,99	45,56

3.Mergesort- Sortowanie przez scalanie

Podobnie do quicksorta jest to algorytm rekurencyjny wykorzystujący technikę „Dziel i zwyciężaj”. Sortowanie przez scalanie polega na dzieleniu zbioru na dwa równe podzbiory. Dzielenie odbywa się do momentu aż nie uzyskamy zbioru jednoelementowego, który jest posortowany. Następnie łączymy podzbiory, aż do momentu uzyskania w pełni posortowanego zbioru. Jego złożoność obliczeniowa wynosi O(n*logn). Potrzebuje on jednak pomocniczej struktury danych co wpływa na zwiększenie złożoności pamięciowej.



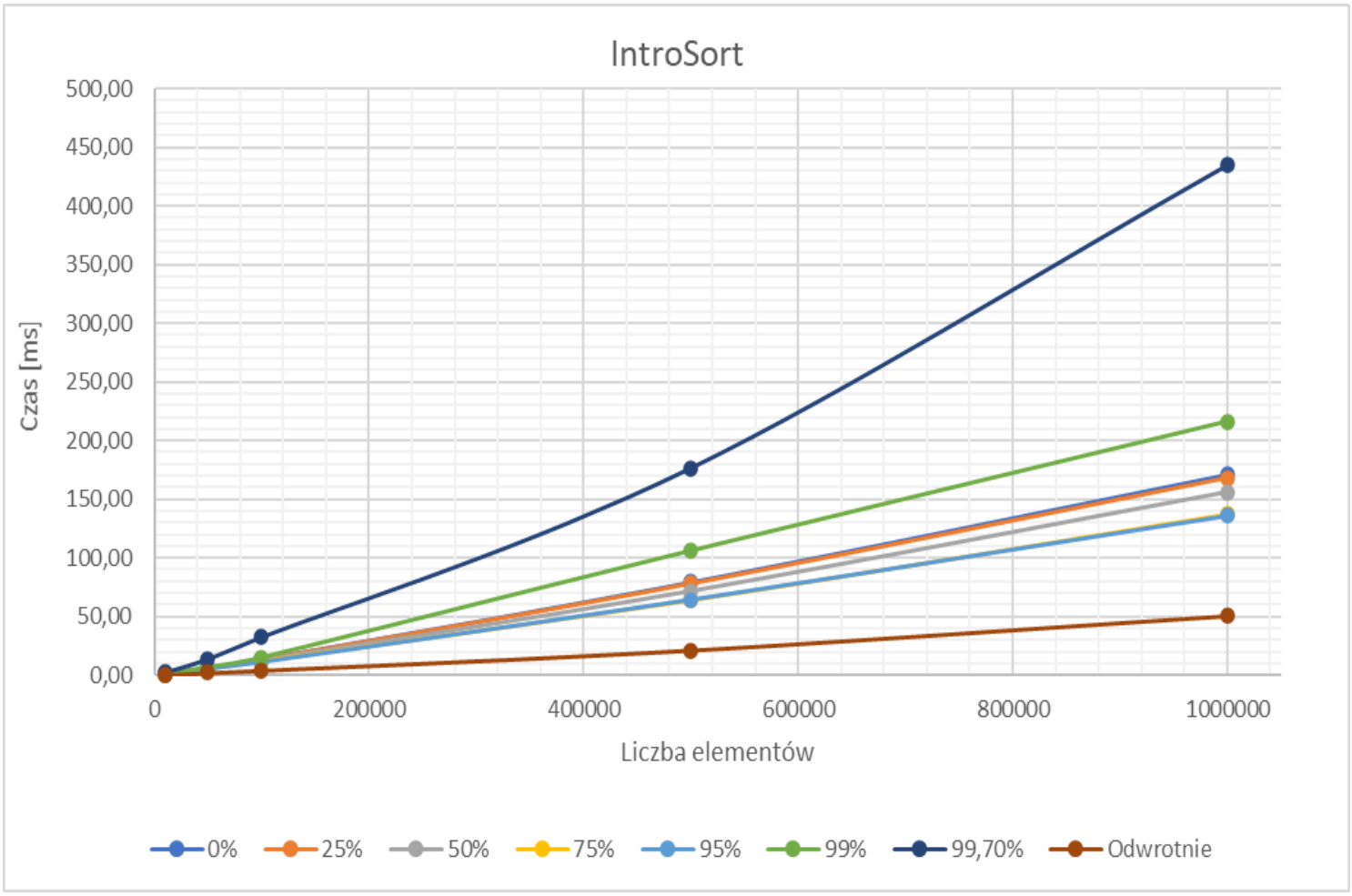
Wykres 2. Wykres zależności czasu sortowania od liczby elementów oraz % posortowania

Tabela 2.Wyniki przeprowadzanych testów dla Mergesorta

Procent posortowania	Liczba Elementów	10000	50000	100000	500000	1000000
0	Min [ms]	1,00	9,00	21,00	118,00	244,00
	Max [ms]	2,00	13,00	21,00	119,00	384,00
	Średni [ms]	1,01	9,72	21,00	118,15	253,73
25	Min [ms]	1,00	9,00	19,00	106,00	221,00
	Max [ms]	1,00	13,00	19,00	108,00	225,00
	Średni [ms]	1,00	9,27	19,00	107,03	221,50
50	Min [ms]	1,00	8,00	17,00	94,00	195,00
	Max [ms]	1,00	8,00	17,00	95,00	390,00
	Średni [ms]	1,00	8,00	17,00	94,19	210,24
75	Min [ms]	1,00	6,00	14,00	80,00	167,00
	Max [ms]	1,00	7,00	15,00	84,00	261,00
	Średni [ms]	1,00	6,23	14,01	80,96	170,74
95	Min [ms]	1,00	6,00	12,00	70,00	145,00
	Max [ms]	1,00	6,00	13,00	115,00	263,00
	Średni [ms]	1,00	6,00	12,79	73,72	152,91
99	Min [ms]	1,00	5,00	12,00	68,00	142,00
	Max [ms]	1,00	6,00	13,00	70,00	226,00
	Średni [ms]	1,00	5,13	12,01	68,44	147,35
99.7	Min [ms]	1,00	5,00	12,00	68,00	141,00
	Max [ms]	1,00	6,00	13,00	71,00	223,00
	Średni [ms]	1,00	5,11	12,09	68,03	149,20
Odwrotnie	Min [ms]	1,00	5,00	11,00	65,00	136,00
	Max [ms]	1,00	6,00	12,00	80,00	229,00
	Średni [ms]	1,00	5,05	11,24	66,90	145,93

4.Introsort-sortowanie introspektywne

Jest to hybrydowy algorytm sortowania. Składa się on z kilku algorytmów wykorzystywanych jako jedna całość. Sortowanie introspektywne wykorzystuje głównie quicksorta, ale eliminuje ono jego najgorszy przypadek. Podczas działania algorytmu ustalana jest maksymalna głębokość wywołań rekurencyjnych. Jeżeli głębokość osiągnie 0 wywołania rekurencyjne kończą się a podzbiór sortowany jest przez sortowanie przez kopcowanie. Następnie jeżeli jego rozmiar jest mniejszy od pewnego założonego wykonuje się sortowanie przez wstawianie, które mimo dużej złożoności obliczeniowej dla małych zbiorów jest bardzo efektywne. Introsort wykorzystuje więc quicksort, sortowanie przez kopcowanie(Heapsort) oraz sortowanie przez wstawianie(Insertion Sort). Jego złożoność obliczeniowa to $O(n \cdot \log_2 n)$.



Wykres 3.Wykres przedstawia zależność czasu od liczby elementów oraz % ich posortowania

Tabela 3.Wyniki przeprowadzonych testów dla Introsorta

Procent posortowania	Liczba Elementów	10000	50000	100000	500000	1000000
0	Min [ms]	1,00	6,00	13,00	77,00	166,00
	Max [ms]	1,00	8,00	14,00	89,00	181,00
	Średni [ms]	1,00	6,37	13,52	79,12	170,78
25	Min [ms]	1,00	6,00	13,00	74,00	161,00
	Max [ms]	1,00	7,00	15,00	90,00	184,00
	Średni [ms]	1,00	6,05	13,28	77,97	167,83
50	Min [ms]	0,00	5,00	11,00	68,00	147,00
	Max [ms]	1,00	8,00	15,00	81,00	171,00
	Średni [ms]	0,52	5,86	12,50	71,61	156,15
75	Min [ms]	0,00	4,00	10,00	57,00	125,00
	Max [ms]	1,00	7,00	15,00	93,00	162,00
	Średni [ms]	0,17	4,89	11,34	63,73	137,34
95	Min [ms]	0,00	4,00	9,00	54,00	120,00
	Max [ms]	1,00	7,00	15,00	86,00	188,00
	Średni [ms]	0,15	4,85	10,66	64,24	136,23
99	Min [ms]	0,00	4,00	11,00	76,00	157,00
	Max [ms]	4,00	17,00	39,00	299,00	507,00
	Średni [ms]	0,69	6,86	15,02	105,99	216,30
99.7	Min [ms]	1,00	5,00	12,00	86,00	209,00
	Max [ms]	9,00	62,00	184,00	638,00	1780,00
	Średni [ms]	2,10	13,52	32,09	176,44	434,73
Odwrotnie	Min [ms]	0,00	1,00	3,00	19,00	45,00
	Max [ms]	0,00	2,00	4,00	24,00	57,00
	Średni [ms]	0,00	1,98	3,98	21,18	50,31

5.Podsumowanie

Uzyskane wyniki zaprezentowane na wykresach oraz w tabelach są w większości zgodne z oczekiwanymi. Algorytmy spełniają założoną złożoność obliczeniową. Widać jednak, że quicksort oraz introsort uzyskują dłuższe czasy sortowania powyżej wartośći 95% posortowanej wcześniej tablicy. Poniżej tej wartości czasy maleją proporcjonalnie do zwiększonej ilości posortowanych elementów. Jasne jest, iż podobne wyniki wynikają z tego, że introsort korzysta przede wszystkim z quicksorta. Nie potrafię jednak znaleźć przyczyny problemu zwiększającego się czasu sortowania dla tablicy posortowanej wcześniej w 99 i 99.7%.