Algorytmy i złożoność obliczeniowa – projekt 1

Badanie efektywności wybranych algorytmów sortowania ze względu na złożoność obliczeniową

Autor: Jakub Smolarczyk

Indeks: 272924

Grupa 18 – termin: czw. parzysty 11:15

Prowadzący: Dariusz Banasiak

Spis treści:

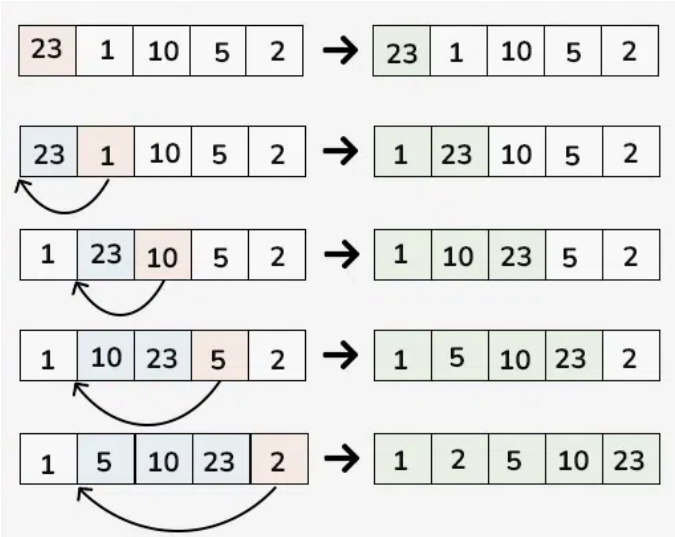
1. Wprowadzenie

Sortowanie jest fundamentalnym zagadnieniem w informatyce. Polega ono na uporządkowaniu określonego zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru. Istnieje wiele różnych algorytmów sortowania o różnej efektywności. Wpływ na nią mają takie czynniki jak np.: złożoność obliczeniowa rozpatrywanego algorytmu oraz stopień wstępnego posortowania zbioru danych.

W tej pracy przedstawione zostaną wyniki badań efektywności wybranych algorytmów sortowania z uwzględnieniem podanych wyżej czynników. Badania obejmują następujące algorytmy sortowania:

- Sortowanie przez wstawianie

  Jeden z najprostszych algorytmów sortowania. Polega on na porównywaniu każdego elementu z poprzednimi elementami zbioru danych. W przypadku chęci uzyskania ciągu niemalejącego proces ten jest kontynuowany dopóki nie zostanie znaleziona wartość mniejsza lub równa wybranemu elementowi albo nie zostanie osiągnięty początek zbioru, po czym wybrany element jest wstawiany w miejsce, gdzie zakończono porównywanie. Proces ten ilustruje rysunek przedstawiony obok.



Rys. 1 Przykładowy zbiór danych posortowany przez wstawianie

  Złożoność obliczeniowa tego algorytmu zarówno dla przypadku średniego jak i pesymistycznego wynosi $O(n^2)$, co w teorii czyni go nieefektywnym dla dużych zbiorów danych.

- Sortowanie przez kopcowanie

  Jest przykładem algorytmu niestabilnego, lecz szybkiego (zarówno dla przypadku pesymistycznego jak i średniego złożoność obliczeniowa wynosi $O(n \log n)$. Przed rozpoczęciem właściwego sortowania zbiór danych jest układany w kopiec maksymalny (dla sortowania niemalejącego). Potem powtarzany jest proces zamiany elementu pierwszego (korzenia) z ostatnim elementem, usunięciu z kopca przemieszczonego korzenia, a następnie naprawie kopca aż do wyczerpania się elementów w kopcu.

- Sortowanie Shella

  Sortowanie Shella można traktować jako inną wersję sortowania przez wstawianie, tylko że w tym przypadku porównywane są elementy co zadany przedział a nie sąsiednie elementy. Pomimo tego podobieństwa jest to przykład algorytmu niestabilnego tak samo jak sortowanie przez kopcowanie. Algorytm sortowania Shella wyróżnia to, że jego złożoność obliczeniowa jest różna w zależności od wybranych odstępów w wykonywanym algorytmie. W tej pracy rozpatrywane jest sortowanie Shella z następującymi odstępami:

  > $\left\lfloor \frac{N}{2^k} \right\rfloor$ ($\left\lfloor \frac{N}{2} \right\rfloor$, $\left\lfloor \frac{N}{4} \right\rfloor$, $\left\lfloor \frac{N}{8} \right\rfloor$ itd.) o pesymistycznej złożoności obliczeniowej $\theta(n^2)$

  > $2^k - 1$ (Hibbarda) o pesymistycznej złożoności obliczeniowej $\theta(n^{\frac{3}{2}})$

  Poniżej przedstawiony jest przykład sortowania Shella z odstępami $\left\lfloor \frac{N}{2^k} \right\rfloor$. Dla tablicy 8 elementowej wykonane jest najpierw sortowanie z odstępami o wartości 4. W dalszej kolejności wykonane byłoby sortowanie z odstępami o wartości 2 i 1.

	Podział, h=4								Sortowanie		Wynik							
	4	2	9	5	6	3	8	1	*- Zbiór wejściowy*									
1	4				6				4	6	4				6			
2		2				3			2	3		2				3		
3			9				8		8	9			8				9	
4				5				1	1	5				1				5
									Zbiór wyjściowy -		4	2	8	1	6	3	9	5

Rys. 2 Przykład działania sortowania Shella

- Sortowanie szybkie

  Jeden z popularnych algorytmów sortowania działających na zasadzie „dziel i zwyciężaj”. Sortowanie to opiera się na wyborze elementu rozdzielającego (zwanego dalej pivotem) i podziale zbioru danych na 2 mniejsze: z wartościami mniejszymi lub równymi wartości pivota, przemieszczonymi do pierwszej połowy zbioru oraz z wartościami większymi od pivota, przemieszczonymi do drugiej połowy zbioru. Na powstałych zbiorach powtarza się ten sam krok aż do uzyskania jednoelementowych zbiorów danych (dane będą wtedy już posortowane).

  Złożoność tego algorytmu może zależeć od sposobu wyboru pivota. W przypadku średnim złożoność obliczeniowa tego algorytmu wynosi $O(n \log n)$, najczęściej przy losowym wyborze pivota. W przypadku pesymistycznym złożoność obliczeniowa wynosi aż $O(n^2)$ i zachodzi to najczęściej w przypadku wyboru pivota na skrajnej pozycji, w większości posortowanym ciągu danych.

2. Plan oraz przebieg eksperymentu

Dla każdego wymienionego wyżej algorytmu sortowania zostało wykonane 100 powtórzeń dla każdej z 7 tablic o różnych rozmiarach, które zostały dobrane eksperymentalnie. Rozmiary tablic są następujące:

10000, 20000, 30000, 40000, 60000, 80000, 100000 elementów

W każdym przypadku tworzona dynamicznie tablica była wypełniana losowymi elementami o wartości w zakresie [1, 2000000]. Proces sortowania odbywał się z użyciem kopii oryginalnej tablicy, skopiowanej za pomocą **std::copy()**, wstępnie posortowanej częściowo, rosnąco, malejąco lub też wcale, w zależności od wyboru. W dalszym kroku wykonywany był odpowiedni algorytm sortowania w przeznaczonej do tego funkcji, poprzedzony bezpośrednio wywołaniem **std::chrono::high_resolution_clock::now()**, które zostało wywołane ponownie tuż po zakończeniu funkcji z algorytmem sortowania. Różnica wartości z obu tych wywołań zrzutowana na odpowiednią jednostkę czasu zwracała czas trwania algorytmu sortowania. Z uzyskanych wyników obliczona została średnia dla każdej kombinacji dostępnych rozmiarów tablicy i stopnia wstępnego posortowania, zapisana w pliku tekstowym „output.txt”. Proces ten był powtarzany dla każdego sprawdzanego algorytmu sortowania.

W programie została uwzględniona również możliwość sprawdzenia poprawności wykonania algorytmów sortowania. W tym celu dla małych tablic dodana została opcja wyświetlenia utworzonej tablicy (niesortowanej) oraz jej posortowanej kopii. Dla większych tablic poprawność była sprawdzana poprzez porównanie każdego elementu posortowanej danym algorytmem tablicy z kopią oryginalnej tablicy posortowaną z użyciem **std::sort()**.

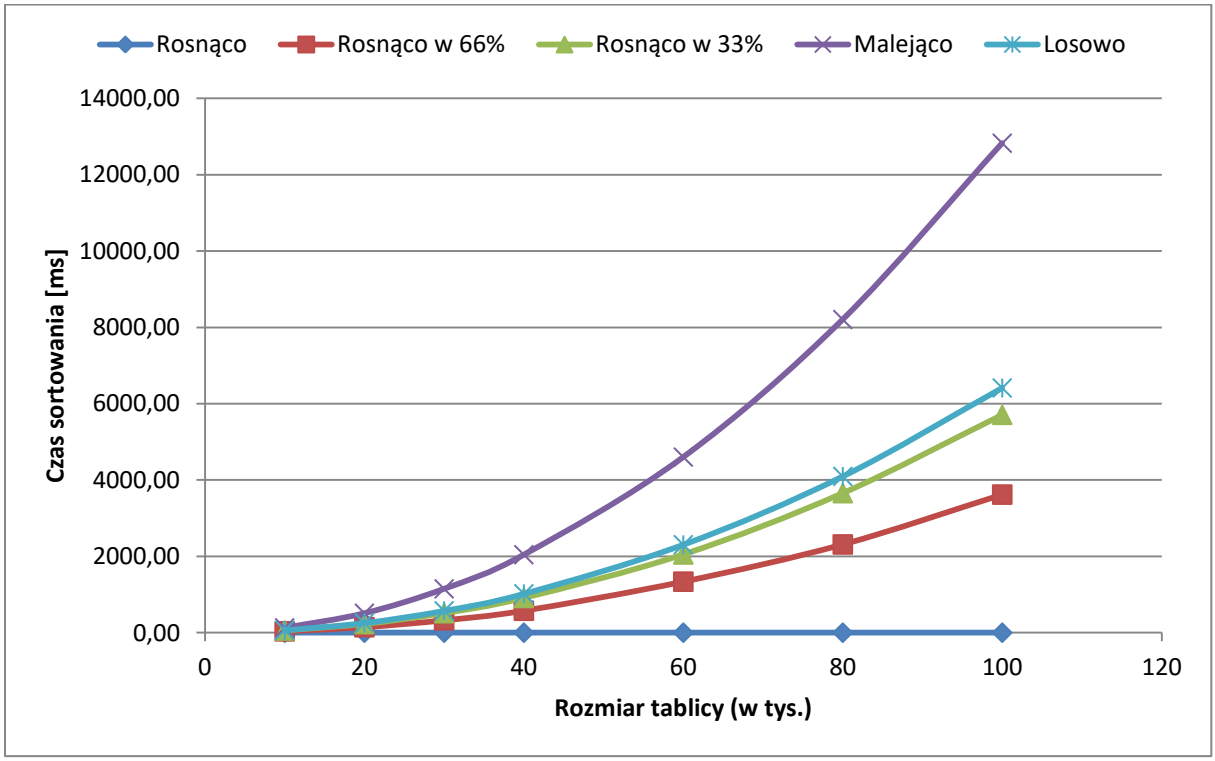
3. Wyniki eksperymentu

- Sortowanie przez wstawianie

Ze względu na swoją złożoność obliczeniową, ten algorytm był najbardziej czasochłonny. Praktycznie dla każdego przypadku, poza optymistycznym, średni czas sortowania dla tablicy 100 tysięcy elementów wyniósł kilka sekund lub więcej. W przypadku optymistycznym algorytm okazał się na tyle szybkim, że czas sortowania wyniósł mniej niż 1 ms. Wyniki wykonanych pomiarów zamieszczone są w poniższej tabeli oraz na wykresie.

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	0,029	36,24	58,26	127,72	65,82
20	0,059	144,19	227,52	510,19	255,69
30	0,091	325,04	512,51	1149,99	574,88
40	0,15	576,18	910,39	2044,14	1021,68
60	0,23	1337,44	2050,41	4602,24	2306,83
80	0,29	2312,58	3657,07	8205,89	4096,12
100	0,37	3619,48	5709,78	12825,00	6411,91

Sortowanie na danych typu int



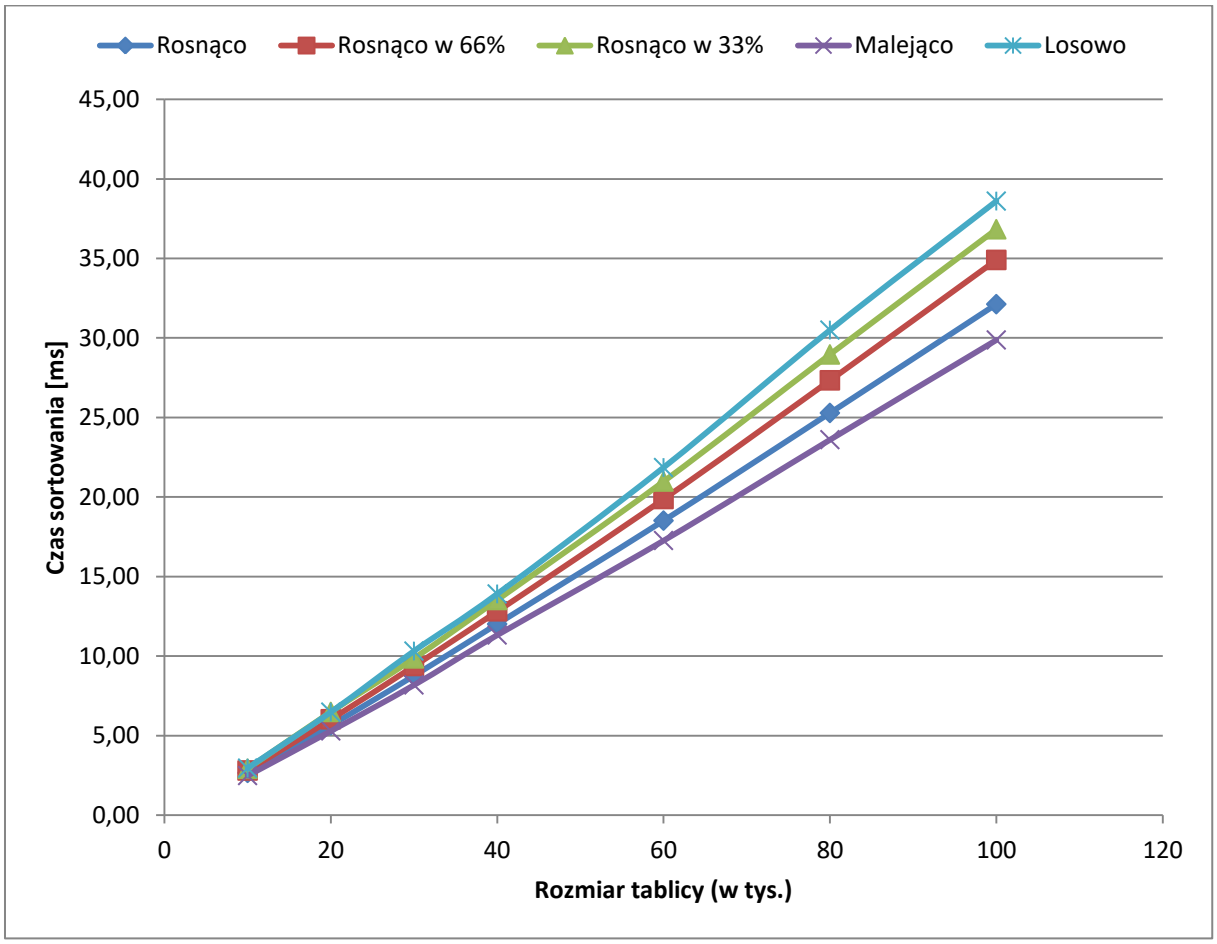
Rys. 3 Czas sortowania w funkcji rozmiaru tablicy dla sortowania przez wstawianie dla danych typu int

- Sortowanie przez kopcowanie

W sortowaniu przez kopcowanie zauważono większą średnią efektywność niż w przypadku sortowania przez wstawianie. Z tego powodu sortowanie przez kopcowanie zostało wybrane do sprawdzenia wpływu typu danych na czas sortowania. Wyniki wykonanych pomiarów zamieszczone są w poniższych tabelach oraz na wykresach.

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	2,65	2,80	2,91	2,48	2,95
20	5,64	6,02	6,48	5,29	6,47
30	8,76	9,37	9,85	8,18	10,31
40	12,01	12,81	13,52	11,32	13,90
60	18,51	19,87	20,97	17,26	21,85
80	25,28	27,33	28,96	23,60	30,49
100	32,12	34,90	36,83	29,87	38,60

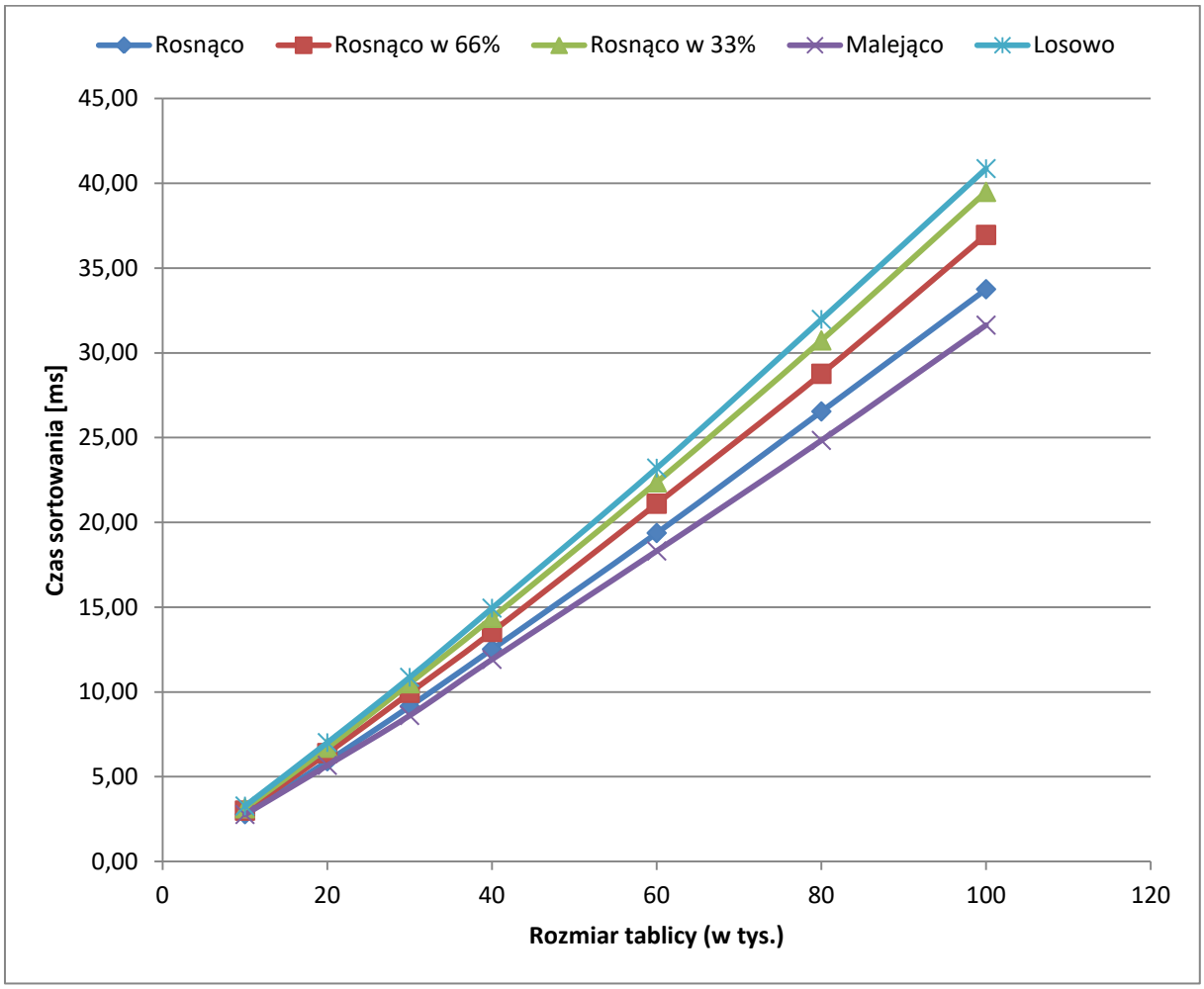
Sortowanie na danych typu int



Rys. 4 Czas sortowania w funkcji rozmiaru tablicy dla sortowania przez kopcowanie dla danych typu int

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	2,79	3,00	3,14	2,77	3,27
20	5,91	6,39	6,71	5,67	7,01
30	9,15	9,95	10,51	8,61	10,86
40	12,52	13,54	14,37	11,92	14,94
60	19,37	21,10	22,36	18,31	23,20
80	26,54	28,76	30,73	24,83	31,97
100	33,75	36,95	39,49	31,63	40,86

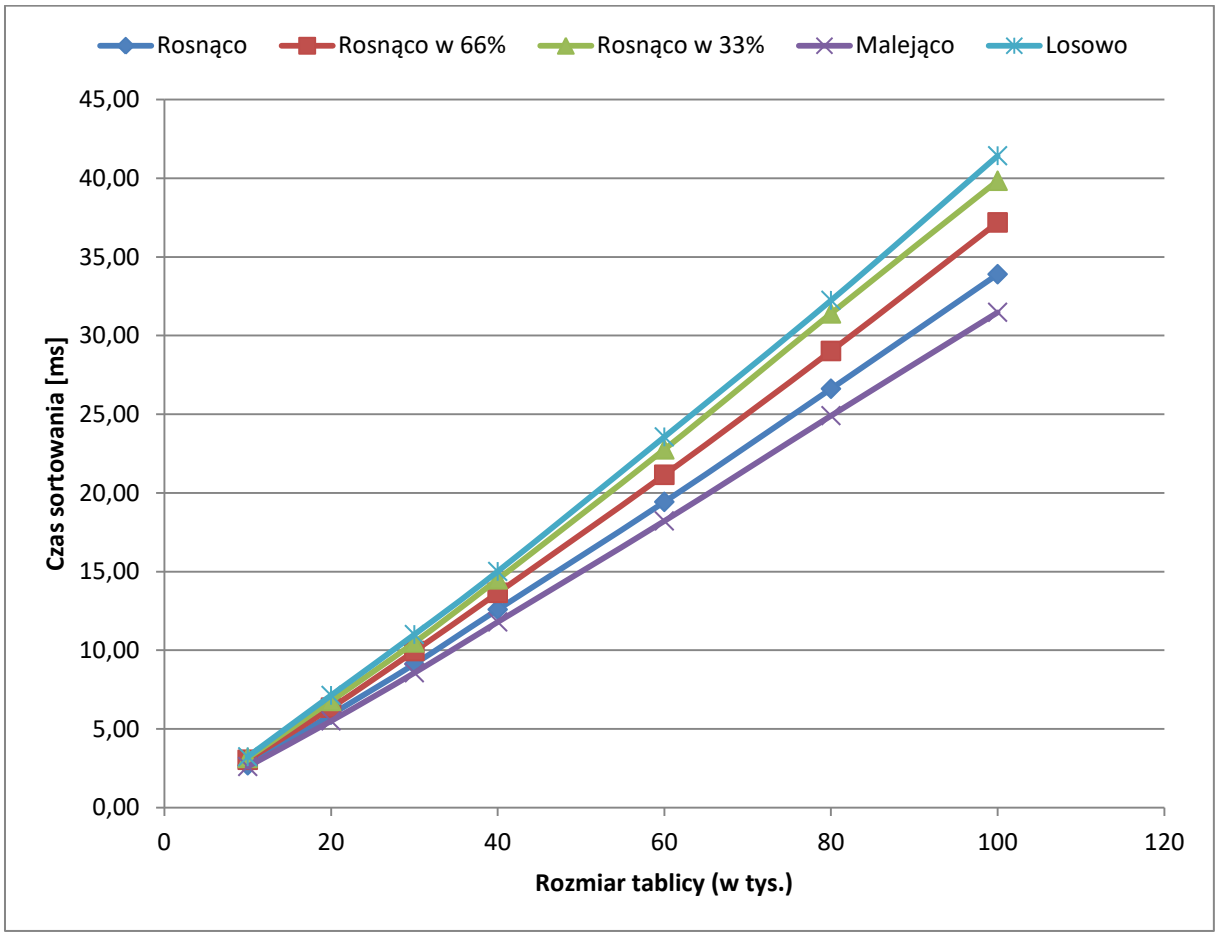
Sortowanie na danych typu float



Rys. 5 Czas sortowania w funkcji rozmiaru tablicy dla sortowania przez kopcowanie dla danych typu float

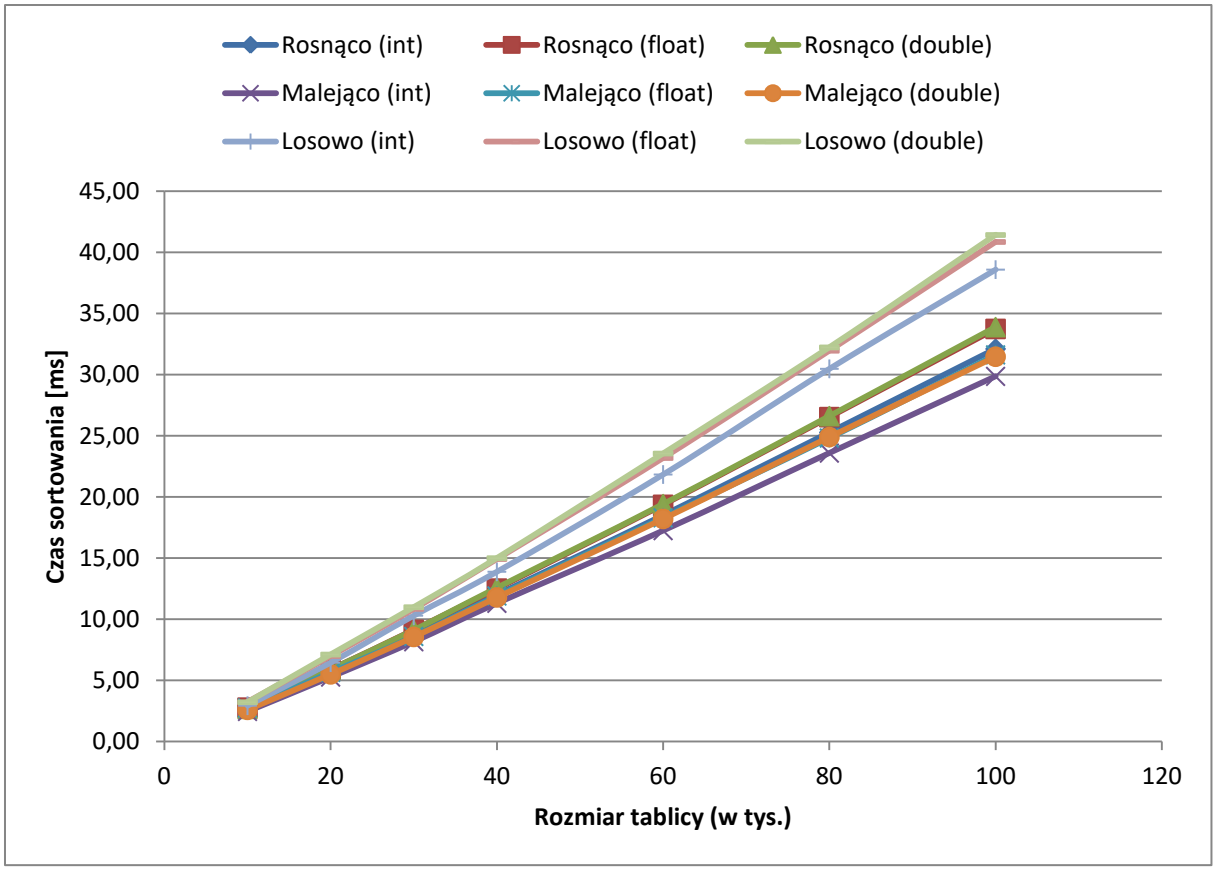
Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	2,69	3,04	3,16	2,61	3,23
20	5,91	6,35	6,78	5,51	7,13
30	9,13	9,95	10,50	8,57	11,00
40	12,59	13,65	14,51	11,80	15,01
60	19,43	21,15	22,77	18,22	23,56
80	26,62	29,02	31,41	24,91	32,24
100	33,90	37,19	39,84	31,48	41,43

Sortowanie na danych typu double



Rys. 6 Czas sortowania w funkcji rozmiaru tablicy dla sortowania przez kopcowanie dla danych typu double

Ze względu na dużą liczbę danych, poniższy wykres porównujący różne typy danych prezentuje tylko przypadki skrajne (posortowana rosnąco, posortowana malejąco i pozostawiona losowo)



Rys. 7 Porównanie czasu sortowania w funkcji rozmiaru tablicy dla sortowania przez kopcowanie dla różnych typów danych

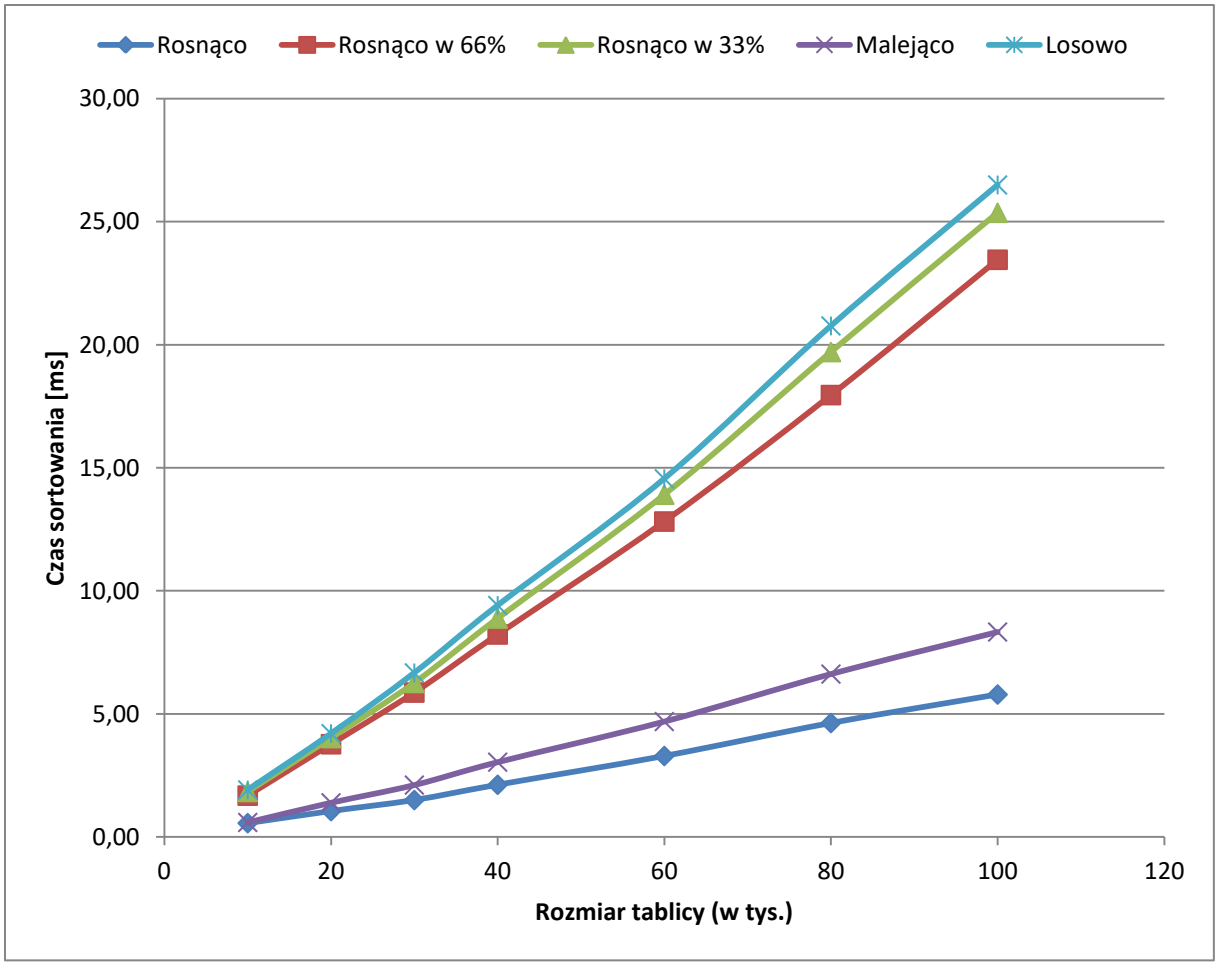
- Sortowanie Shella

Sortowanie Shella dla obu uwzględnionych odstępów okazało się podobnie efektywnym algorytmem co sortowanie przez kopcowanie. Tak jak wcześniej zostało opisane, sortowanie Shella zostało zbadane dla dwóch różnych odstępów: Shella oraz Hibbarda. Wyniki wykonanych pomiarów zamieszczone są w poniższych tabelach oraz na wykresach.

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	0,56	1,68	1,83	0,59	1,91
20	1,05	3,77	4,02	1,39	4,20
30	1,50	5,87	6,26	2,11	6,67
40	2,12	8,23	8,88	3,04	9,41
60	3,29	12,82	13,91	4,69	14,56
80	4,63	17,96	19,71	6,62	20,77
100	5,79	23,46	25,37	8,33	26,50

Sortowanie na danych typu int (z odstępami Shella)

*wynik nieprecyzyjny z powodu licznych błędów grubych (część pomiarów zwracała wartość 0 ns)

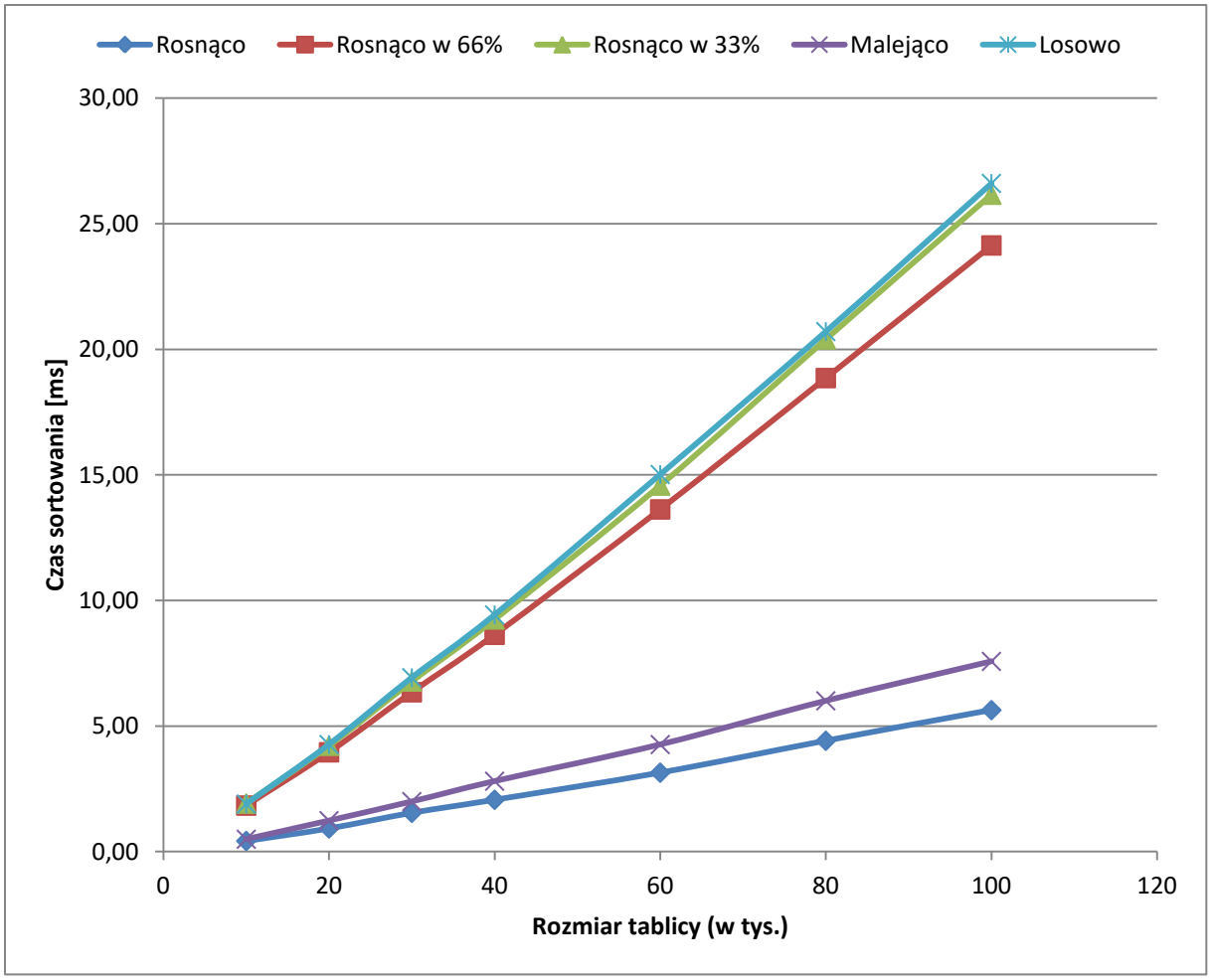


Rys. 8 Czas sortowania w funkcji rozmiaru tablicy dla sortowania Shella z odstępami Shella dla danych typu int

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	0,43	1,84	1,92	0,50	1,90
20	0,93	3,96	4,22	1,24	4,27
30	1,55	6,35	6,78	2,00	6,93
40	2,07	8,64	9,25	2,81	9,43
60	3,15	13,63	14,58	4,27	15,02
80	4,42	18,86	20,40	6,01	20,71
100	5,64	24,14	26,16	7,58	26,61

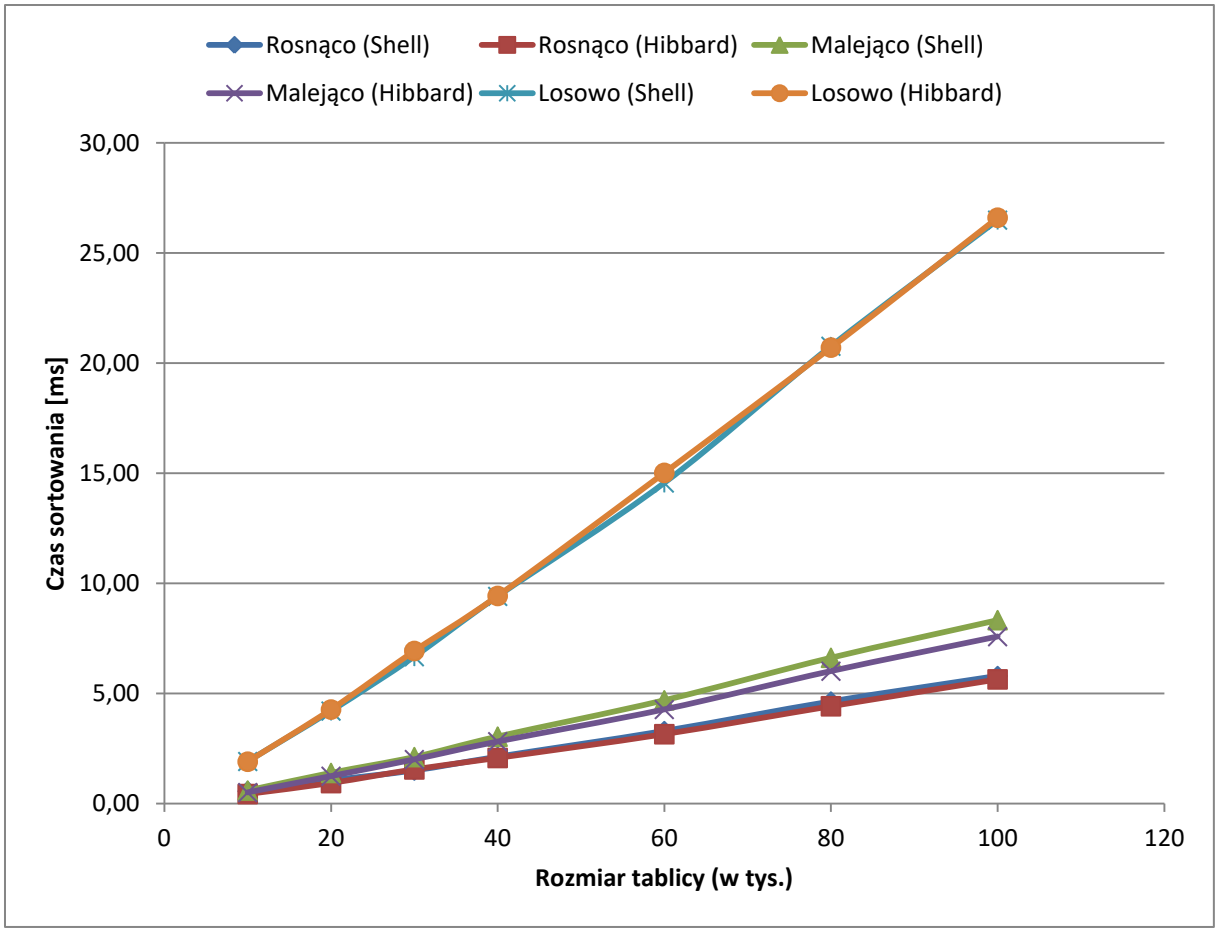
Sortowanie na danych typu int (z odstępami Hibbarda)

*wynik nieprecyzyjny z powodu licznych błędów grubych (część pomiarów zwracała wartość 0 ns)



Rys. 9 Czas sortowania w funkcji rozmiaru tablicy dla sortowania Shella z odstępami Hibbarda dla danych typu int

Dla obu powyższych odstępów wpływ rozmieszczenia danych okazał się zbliżony. Poniżej dla przypadków skrajnych (posortowana rosnąco, posortowana malejąco i pozostawiona losowo) porównane są czasy sortowania z odstępami Shella i Hibbarda.



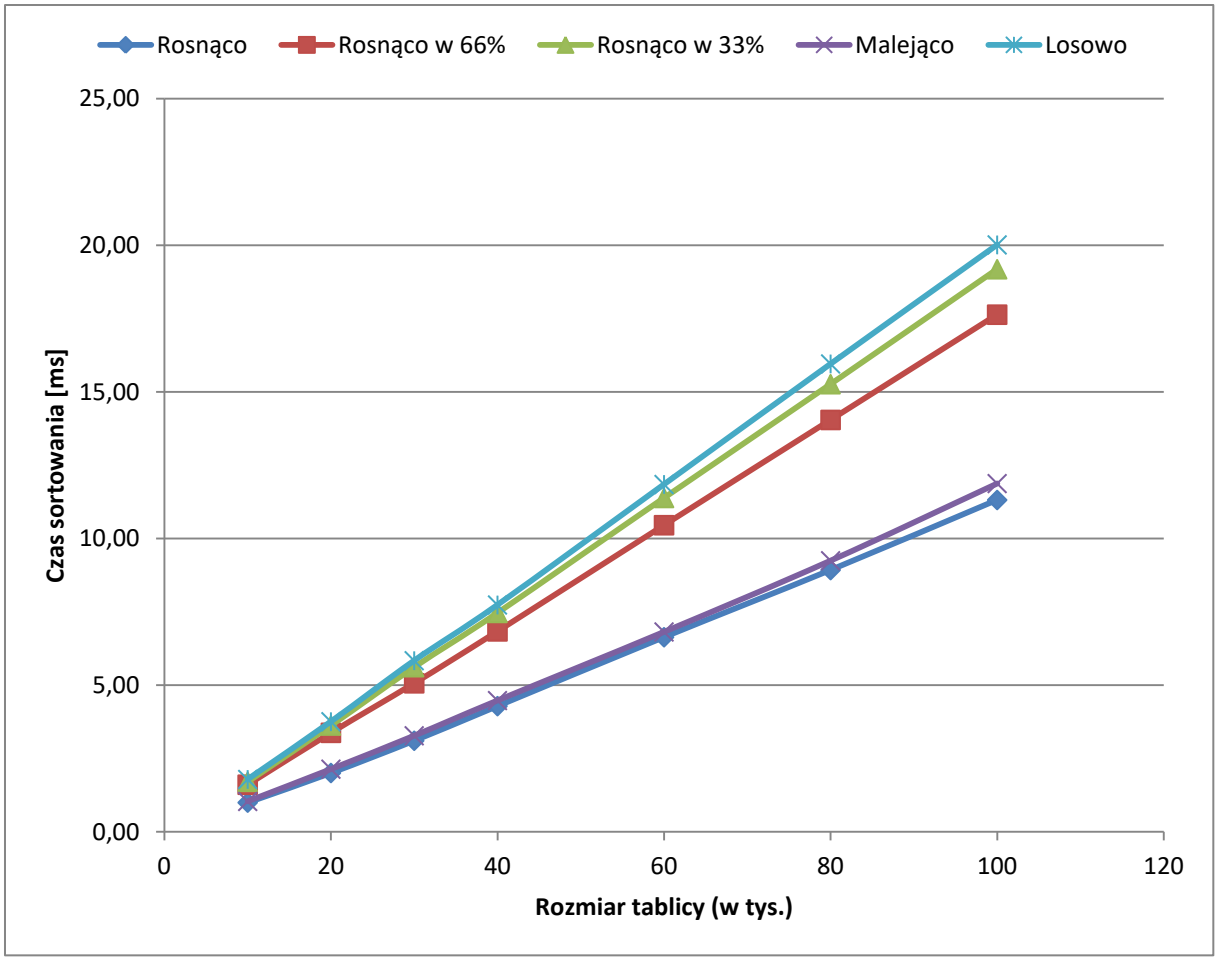
Rys. 10 Porównanie czasu sortowania w funkcji rozmiaru tablicy dla sortowania Shella z odstępami Shella i Hibbarda dla danych typu int

- Sortowanie szybkie

W sortowaniu szybkim poza rozmieszczeniem danych w tablicy, zbadany został również wpływ wyboru pivota na czas wykonania algorytmu. Wyniki wykonanych pomiarów zamieszczone są w poniższych tabelach oraz na wykresach.

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	1,00	1,61	1,71	1,04	1,79
20	2,01	3,38	3,63	2,14	3,76
30	3,11	5,07	5,62	3,28	5,84
40	4,29	6,84	7,47	4,48	7,74
60	6,64	10,46	11,39	6,82	11,85
80	8,93	14,05	15,27	9,25	15,96
100	11,32	17,64	19,19	11,88	20,02

Sortowanie na danych typu int (dla losowego pivota)

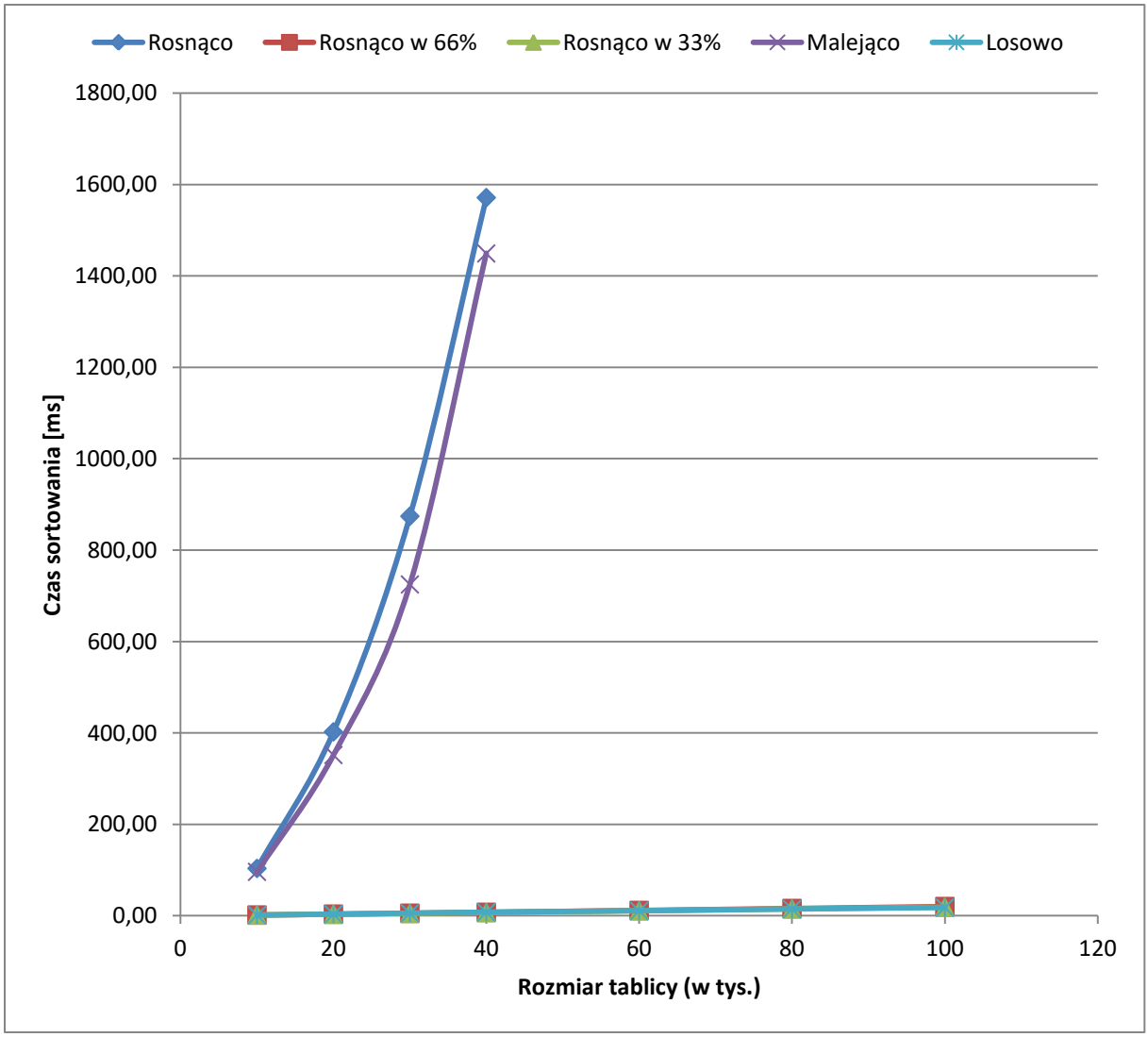


Rys. 11 Czas sortowania w funkcji rozmiaru tablicy dla sortowania szybkiego z losowo wybranym pivotem dla danych typu int

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	103,71	1,74	1,65	96,22	1,47
20	402,55	3,62	3,44	351,93	3,37
30	874,41	5,63	5,19	725,17	5,68
40	1571,64	7,65	7,07	1449,49	7,68
60	*	11,81	10,87	*	10,83
80	*	16,03	14,79	*	14,61
100	*	20,31	18,67	*	17,96

Sortowanie na danych typu int (dla skrajnie lewego pivota)

*przepełnienie stosu w wyniku wielokrotnej rekurencji

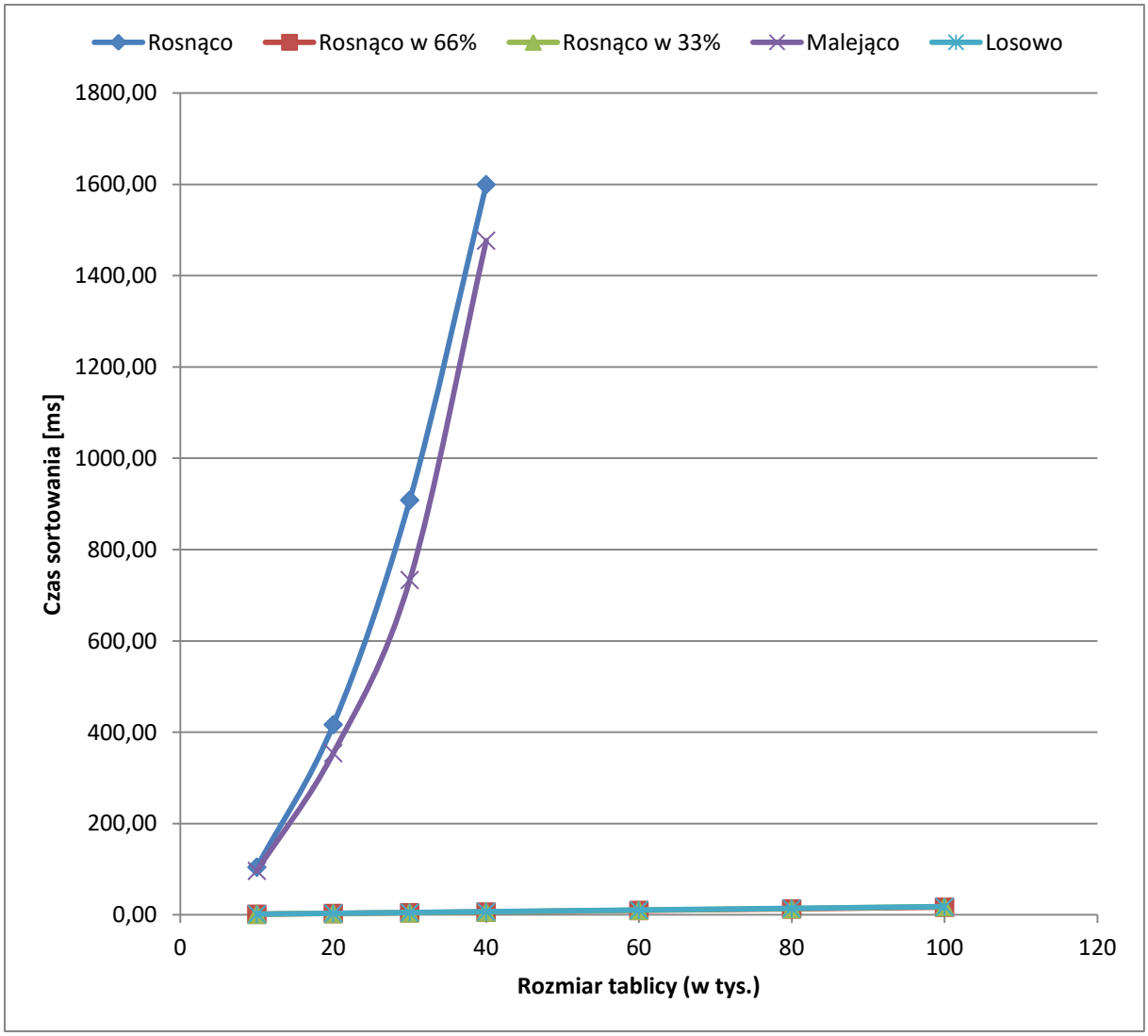


Rys. 12 Czas sortowania w funkcji rozmiaru tablicy dla sortowania szybkiego ze skrajnym lewym pivotem dla danych typu int

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	104,65	1,47	1,55	96,54	1,51
20	416,76	3,05	3,19	354,29	3,23
30	908,58	4,72	4,78	733,56	5,05
40	1599,78	6,25	6,54	1476,99	6,90
60	*	9,78	10,17	*	10,50
80	*	13,11	13,91	*	14,08
100	*	16,82	17,87	*	17,79

Sortowanie na danych typu int (dla skrajnie prawego pivota)

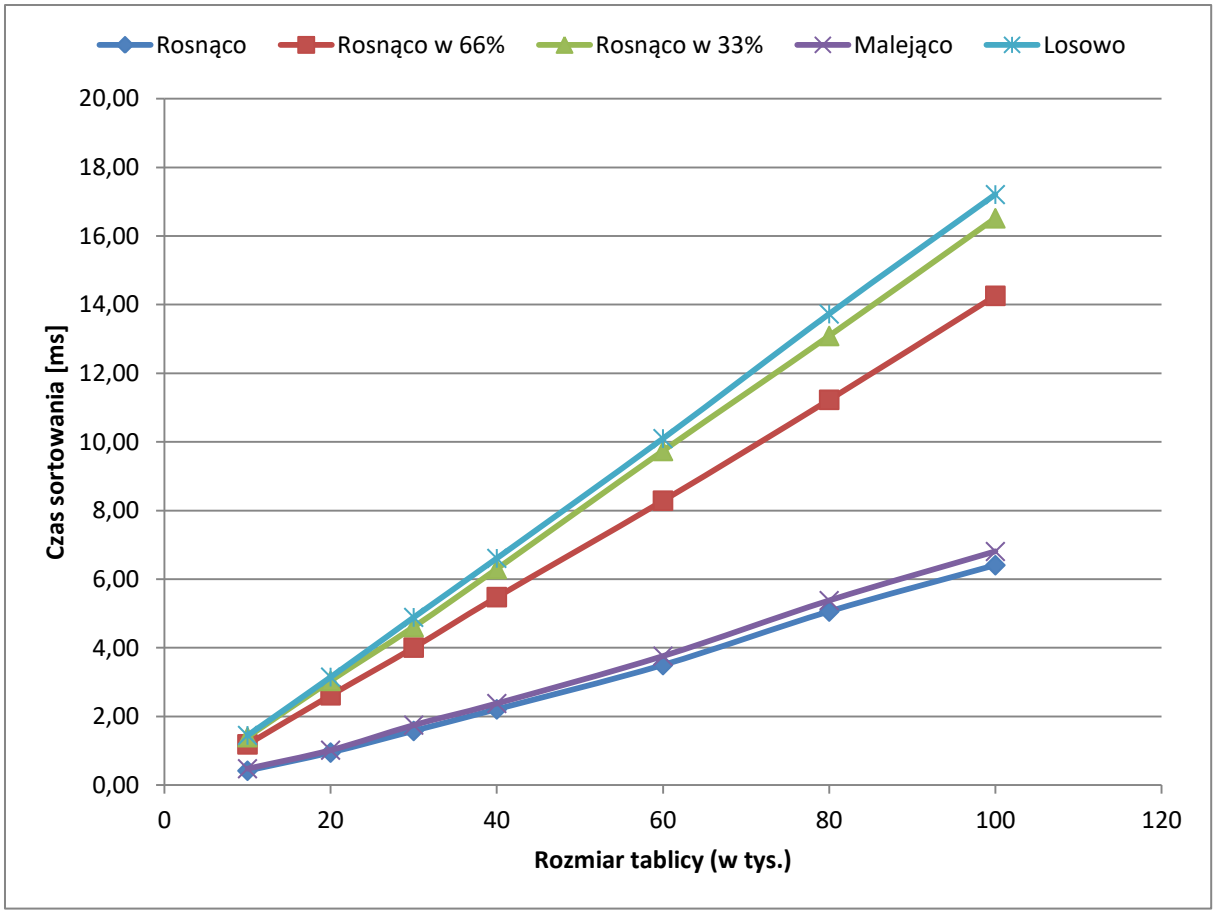
*przepełnienie stosu w wyniku wielokrotnej rekurencji



Rys. 13 Czas sortowania w funkcji rozmiaru tablicy dla sortowania szybkiego ze skrajnym prawym pivotem dla danych typu int

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	0,42	1,19	1,40	0,48	1,45
20	0,95	2,62	3,04	1,02	3,15
30	1,58	4,01	4,61	1,75	4,89
40	2,21	5,48	6,30	2,38	6,61
60	3,50	8,29	9,74	3,76	10,10
80	5,06	11,23	13,09	5,38	13,73
100	6,41	14,26	16,52	6,81	17,21

Sortowanie na danych typu int (dla środkowego pivota)



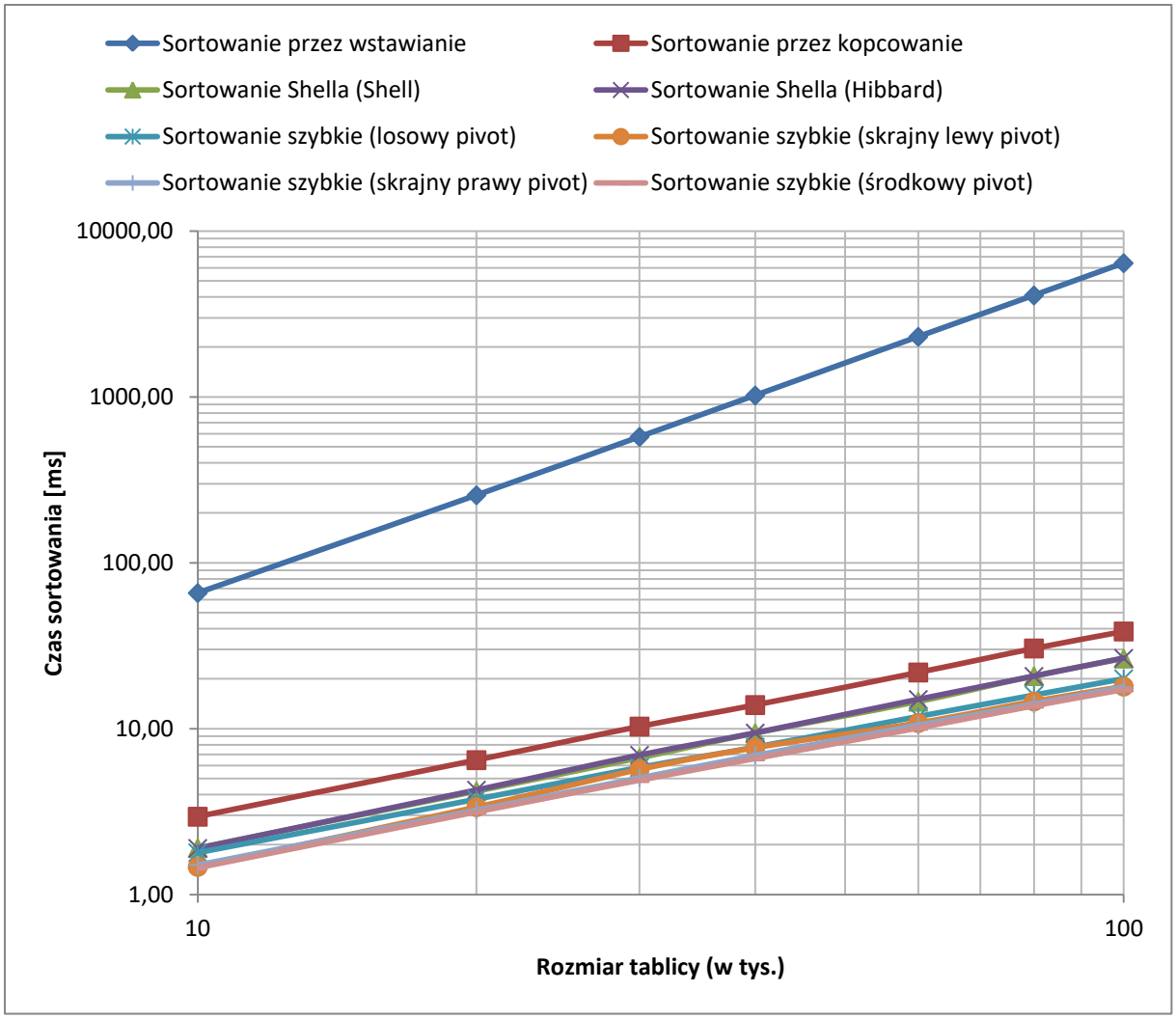
Rys. 14 Czas sortowania w funkcji rozmiaru tablicy dla sortowania szybkiego ze środkowym pivotem dla danych typu int

- Zestawienie wszystkich algorytmów sortowania

Poniżej zamieszczone są wykresy porównujące wszystkie algorytmy sortowania przy niesortowanej wstępnie tablicy, czyli dla przypadku średniego. Dla lepszej czytelności danych na wykresie została zastosowana skala logarytmiczna.

Rozmiary tablic (w tys.)	Średni czas sortowania dla danego algorytmu sortowania (przypadek średni) [ms]							
	Przez wstawianie	Przez kopcowanie	Shella (Shell)	Shella (Hibbard)	Szybkie (losowy pivot)	Szybkie (skrajny lewy pivot)	Szybkie (skrajny prawy pivot)	Szybkie (środkowy pivot)
10	65,82	2,95	1,91	1,90	1,79	1,47	1,51	1,45
20	255,69	6,47	4,20	4,27	3,76	3,37	3,23	3,15
30	574,88	10,31	6,67	6,93	5,84	5,68	5,05	4,89
40	1021,68	13,90	9,41	9,43	7,74	7,68	6,90	6,61
60	2306,83	21,85	14,56	15,02	11,85	10,83	10,50	10,10
80	4096,12	30,49	20,77	20,71	15,96	14,61	14,08	13,73
100	6411,91	38,60	26,50	26,61	20,02	17,96	17,79	17,21

Sortowanie na danych typu int (przypadki średnie poszczególnych algorytmów sortowania)



Rys. 15 Zestawienie różnych algorytmów sortowania na wykresie czasu sortowania w funkcji rozmiaru tablicy na wstępnie niesortowanej tablicy dla danych typu int **(skala logarytmiczna)**

4. Podsumowanie i wnioski

Na podstawie tabel oraz wykresów przedstawionych w powyższych wynikach, można dokonać następujących obserwacji oraz uzyskać wnioski:

- Sortowanie przez wstawianie to algorytm efektywny tylko dla małej ilości danych lub częściowo posortowanych. Od losowego rozmieszczenia danych gorsze były tylko dane posortowane malejąco, czyli przypadek najgorszy o (pesymistycznej) złożoności obliczeniowej $O(n^2)$. W przypadku przeprowadzonego eksperymentu dla tablicy 100 tysięcy elementów o kolejności nierosnącej średni czas sortowania wyniósł ponad 12 sekund, podczas gdy czas sortowania pozostałych algorytmów mieścił się poniżej 0,1 sekundy dla zbliżonych zbiorów danych.
- W sortowaniu przez kopcowanie wynika na pozór nieoczywista zależność, gdzie czas sortowania danych o kolejności nierosnącej jest minimalnie krótszy niż danych o kolejności niemalejącej. Powodem jest implementacja algorytmu, w którym przed właściwym sortowaniem budowany jest z podanej tablicy kopiec maksymalny. Z racji

sposobu ułożenia elementów w tablicy o kolejności elementów nierosnącej, taki zbiór danych jednocześnie spełnia kryteria kopca maksymalnego, toteż funkcja wstępnej budowy kopca trwa najkrócej dla takiego zbioru danych, stąd taka zależność.

- W przypadku testowania różnych typów danych dla sortowania przez kopcowanie nie zauważono znaczącej różnicy czasu sortowania pomiędzy testowanymi typami danych (int, float, double). Właściwie we wszystkich punktach pomiarowych, dane typu int sortowane były tylko minimalnie szybciej od danych zmiennoprzecinkowych (float i double). Uzyskano niewielką różnicę czasów na poziomie około 3~4%.
- Dla algorytmu sortowania Shella zauważona została nietypowa zależność, w której zbiór danych wstępnie posortowany rosnąco lub malejąco, znacząco krócej jest sortowany niż dane posortowane częściowo lub wcale. Dzieje się tak, ponieważ podczas kolejnych iteracji algorytmu, gdzie w każdej odstępy algorytmu się zmieniają, losowe dane mogą być przemieszczane wielokrotnie w przód i w tył, co w uporządkowanych zbiorach danych jest zminimalizowane. Należy jednak zauważyć, że ta zależność została uzyskana dla odstępów Shella i Hibbarda. Dla innych odstępów zależności te mogą ulec zmianie, bowiem mają one znaczący wpływ na złożoność obliczeniową algorytmu w ogóle.
- W większości przypadków sortowania szybkiego uzyskano zgodnie z oczekiwaniami krótki czas sortowania. Należy jednak zwrócić uwagę na to, że są to przypadki średnie o mniejszej złożoności obliczeniowej $O(n \log n)$. Dla przypadków pesymistycznych uzyskano znacznie gorszą złożoność obliczeniową na poziomie $O(n^2)$, osiągając porównywalnie niską efektywność co algorytm sortowania przez wstawianie
- Zestawiając wszystkie badane algorytmy sortowania na jednym wykresie, dostrzec można, że dla przypadku średniego czyli losowo rozmieszczonych danych, największą efektywność uzyskuje sortowanie szybkie. Tak jak jednak podałem wyżej, należy być świadomym ograniczeń różnych algorytmów sortowania. Przykładowo dla danych w większości posortowanych rosnąco , sortowanie przez wstawianie wykazałoby się znacznie większą efektywnością od pozostałych algorytmów.

5. Literatura

https://en.wikipedia.org/wiki/Insertion_sort
https://en.wikipedia.org/wiki/Heapsort
https://en.wikipedia.org/wiki/Shellsort
https://en.wikipedia.org/wiki/Quicksort
https://stackoverflow.com/questions/49090366/how-to-convert-stdchronohigh-resolution-clocknow-to-milliseconds-micros
https://www.geeksforgeeks.org/insertion-sort/
https://www.geeksforgeeks.org/heap-sort/
https://www.geeksforgeeks.org/shellsort/
https://eduinf.waw.pl/inf/alg/003_sort/0012.php