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:

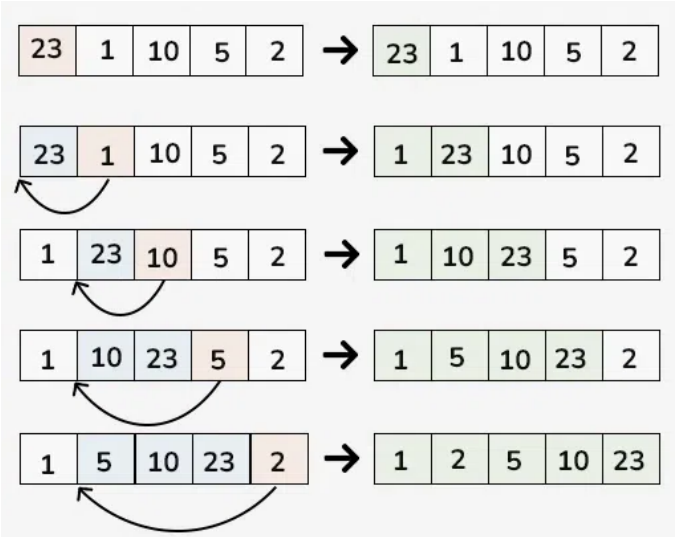
1. Wprowadzenie

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 reprezentatywnych rozmiarach, które zostały dobrane eksperymentalnie dla poszczególnych algorytmów sortowania. Rozmiary tablic są następujące:

Sortowanie przez wstawianie (typ danych: int):
8000, 12000, 18000, 27000, 40000, 60000, 100000 elementów

Sortowanie przez kopcowanie (typ danych: int, float, double):
10000, 20000, 50000, 100000, 200000, 500000, 1000000 elementów

Sortowanie Shella (typ danych: int):
30000, 50000, 100000, 200000, 500000, 1000000, 2000000 elementów

Sortowanie szybkie (typ danych: int):
10000, 20000, 40000, 80000, 160000, 320000, 640000 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()**.

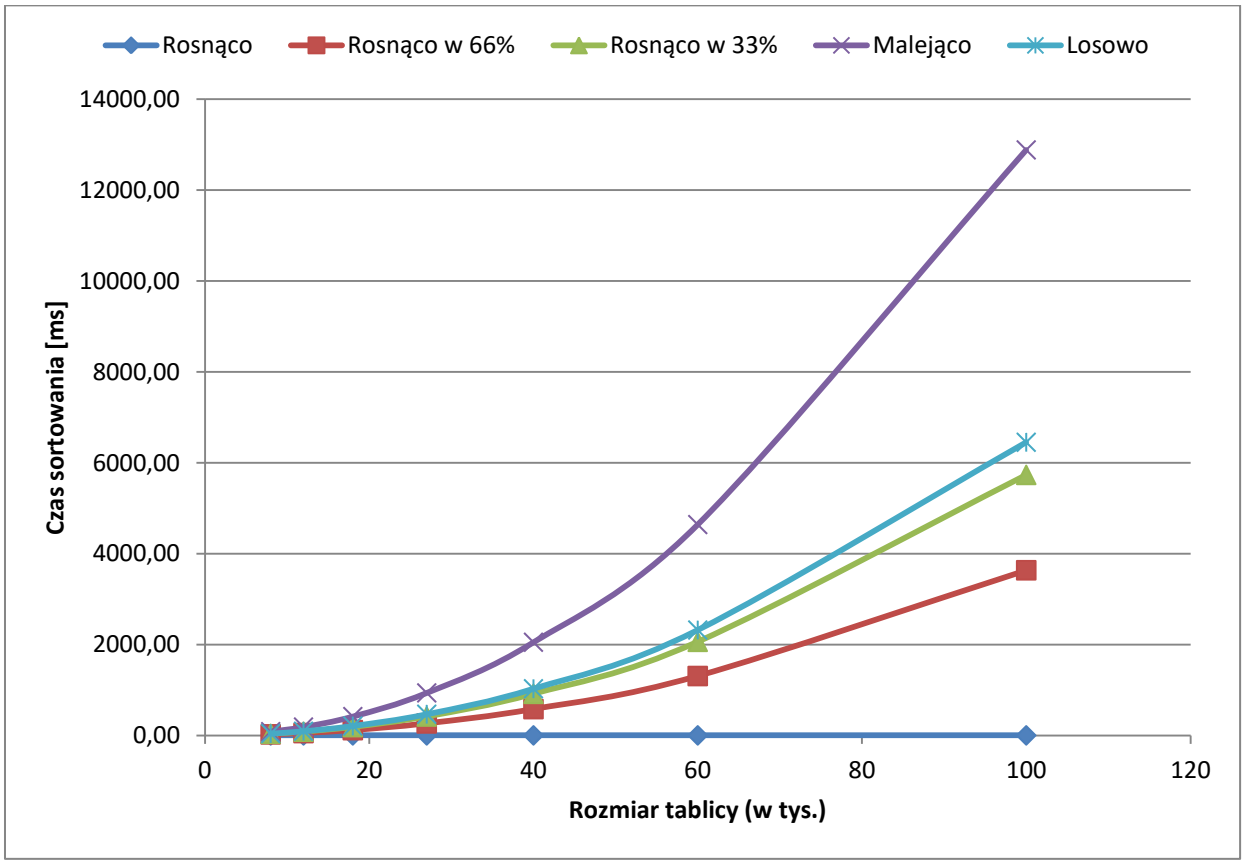
3. Wyniki eksperymentu

- Sortowanie przez wstawianie

Ze względu na swoją złożoność obliczeniową, ten algorytm był najbardziej czasochłonny. Z tego powodu zredukowano zakres sprawdzanych rozmiarów tablic do 100 tysięcy, co zostało już podane w planie eksperymentu. 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)
8	1,01	23,19	36,48	83,68	40,95
12	1,00	52,08	82,21	185,43	92,51
18	1,01	116,95	185,12	415,99	208,19
27	1,01	264,53	416,77	936,17	466,45
40	1,01	579,79	915,60	2050,46	1028,29
60	1,00	1307,35	2060,30	4640,39	2320,02
100	1,01	3634,81	5733,54	12884,80	6450,56

Sortowanie na danych typu int



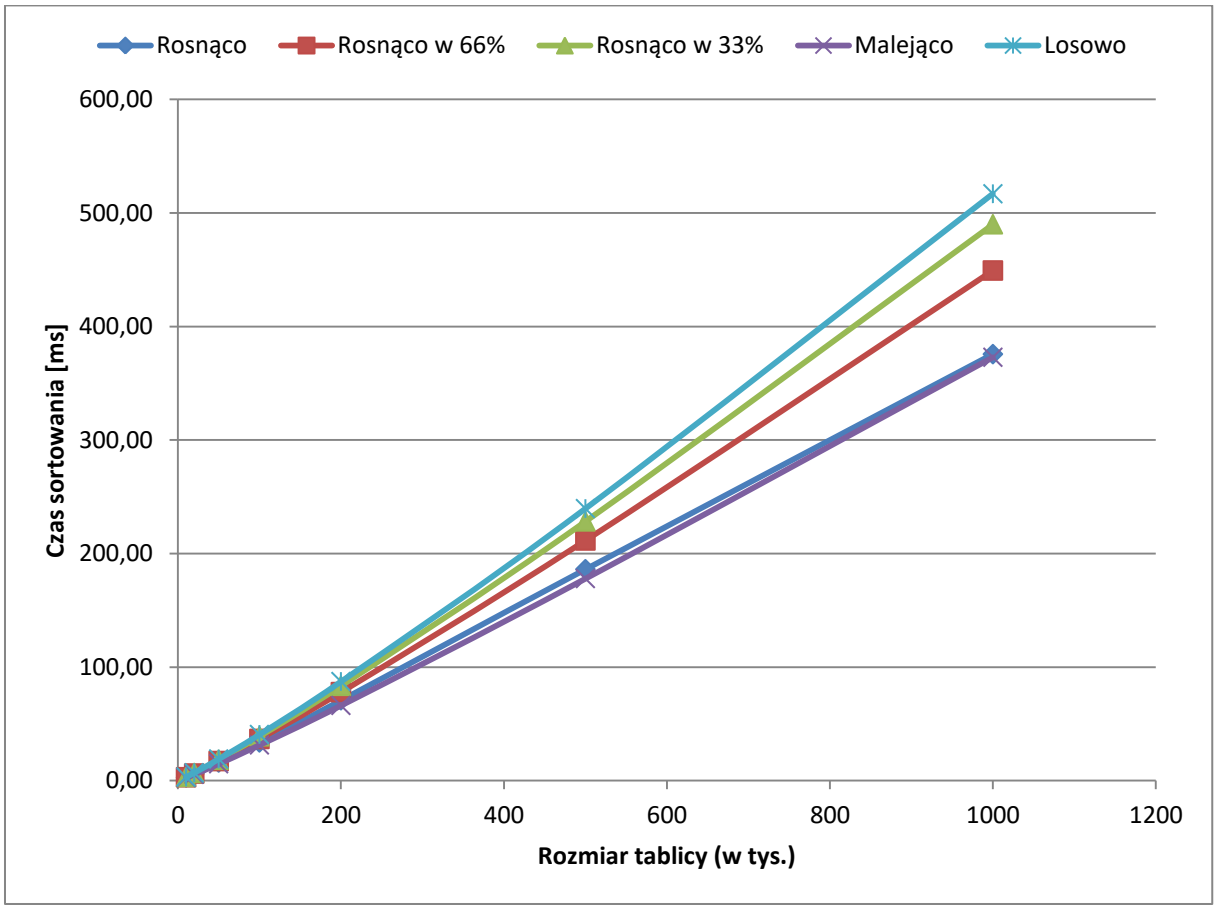
Rys. 3 Czas sortowania w funkcji rozmiaru tablicy dla sortowania przez wstawianie dla danych typu int

- Sortowanie przez kopcowanie

W sortowaniu przez kopcowanie zakres sprawdzanych tablic został powiększony ze względu na większą efektywność algorytmu przy rozmiarach większych niż 100 tysięcy. Ponadto dla tego algorytmu została wybrana opcja 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	2,79	2,96	3,23	2,74	3,26
20	5,89	6,21	6,75	5,54	6,92
50	15,97	17,21	18,35	14,87	19,06
100	33,63	36,68	39,08	31,42	40,65
200	70,79	77,93	83,52	66,19	87,27
500	186,13	211,49	228,31	177,91	239,57
1000	375,57	449,20	489,97	372,80	516,82

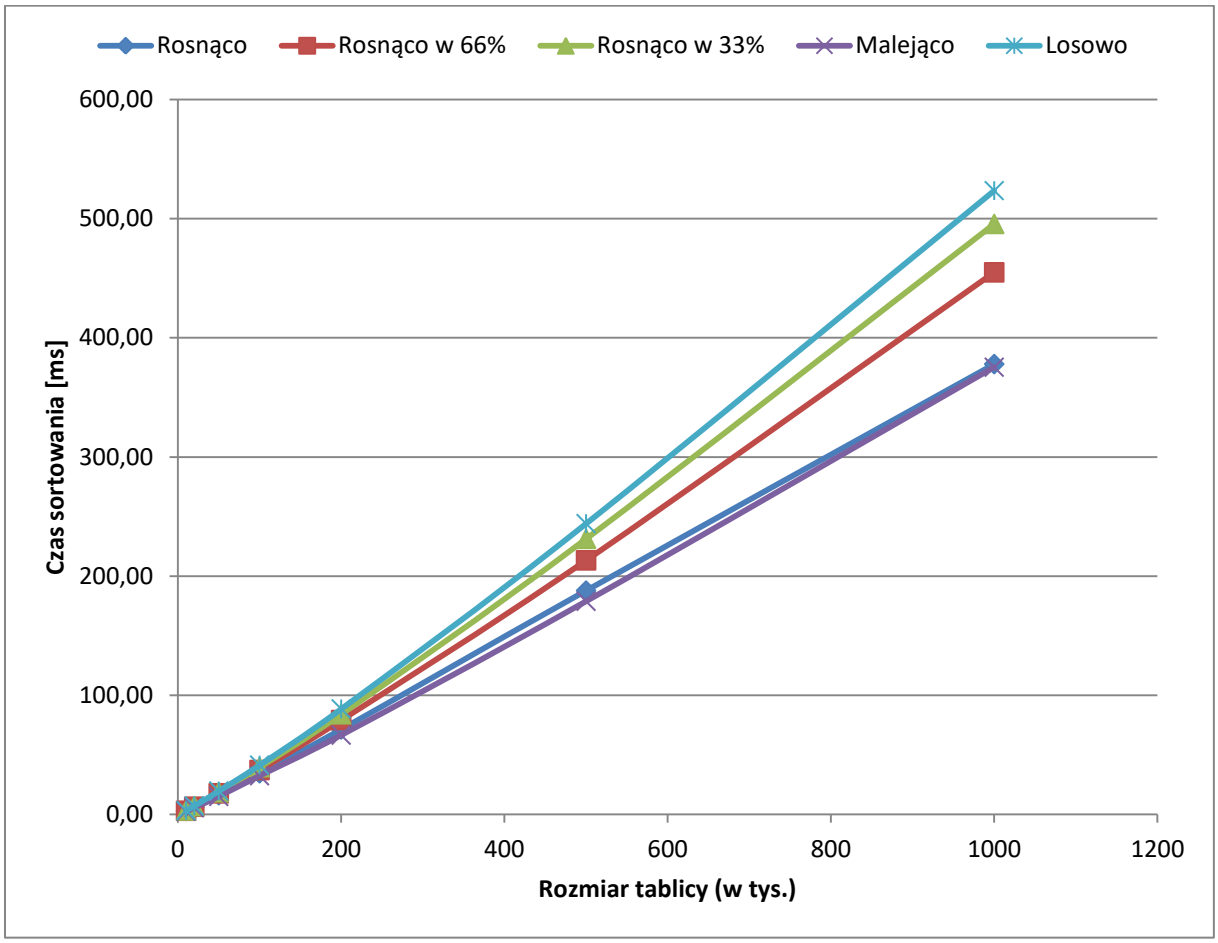
Sortowanie na danych typu int



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	2,78	2,94	3,13	2,51	3,23
20	5,92	6,32	6,80	5,48	7,01
50	16,05	17,59	18,46	14,95	19,52
100	34,43	37,09	39,53	32,31	41,31
200	71,19	79,05	83,94	66,43	88,47
500	187,75	213,14	231,19	178,74	244,00
1000	377,91	454,91	495,56	375,31	523,53

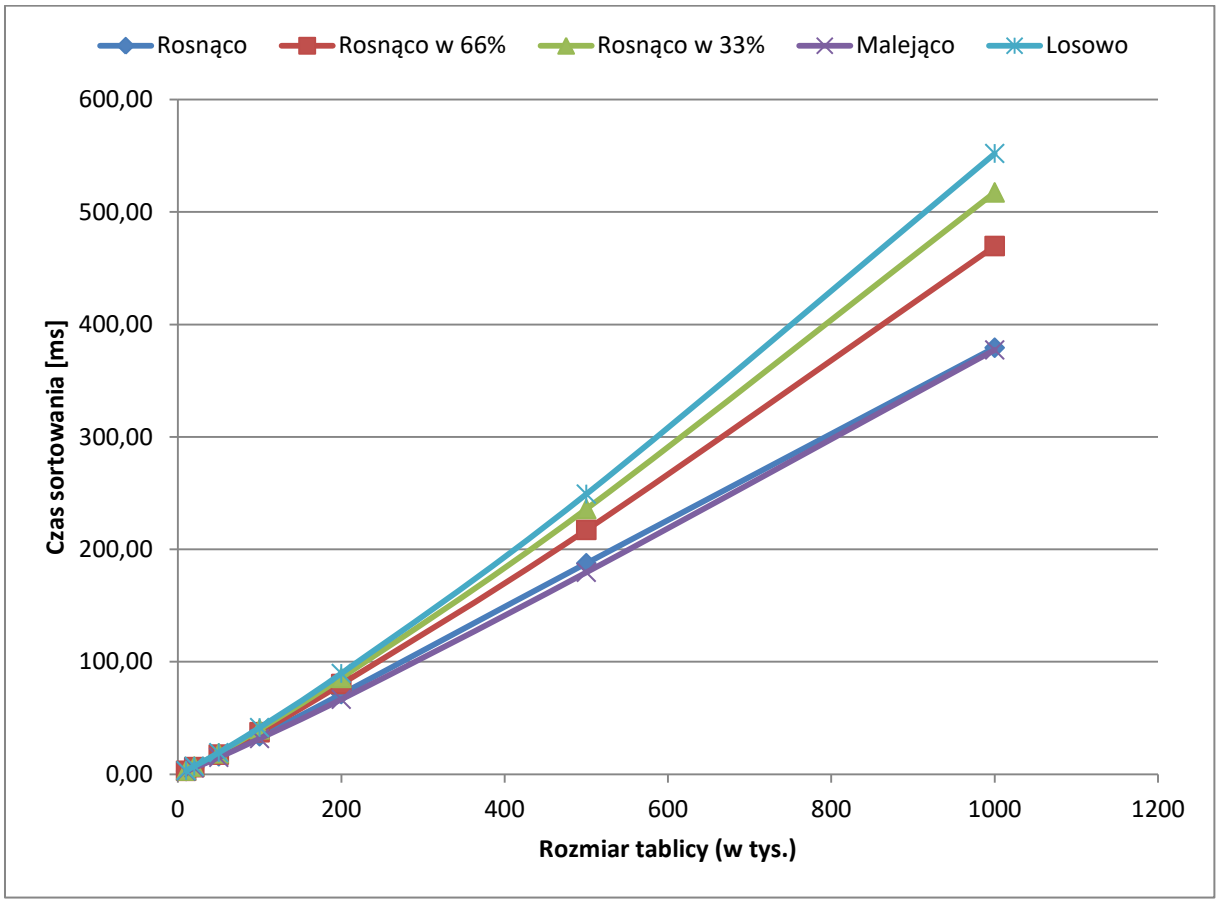
Sortowanie na danych typu float



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

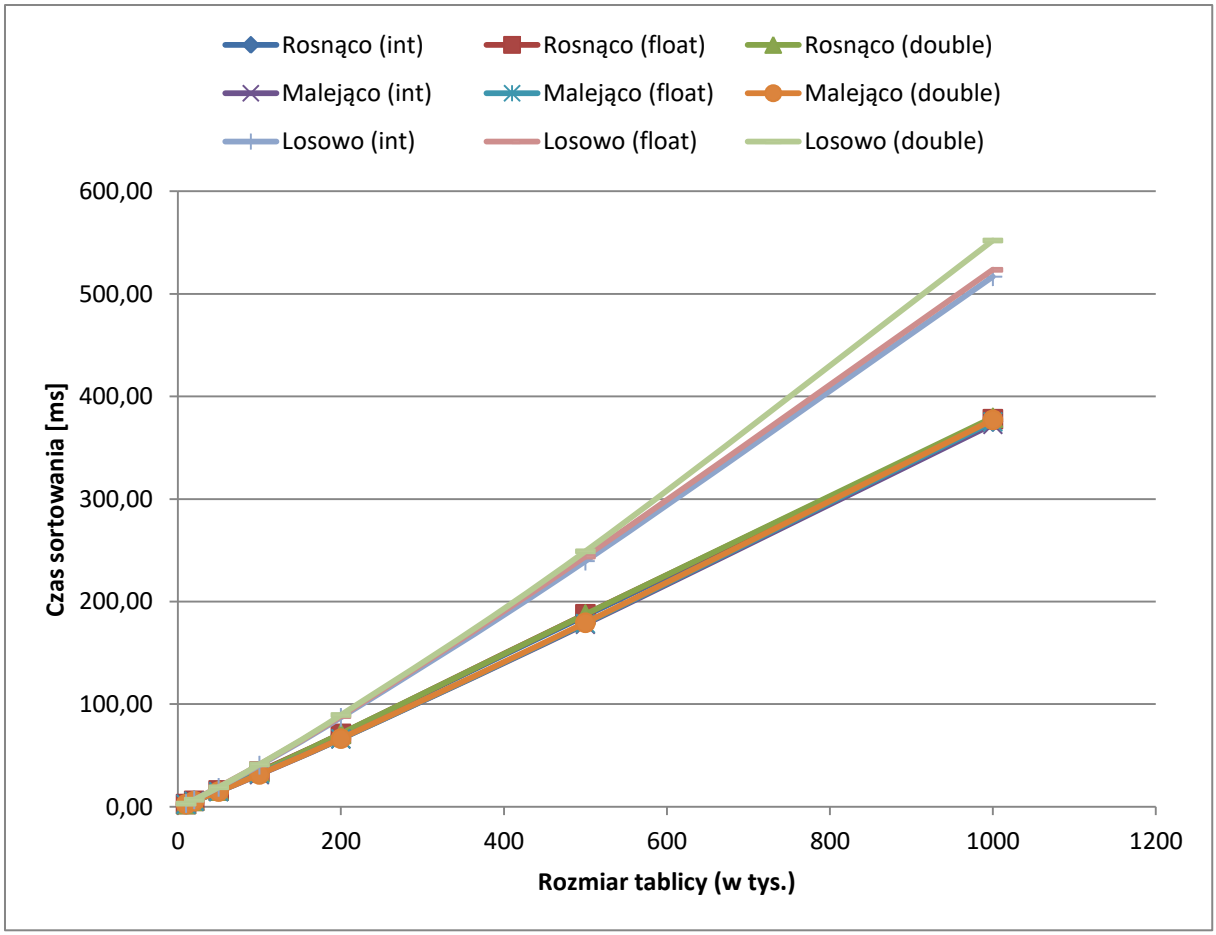
Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
10	2,81	3,01	3,18	2,69	3,20
20	5,92	6,26	6,78	5,62	7,03
50	15,96	17,40	18,53	14,93	19,16
100	33,76	37,29	39,95	31,67	41,49
200	71,29	80,15	85,65	66,56	89,61
500	187,57	217,28	235,84	179,48	249,14
1000	379,27	469,72	517,41	377,32	552,07

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)



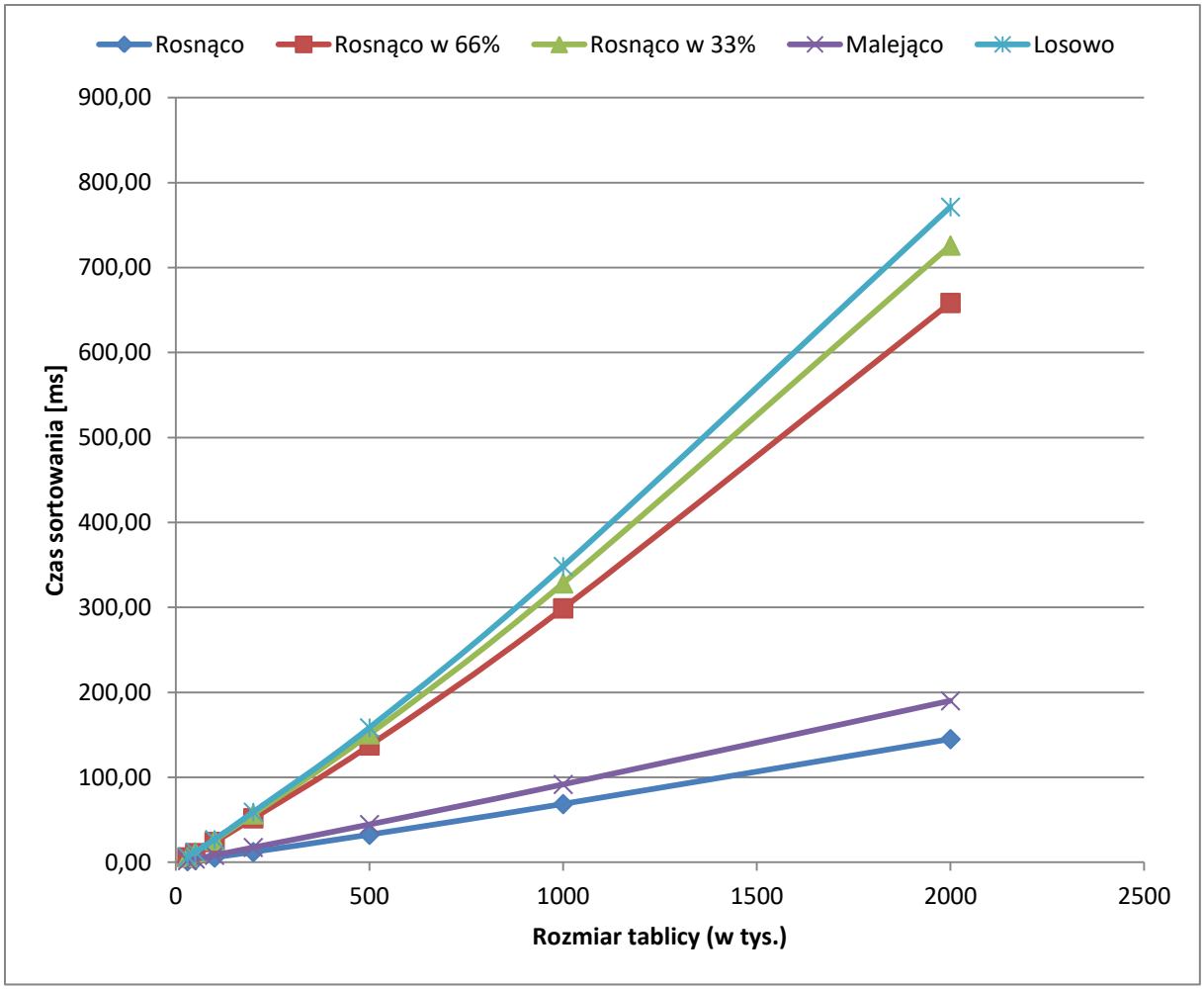
Rys. 7 Porównanie czasu sortowania w funkcji rozmiaru tablicy dla sortowania przez kopcowanie dla różnych typów danych

- Sortowanie Shella

Sortowanie Shella dla obu uwzględnionych odstępów okazało się na tyle efektywnym algorytmem, że rozmiary tablic zostały ponownie powiększone. 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)
30	1,50	5,92	6,36	2,12	6,69
50	2,77	10,92	11,78	3,97	12,29
100	5,79	24,08	25,88	8,36	26,86
200	12,16	51,96	56,61	17,58	59,18
500	32,51	137,72	150,85	44,66	158,45
1000	68,67	299,03	328,43	91,77	348,45
2000	145,13	658,45	726,05	190,18	771,38

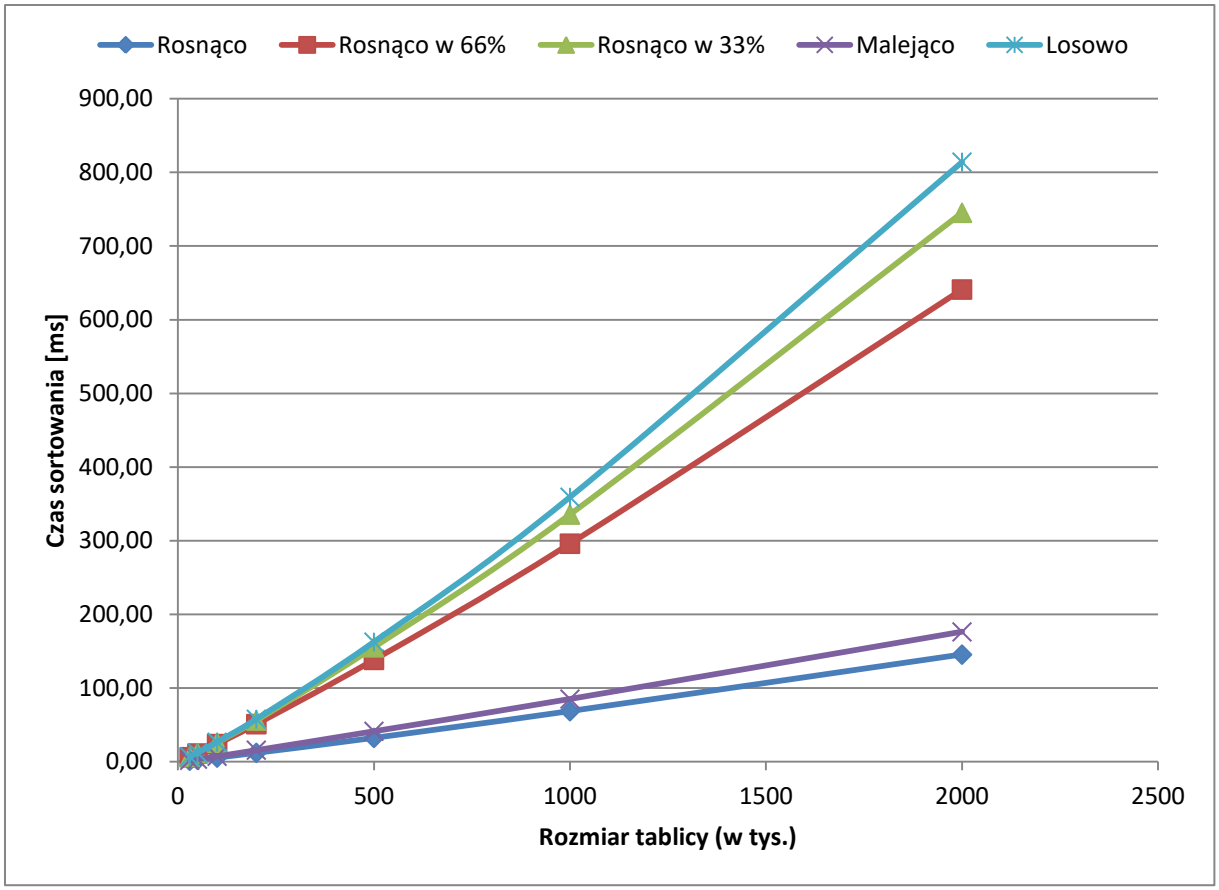
Sortowanie na danych typu int (z odstępami Shella)



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

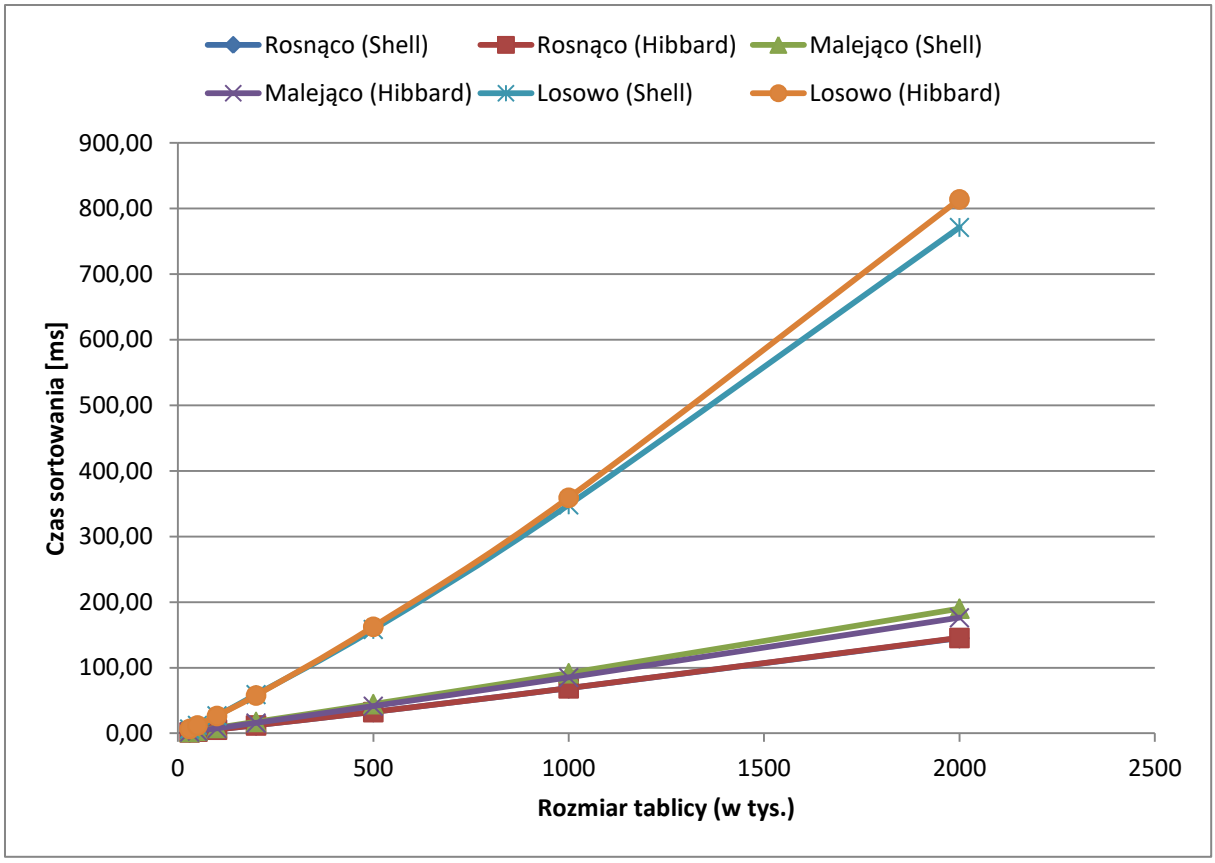
Rozmiary tablic (w tys.)	Średni czas sortowania dla danego typu wstępnego posortowania [ms]				
	Rosnąco	Rosnąco w 66%	Rosnąco w 33%	Malejąco	Losowo (tablica wstępnie niesortowana)
30	1,49	6,21	6,55	1,99	6,80
50	2,73	11,24	11,92	3,53	12,08
100	5,69	24,03	25,89	7,46	26,55
200	12,13	51,09	56,04	15,78	57,77
500	32,44	138,38	155,43	41,43	162,52
1000	68,72	296,07	335,67	85,20	359,22
2000	145,48	641,14	745,30	176,34	814,19

Sortowanie na danych typu int (z odstępami Hibbarda)



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.



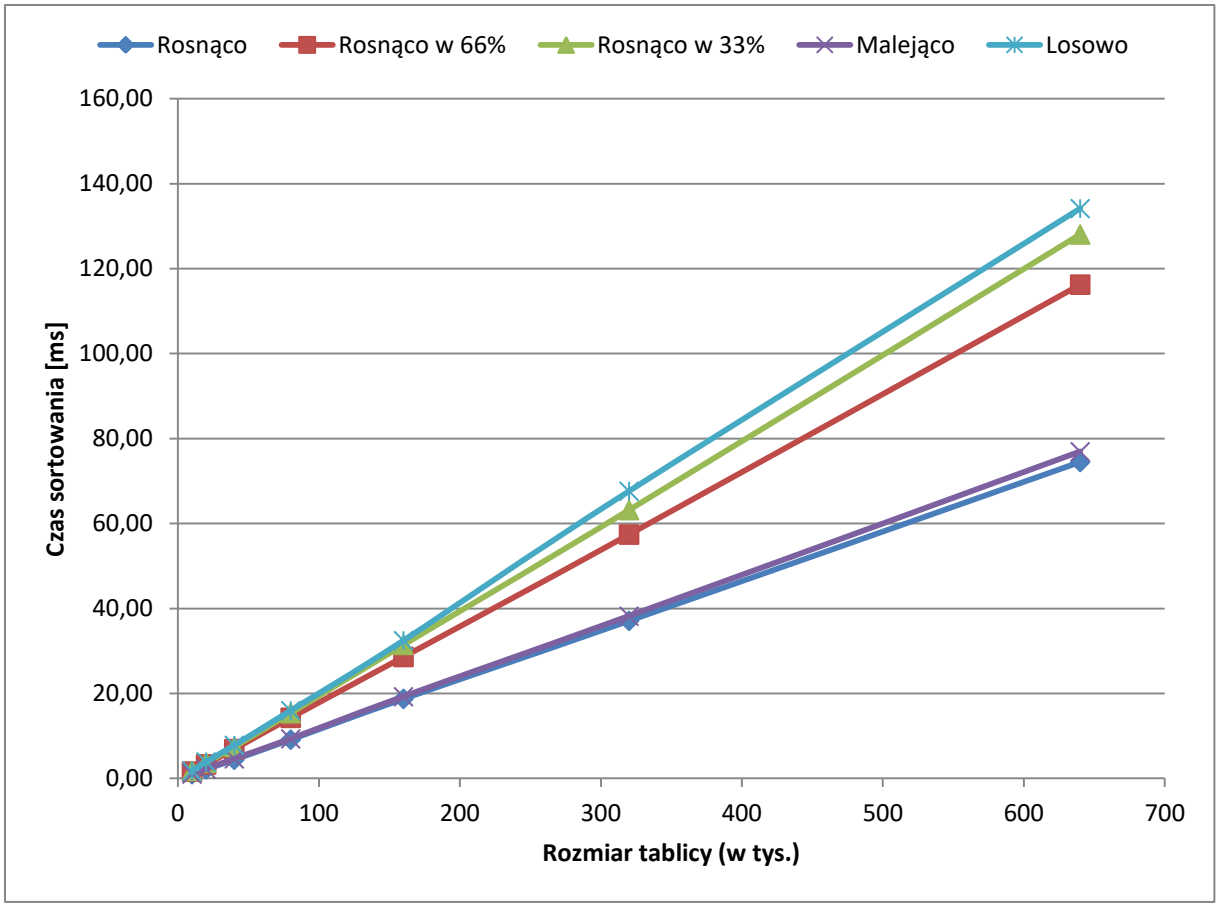
Rys. 10 Porównanie czasu sortowania w funkcji rozmiaru tablicy dla sortowania Shella z odstępami Shella i Hibbarda dla danych typu int

- Sortowanie szybkie

Ze względu na nakład pracy przy tym sortowaniu, badania zostały zredukowane do następujących etapów: sprawdzenie dla losowo rozmieszczonego pivota wszystkich typów rozmieszczenia danych oraz sprawdzenie wszystkich pivotów (skrajny lewy, skrajny prawy, środkowy i losowy) dla losowego rozmieszczenia danych. 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	1,02	1,62	1,78	1,04	1,75
20	2,16	3,31	3,69	2,21	3,90
40	4,42	6,91	7,45	4,62	7,81
80	9,14	14,28	15,39	9,32	15,97
160	18,74	28,61	31,46	19,25	32,46
320	37,11	57,43	63,19	38,21	67,62
640	74,51	116,26	128,07	76,93	134,16

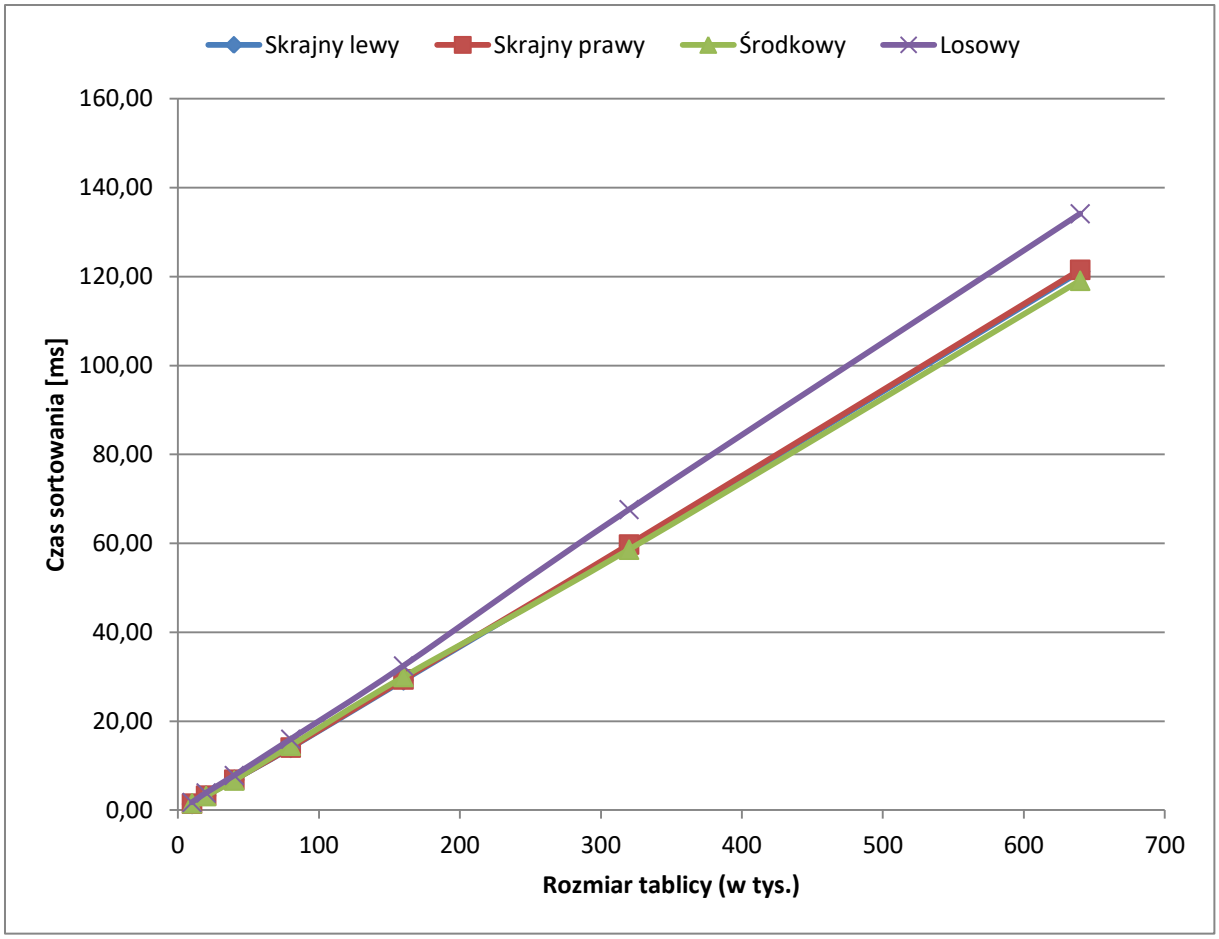
Sortowanie na danych typu int (dla losowego pivota)



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

Rozmiar tablicy (w tys.)	Średni czas sortowania dla danego sposobu wyboru pivota [ms]			
	Skrajny lewy	Skrajny prawy	Środkowy	Losowy
10	1,41	1,48	1,54	1,75
20	3,26	3,28	3,19	3,90
40	6,84	6,86	6,69	7,81
80	14,03	14,10	14,51	15,97
160	29,22	29,42	29,98	32,46
320	59,53	59,76	58,61	67,62
640	121,17	121,55	119,09	134,16

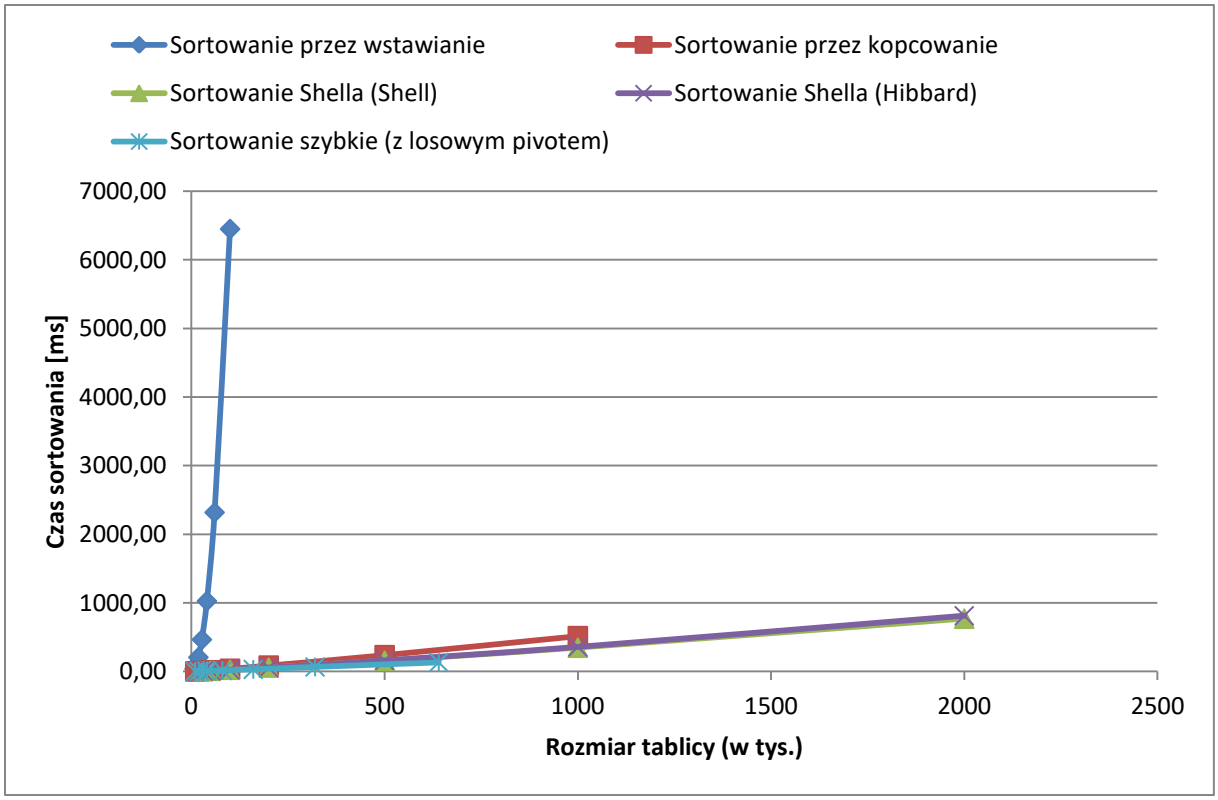
Sortowanie na danych typu int (tablica wstępnie niesortowana)



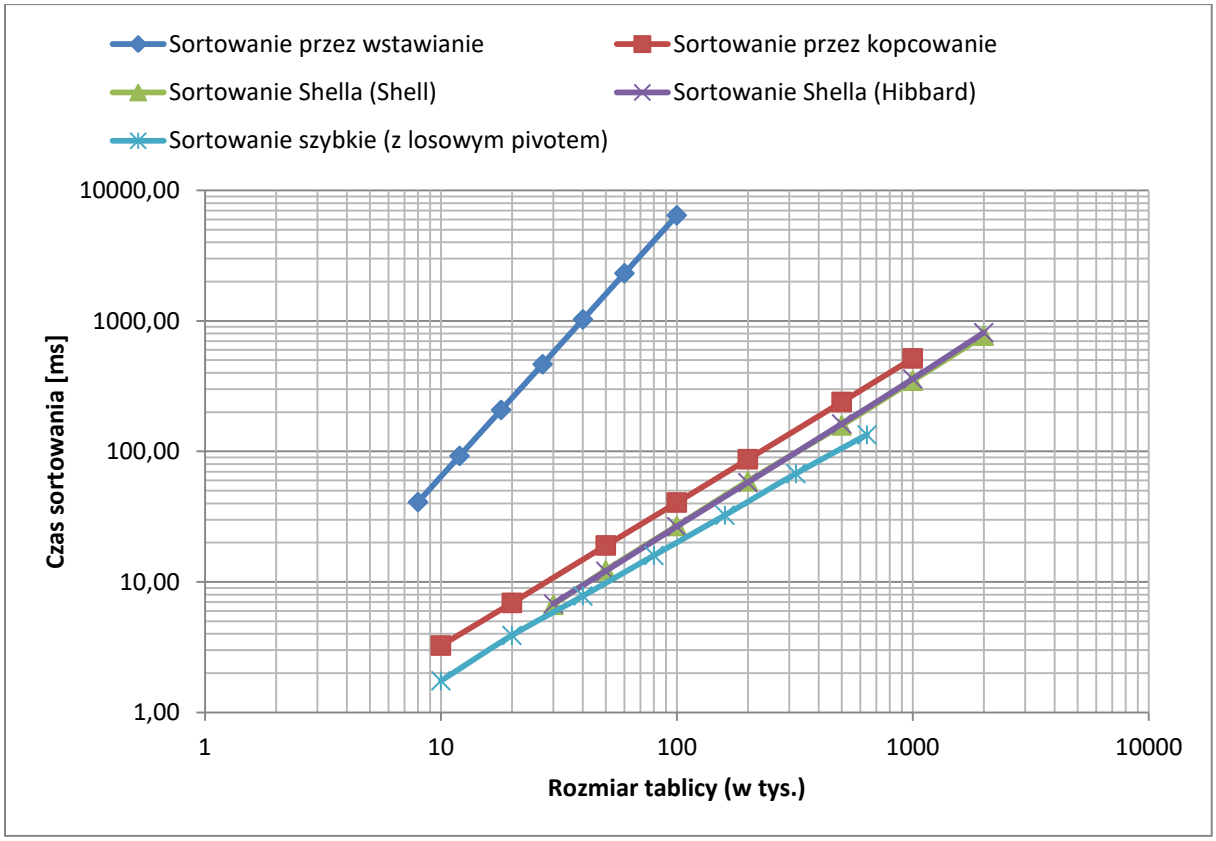
Rys. 12 Zestawienie różnych sposobów wyboru pivota na wykresie czasu sortowania w funkcji rozmiaru tablicy dla sortowania szybkiego na wstępnie niesortowanej tablicy dla danych typu int

- Zestawienie wszystkich algorytmów sortowania

Poniżej zamieszczone są wykresy porównujące wszystkie algorytmy sortowania przy niesortowanej wstępnie tablicy (oraz dla sortowania szybkiego tylko z pivotem losowym)



Rys. 13 Zestawienie różnych algorytmów sortowania na wykresie czasu sortowania w funkcji rozmiaru tablicy na wstępnie niesortowanej tablicy dla danych typu int **(skala liniowa)**



Rys. 14 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 kopcowanie 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 1 sekundy dla wielokrotnie większych 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 (poza rozmieszczeniem losowym dla typu double przy tablicy 1 miliona elementów) uzyskano zbliżony czas sortowania.
- 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 przypadku sortowania szybkiego uzyskano zgodnie z oczekiwaniami krótki czas sortowania. Należy jednak zwrócić uwagę na to, że zostały dobrane przypadki o mniejszej złożoności obliczeniowej $O(n \log n)$. Dla przypadków pesymistycznych nieuwzględnionych w trakcie badania (skrajny pivot w uporządkowanym zbiorze danych) złożoność obliczeniowa zapewne wyniosłaby $O(n^2)$, osiągając porównywalnie niską efektywność co algorytm sortowania przez wstawianie
- Zestawiając wszystkie badane algorytmy sortowania na jednym wykresie, dostrzec można (szczególnie na wykresie ze skalą logarytmiczną), 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