Badanie efektywności wybranych algorytmów sortowania ze względu na złożoność obliczeniową

Wykonawca: Jędrzej Radłowski 272927

Nazwa kursu: Algorytmy i złożoność obliczeniowa (projekt)

Tydzień i godzina zajęć: Czwartek 13;15 – 15;00, tydzień nieparzysty

Spis treści

Wprowadzenie wraz z opisem badanych algorytmów 3
Sortowanie przez wstawianie 3
Sortowanie przez wstawianie binarne 3
Sortowanie przez kopcowanie 3
Sortowanie szybkie 3
Plan eksperymentu, założenia początkowe 4
Przebieg badania 4
Wyniki algorytmów sortujących 5
Wyniki dla sortowania przez wstawianie 5
Wyniki dla sortowania przez wstawianie binarne 7
Wyniki dla sortowania przez kopcowanie 9
Wyniki dla sortowania szybkiego 11
Zależność danych wejściowych na różne algorytmy 13
Dane wejściowe posortowane rosnąco 13
Dane wejściowe posortowane malejąco 14
Dane wejściowe posortowane rosnąco w 33% 15
Dane wejściowe posortowane rosnąco w 66% 16
Dane wejściowe losowe 17
Podsumowanie i wnioski 18
Źródła 19

1. Wprowadzenie wraz z opisem badanych algorytmów
    a) Sortowanie przez wstawianie

Sortowanie przez wstawianie to jeden z najprostszych algorytmów sortowania. Algorytm ten przechodzi iteracyjnie przez całą tablicę, porównując ze sobą elementy sąsiadujące. W momencie gdy element znajdujący się dalej jest mniejszy od poprzedniego to następuje zamiana obu elementów. Elementy zamieniane są do momentu gdy dana część tablicy posortowana jest rosnąco. Algorytm ten działa najgorzej w przypadku gdy tablica posortowana jest malejąco. Dla takiego rozkładu danych w tablicy algorytm posiada złożoność kwadratową, przedstawianą jako $O\ (n^2)$. Taka sama złożoność będzie również dla przypadku średniego. Sprawia to, że algorytm ten nie jest używany dla tablic o dużych rozmiarach. Sprawdza się on dobrze w tablicach o małej ilości elementów, a także w tablicach prawie posortowanych, co czyni go skutecznym wyborem dla takich sytuacji. Algorytm ten jest także stabilny, elementy posiadające te same wartości będą miały taką samą kolejność w tablicy wejściowej i wynikowej.

    b) Sortowanie przez wstawianie binarne

Jest to modyfikacja klasycznego sortowania przez wstawianie. Wykorzystuje się tutaj wyszukiwanie binarne w celu znalezienia właściwego miejsca dla nowego elementu. Dzięki temu operacje zamiany są mocno redukowane. Dla przypadku średniego złożoność tego algorytmu, tak jak klasycznego sortowania przez wstawianie, wynosi $O\ (n^2)$. W praktyce jednak jest ono bardziej efektywne od klasycznego odpowiednika. Tutaj także sprawdza się ono dobrze w tablicach o małej ilości elementów, jak i w tablicach prawie posortowanych. Algorytm ten jest także stabilny.

    c) Sortowanie przez kopcowanie

Sortowanie to wykorzystuje strukturę danych zwaną kopcem. Jest ona oparta na drzewie, gdzie wartość węzła rodzica jest nie mniejsza od wartości węzłów jego potomków. Dla sortowania tego kopiec musi spełniać dodatkowo jeszcze jedną własność, mianowicie liście występują na ostatnim i ewentualnie przedostatnim poziomie oraz liście na ostatnim poziomie są spójnie ułożone od lewej do prawej strony. Algorytm ten polega na stopniowym usuwaniu elementów z korzenia kopca, jednocześnie uzyskując element największy na odpowiedniej pozycji, a następnie naprawie kopca. Dzięki zastosowaniu kopca złożoność algorytmiczna tego sortowania wynosi $O(\ n \log n)$, gdzie n jest wielkością kopca. Algorytm ten jest niestabilny.

    d) Sortowanie szybkie

Jest to jeden z najszybszych i najbardziej efektywnych algorytmów sortowania. Opiera się on na strategii „dziel i zwyciężaj". Dzieli on tablicę na mniejsze części, sortując je niezależnie, a później łącząc je w całość. Złożoność algorytmiczna dla przypadku średniego wynosi $O(\ n \log n)$, natomiast dla przypadku najgorszego jest to już złożoność rzędu $O\ (n^2)$. Mimo tego faktu sortowanie to jest często wykorzystywane ze względu na swoją wysoką wydajność dla różnych typów danych. Algorytm ten w większości implementacji jest niestabilny.

2. Plan eksperymentu, założenia początkowe

Komputer, na którym zostały wykonane obliczenia posiada procesor Intel Core i5-3230M, o częstotliwości taktowania 2.6 Ghz. Program został napisany w języku c++ i skompilowany dla 64 bitowej architektury. Przyjęto siedem rozmiarów tablic: 10000, 20000, 30000, 50000, 70000, 90000, 120000. Tablice te zawierały 4 bajtowe liczby całkowite, a pomiary mierzone były w milisekundach. Do zmierzenia czasu wykorzystano std::chrono::high_resolution_clock. Warto pamiętać, że różne algorytmy mają różną złożoność dla odpowiedniego początkowego rozkładu danych w tablicy, dlatego też pomiary były prowadzone dla 5 stanów początkowych: tablica posortowana rosnąco, tablica posortowana malejąco, tablica posortowana w 33%, tablica posortowana w 66% i tablica losowa. Do każdego rozkładu użyto sortowania szybkiego, gdzie dodatkowo w drugim przypadku użyto napisanej metody swap, a w trzecim i czwartym funkcji shuffle. Każdy pomiar (rozmiar tablicy i początkowy rozkład danych) został powtórzony 100 razy, tak aby ostateczny wynik był wiarygodny.

3. Przebieg badania

Program umożliwia odczyt danych z pliku tekstowego, generacje danych losowych, a także funkcję testowania. Ostatnia polega na stukrotnym pomiarze dla każdego sortowania i każdego rozmiaru tablicy. Ostateczne dane wynikowe to średnia arytmetyczna z wykonanych pomiarów i zapisywane są one do pliku tekstowego. W pozostałych przypadkach wynikiem jest konsolowe wypisanie tablicy przed sortowaniem i po sortowaniu.

4. Wyniki algorytmów sortujących

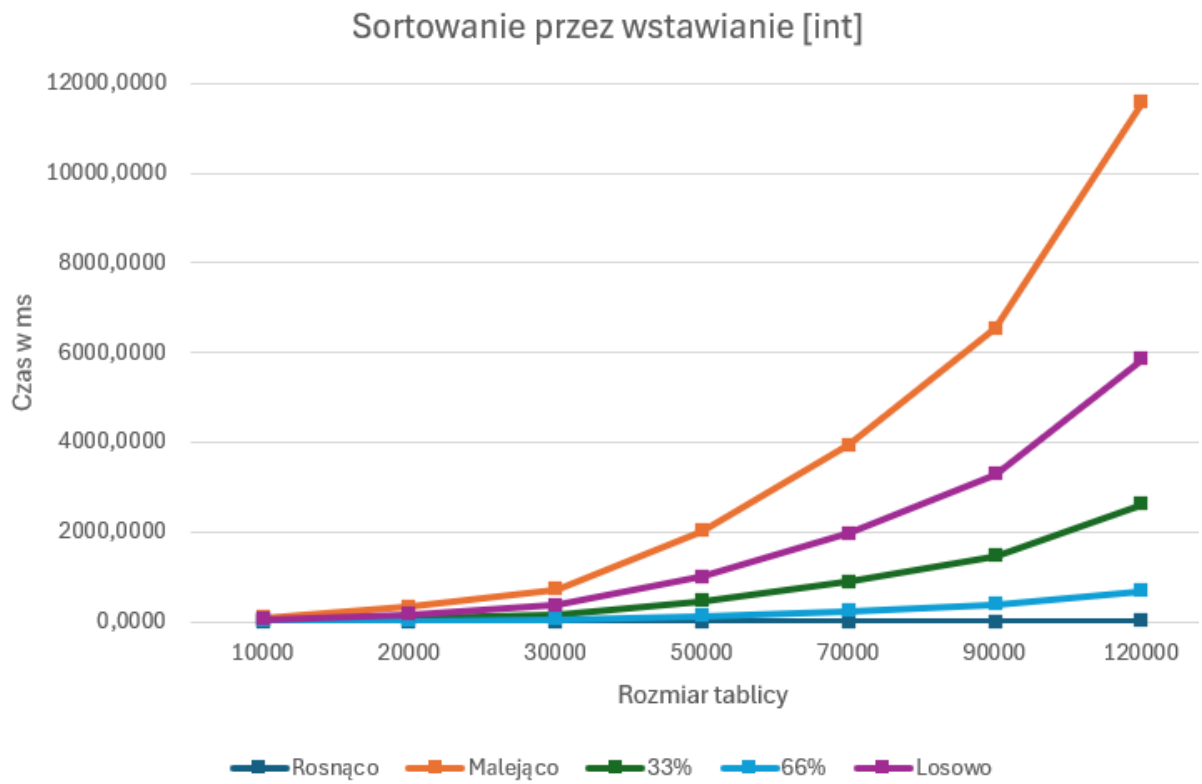
a) Wyniki dla sortowania przez wstawianie

Rozmiar tablicy	Insertion Sort [int]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,0101	82,3261	18,3588	4,6450	41,9805
20000	0,0189	319,8190	71,7251	18,2440	165,7300
30000	0,0283	723,9230	164,3550	41,8480	358,9930
50000	0,0475	2011,3000	451,3930	116,0880	1002,3100
70000	0,0672	3941,4800	886,4350	232,8700	1966,3300
90000	0,0917	6531,3000	1461,9100	376,3130	3281,9900
120000	0,1235	11580,0000	2619,4800	671,3680	5850,9600

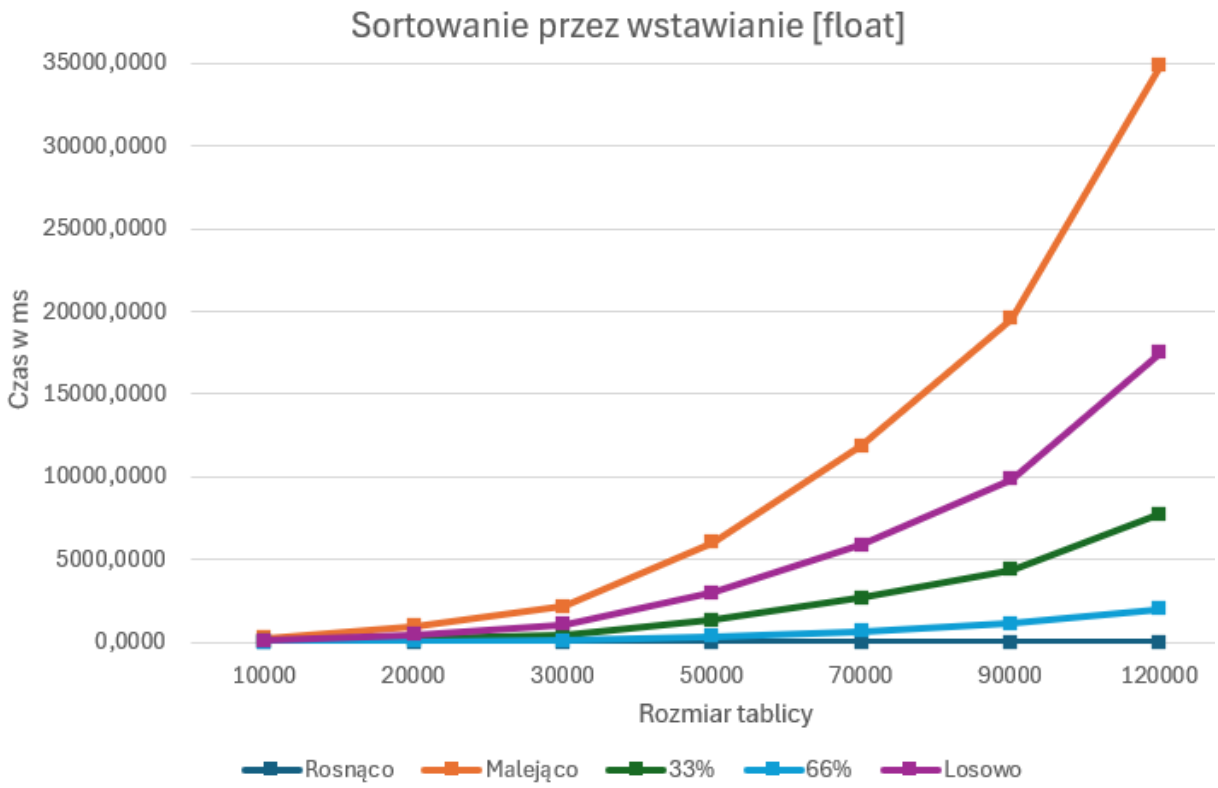
Tabela 1. Wyniki w ms dla insertion sort [int]

Rozmiar tablicy	Insertion Sort [float]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,0126	242,2580	52,5459	13,8260	123,4530
20000	0,0253	975,6600	213,5750	54,4716	481,6000
30000	0,0383	2164,1200	488,2440	127,7190	1085,9000
50000	0,0686	6066,8500	1360,7300	350,1730	3001,7600
70000	0,0903	11873,8000	2673,5000	689,9330	5914,4000
90000	0,1269	19543,7000	4378,5800	1130,9600	9863,1200
120000	0,1540	34847,1000	7732,5100	2015,6800	17504,3000

Tabela 2. Wyniki w ms dla insertion sort [float]



Wykres 1. Wyniki w ms dla insertion sort [int]



Wykres 2. Wyniki w ms dla insertion sort [float]

Wnioski: Wyniki potwierdzają charakterystykę sortowania przez wstawianie. Dla danych posortowanych rosnąco algorytm wykonuje się błyskawicznie, ponieważ nie jest konieczne wykonywanie żadnych operacji. Dla danych posortowanych malejąco zauważalny jest znaczny wzrost czasu sortowania, ponieważ algorytm musi przejść przez całą tablicę aby umieścić dany element w odpowiednim miejscu. Dla danych posortowanych częściowo widać, że im więcej elementów jest posortowanych, tym krótszy czas sortowania. W przypadku danych losowych można zauważyć zależność kwadratową. Jeżeli chodzi o wpływ typu danych na czas sortowania, to można tutaj zauważyć, że dla typu float sortowanie przez wstawianie potrzebowało dużo więcej czasu na skończenie wykonywania algorytmu. Dla typu int sortowanie trwało trzy razy krócej niż dla typu float.

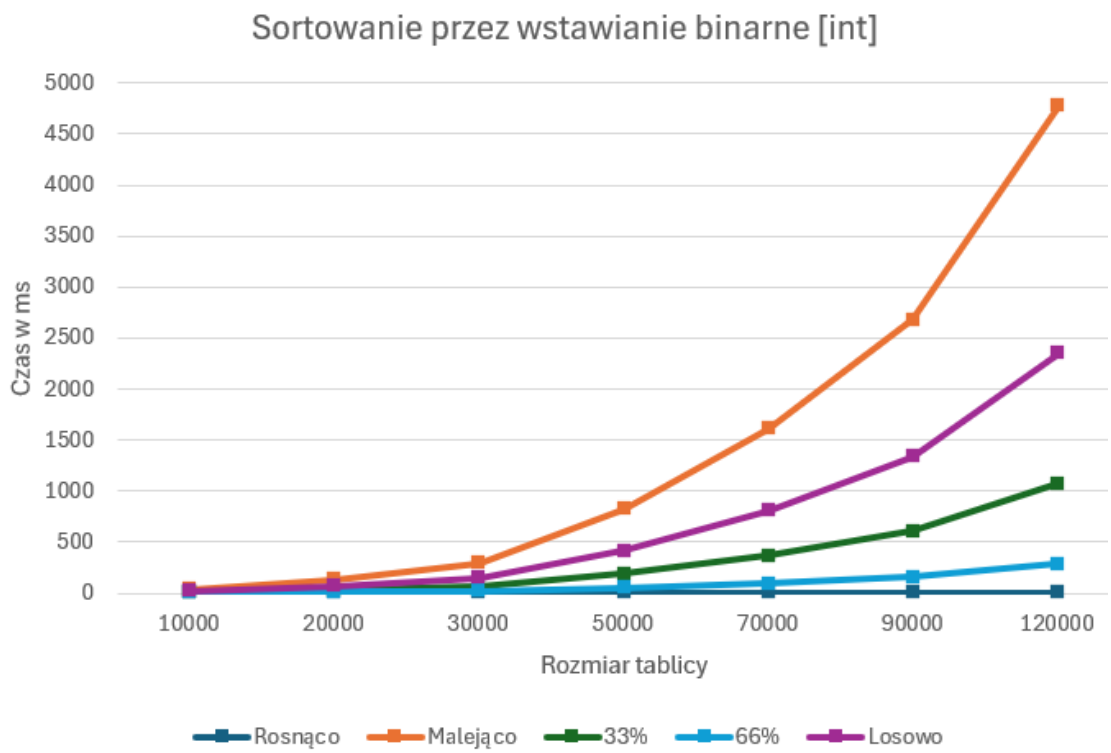
b) Wyniki dla sortowania przez wstawianie binarne

Rozmiar tablicy	Binary insertion sort [int]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,4636	34,7115	7,8542	2,3693	18,4913
20000	0,9823	131,0370	30,0137	8,9084	68,1987
30000	1,5880	294,8610	67,6253	18,5649	150,2430
50000	2,7990	819,5940	187,0760	50,0776	411,2020
70000	4,1224	1614,7400	367,1090	96,3717	808,5600
90000	5,7053	2677,1800	606,5030	157,7950	1333,9600
120000	7,7252	4772,8100	1074,2300	282,2320	2352,7800

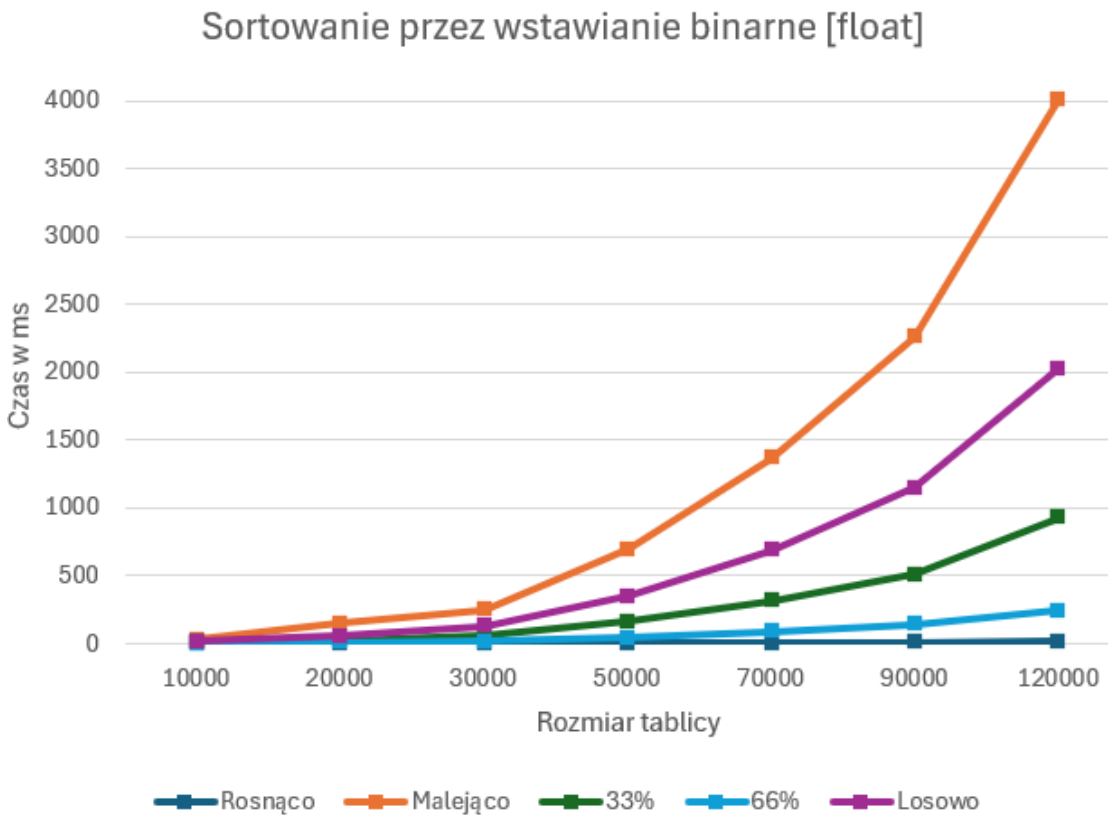
Tabela 3. Wyniki w ms dla binary insertion sort [int]

Rozmiar tablicy	Binary insertion sort [float]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,6505	28,2108	6,8322	2,2527	15,4153
20000	1,6801	151,1580	26,4671	8,1470	56,8954
30000	2,3882	250,8350	60,9261	16,7878	129,8910
50000	4,2723	699,3350	166,0700	44,9067	351,3730
70000	6,2813	1371,1200	315,2260	87,1856	689,5340
90000	8,4919	2260,4300	510,1330	140,2030	1149,5700
120000	17,5990	4017,4000	927,5230	246,2910	2026,0800

Tabela 4. Wyniki w ms dla binary insertion sort [float]



Wykres 3. Wyniki w ms dla binary insertion sort [int]



Wykres 4. Wyniki w ms dla binary insertion sort [float]

Wnioski: Wyniki ponownie odzwierciedlają charakterystykę sortowania. Tak jak przy klasycznym sortowaniu, dla danych posortowanych rosnąco algorytm wykonuje się momentalnie, dla części danych posortowanych rosnąco im więcej elementów jest posortowanych tym krótszy czas, a dla danych losowych widać zależność kwadratową. W tym sortowaniu wpływ typu danych jest już o wiele mniejszy, dodatkowo algorytm dla typu float jest szybszy niż dla typu int.

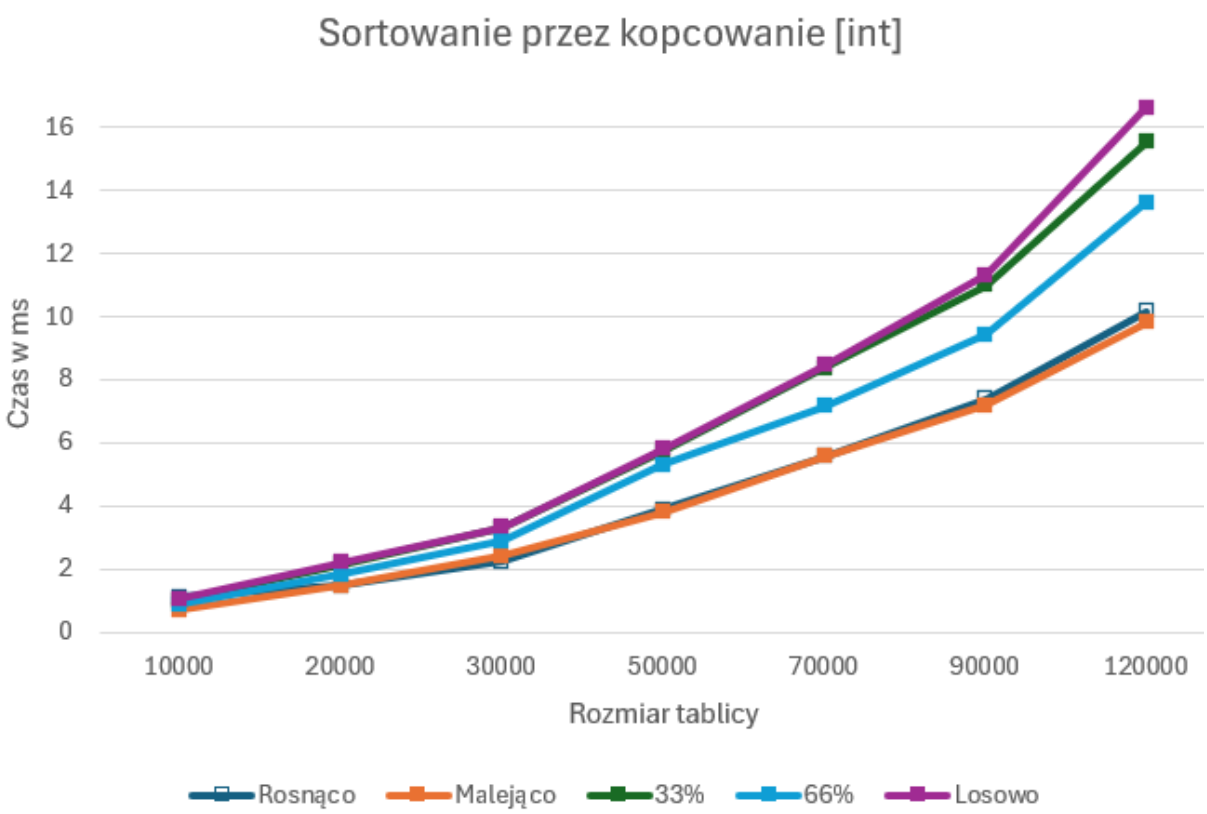
c) Sortowanie przez kopcowanie

Rozmiar tablicy	Heap sort [int]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	1,0958	0,7043	0,9788	0,8569	1,0499
20000	1,4664	1,4735	2,1346	1,8102	2,1900
30000	2,2377	2,3981	3,3010	2,8726	3,3270
50000	3,8640	3,7896	5,7142	5,2876	5,8063
70000	5,5576	5,5642	8,3636	7,1560	8,4537
90000	7,3867	7,1904	10,9801	9,4281	11,3016
120000	10,1545	9,8116	15,5185	13,6123	16,6261

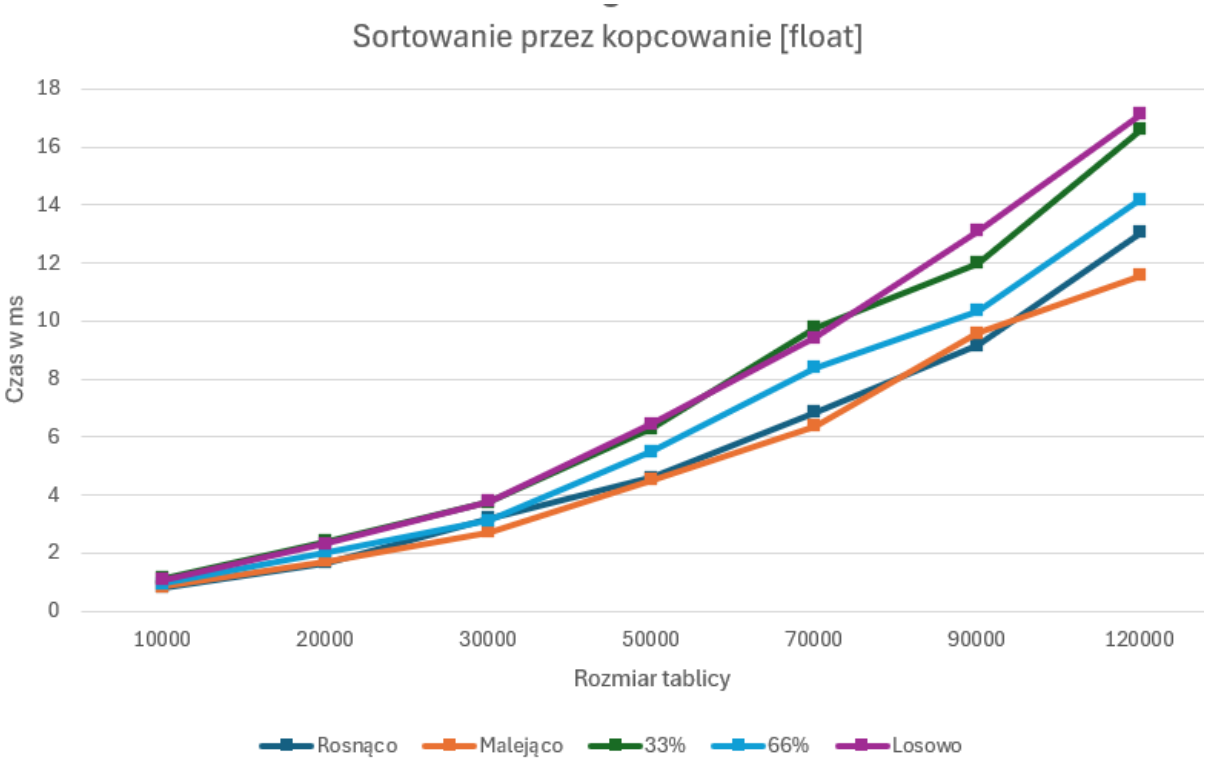
Tabela 5. Wyniki w ms dla heap sort [int]

Rozmiar tablicy	Heap sort [float]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,7974	0,8263	1,1156	0,9424	1,0856
20000	1,6567	1,6977	2,3979	2,0289	2,3302
30000	3,1817	2,7265	3,7606	3,1090	3,7700
50000	4,5900	4,5228	6,2980	5,5054	6,4407
70000	6,8505	6,3628	9,7593	8,3841	9,4069
90000	9,1428	9,5822	11,9933	10,3335	13,0904
120000	13,0596	11,5699	16,5879	14,1948	17,1190

Tabela 6. Wyniki w ms dla heap sort [float]



Wykres 5. Wyniki w ms dla heap sort [int]



Wykres 6. Wyniki w ms dla heap sort [float]

Wyniki: Dla danych posortowanych rosnąco i malejąco czasy sortowań są zbliżone i o wiele krótsze od poprzednich algorytmów. Dla danych częściowo posortowanych czasy są nieco wyższe niż dla tablic posortowanych lecz dalej relatywnie niskie. Dla danych losowych zauważyć można najwyższe wartości czasów sortowania, co także jest spodziewanym wynikiem. Zauważyć można, że w tym algorytmie wpływ typu danych na czas wykonania sortowania jest niewielki. Jest to różnica na poziomie 1ms, z przewagą dla typu int.

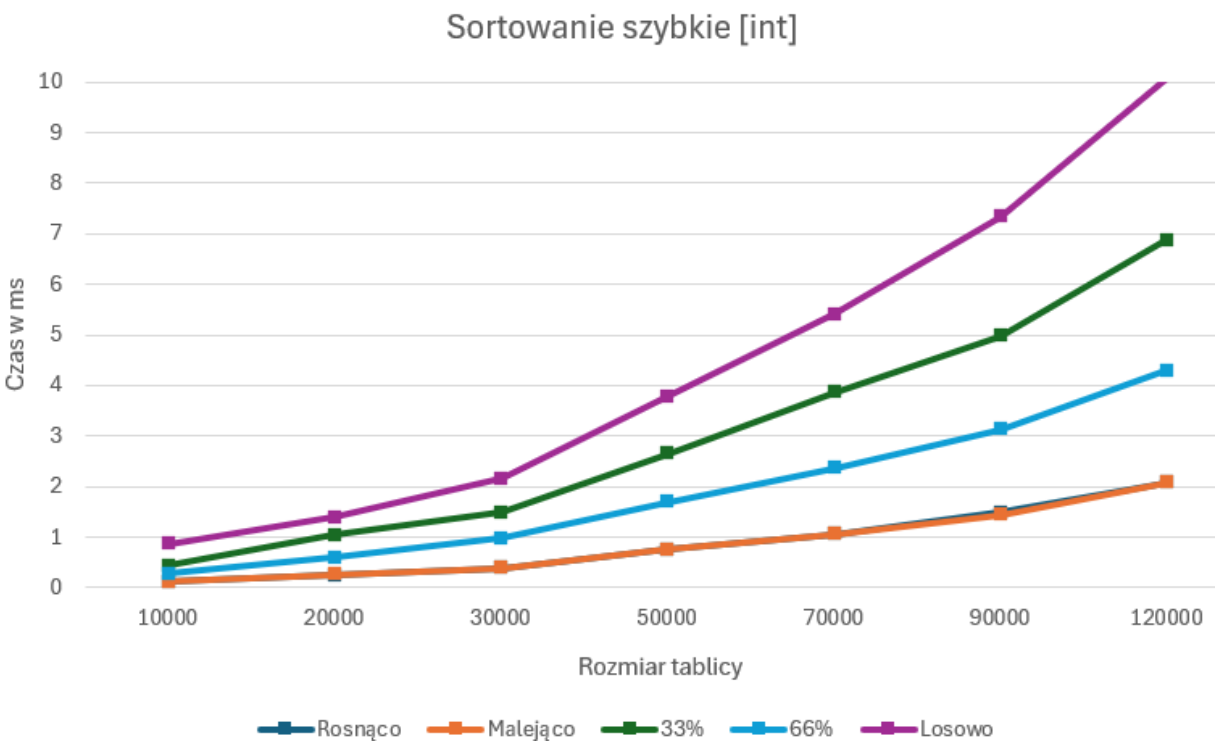
d) Sortowanie szybkie

Rozmiar tablicy	Quick sort [int]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,1187	0,1196	0,4415	0,2710	0,8588
20000	0,2422	0,2549	1,0363	0,5860	1,3913
30000	0,3808	0,3873	1,4814	0,9700	2,1573
50000	0,7459	0,7452	2,6532	1,6866	3,7811
70000	1,0588	1,0609	3,8550	2,3573	5,4018
90000	1,4873	1,4431	4,9759	3,1244	7,3344
120000	2,0711	2,0718	6,8758	4,2922	10,0834

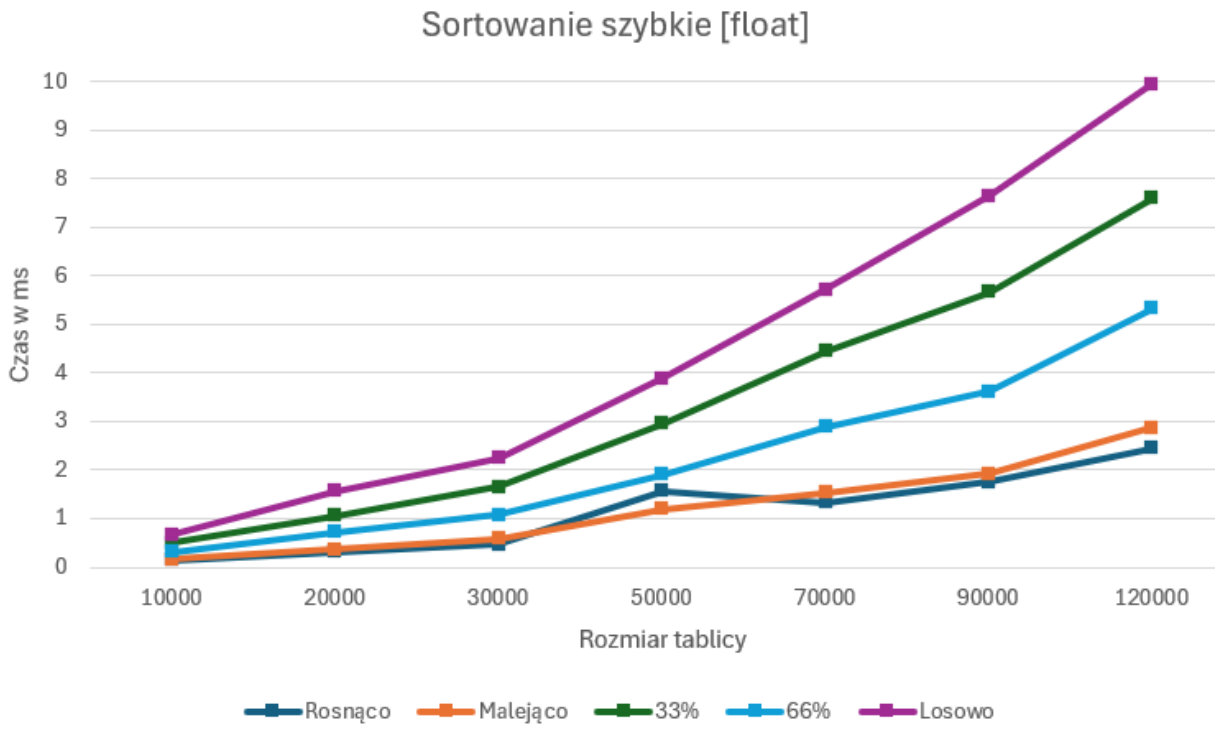
Tabela 7. Wyniki w ms dla quick sort [int]

Rozmiar tablicy	Quick sort [float]				
	Rosnąco	Malejąco	33%	66%	Losowo
	Czas [ms]				
10000	0,1412	0,1673	0,4951	0,3193	0,6759
20000	0,3143	0,3598	1,0649	0,7281	1,5645
30000	0,4605	0,5782	1,6446	1,0704	2,2486
50000	1,5742	1,1858	2,9516	1,9057	3,8830
70000	1,3264	1,5245	4,4411	2,8932	5,7230
90000	1,7516	1,9296	5,6686	3,6100	7,6535
120000	2,4482	2,8753	7,5898	5,3307	9,9567

Tabela 8. Wyniki w ms dla quick sort [float]



Wykres 7. Wyniki w ms dla quick sort [int]



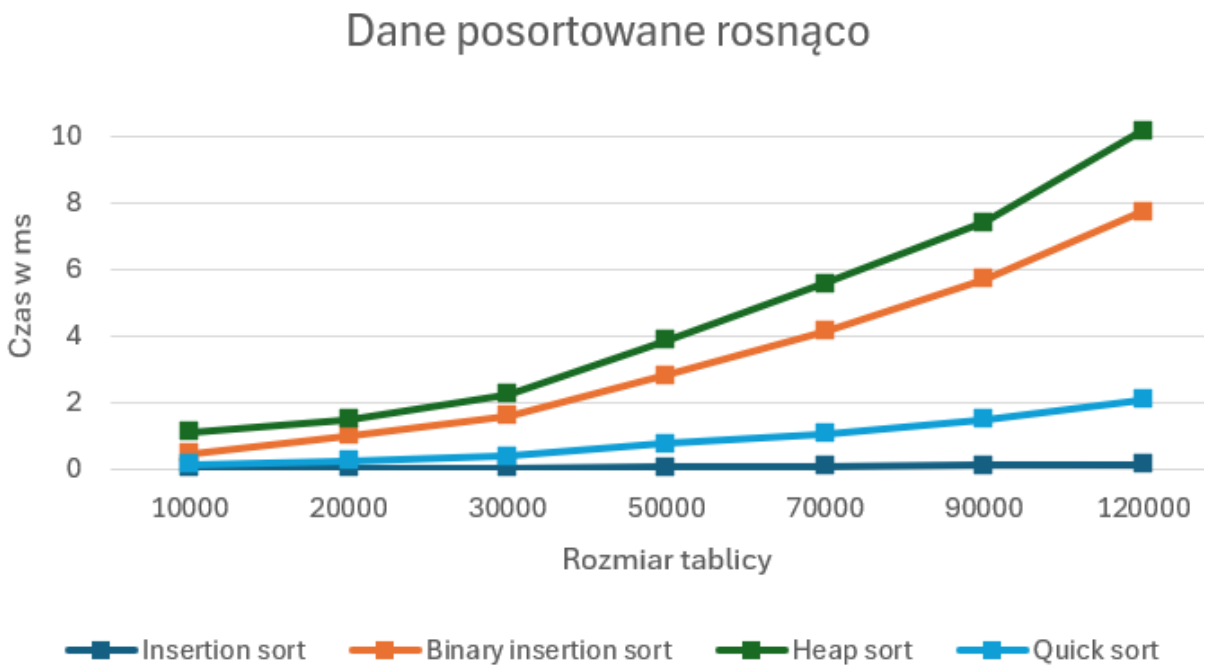
Wykres 8. Wyniki w ms dla quick sort [float]

Wnioski: Dla tablic posortowanych czasy sortowania są bardzo krótkie i zbliżone. Dla danych częściowo posortowanych czasy są nieco wyższe lecz dalej relatywnie niskie. Dla danych losowych zauważyć można najwyższe wartości czasów sortowania, co jest zgodne z oczekiwaniami. Podobnie jak dla sortowania przez kopcowanie zauważyć można, że w tym algorytmie wpływ typu danych na czas wykonania sortowania jest niewielki. Jest to znowu różnica na poziomie 1ms, z przewagą dla typu int.

5. Zależność danych wejściowych na różne algorytmy
   a) Dane wejściowe posortowane rosnąco

Rozmiar tablicy	Insertion sort	Binary insertion sort	Heap sort	Quick sort
10000	0,0101	0,4636	1,0958	0,1187
20000	0,0189	0,9823	1,4664	0,2422
30000	0,0283	1,5880	2,2377	0,3808
50000	0,0475	2,7990	3,8640	0,7459
70000	0,0672	4,1224	5,5576	1,0588
90000	0,0917	5,7053	7,3867	1,4873
120000	0,1235	7,7252	10,1545	2,0711

Tabela 9. Wyniki dla danych posortowanych rosnąco



Wykres 9. Wyniki dla danych posortowanych rosnąco

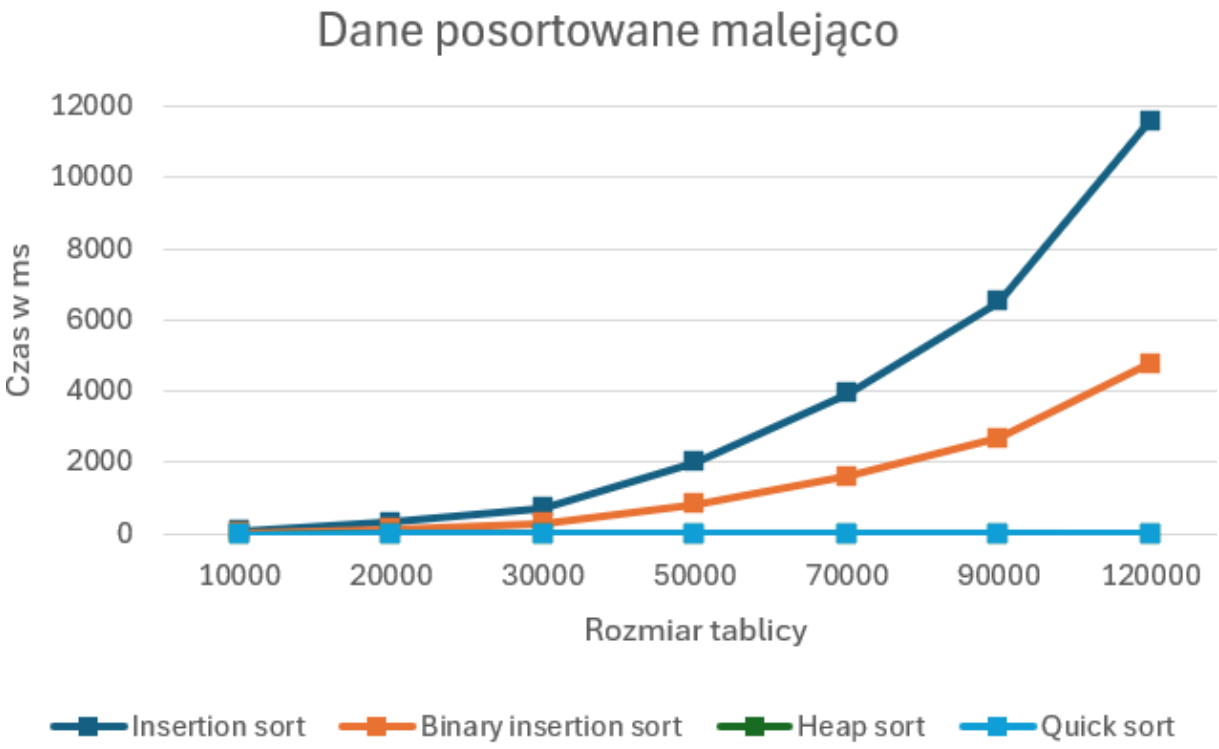
Wnioski: Zauważyć można, że dla takich danych wejściowych najlepsze jest sortowanie przez wstawianie, ponieważ nie wykonują się tam żadne operacje. Zaraz później jest sortowanie szybkie, które zachowuje swoją wysoka wydajność. Następnie plasuje się sortowanie przez wstawianie binarne. Zajmuje ono dłużej ze względu na

obliczanie odpowiedniego miejsca dla każdego elementu. Ostatnie jest sortowanie przez kopcowanie, lecz nadaj wykonuje się ono bardzo szybko.

b) Dane wejściowe posortowane malejąco

Rozmiar tablicy	Insertion sort	Binary insertion sort	Heap sort	Quick sort
10000	82,3261	34,7115	0,7043	0,1196
20000	319,8190	131,0370	1,4735	0,2549
30000	723,9230	294,8610	2,3981	0,3873
50000	2011,3000	819,5940	3,7896	0,7452
70000	3941,4800	1614,7400	5,5642	1,0609
90000	6531,3000	2677,1800	7,1904	1,4431
120000	11580,0000	4772,8100	9,8116	2,0718

Tabela 10. Wyniki dla danych posortowanych malejąco



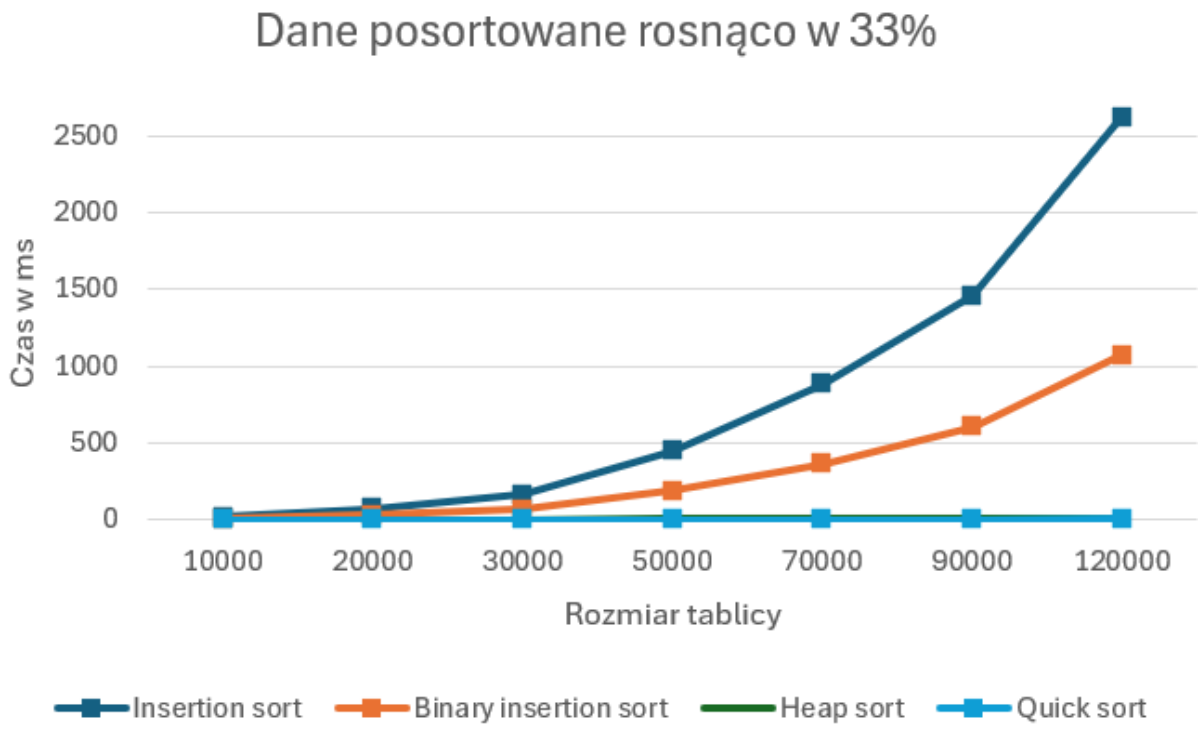
Wykres 10. Wyniki dla danych posortowanych malejąco

Wnioski: Najlepsze dla tego rozkładu danych jest sortowanie szybkie, zaraz później sortowanie przez kopcowanie. W porównaniu do dwóch pozostałych algorytmów, te dwa wykonują się niemal natychmiastowo. Widać tutaj też przewagę sortowania przez wstawianie binarnego nad jego klasyczną wersją.

c) Dane wejściowe posortowane rosnąco w 33%

Rozmiar tablicy	Insertion sort	Binary insertion sort	Heap sort	Quick sort
10000	18,3588	7,8542	0,9788	0,4415
20000	71,7251	30,0137	2,1346	1,0363
30000	164,3550	67,6253	3,3010	1,4814
50000	451,3930	187,0760	5,7142	2,6532
70000	886,4350	367,1090	8,3636	3,8550
90000	1461,9100	606,5030	10,9801	4,9759
120000	2619,4800	1074,2300	15,5185	6,8758

Tabela 11. Wyniki dla danych posortowanych rosnąco w 33%



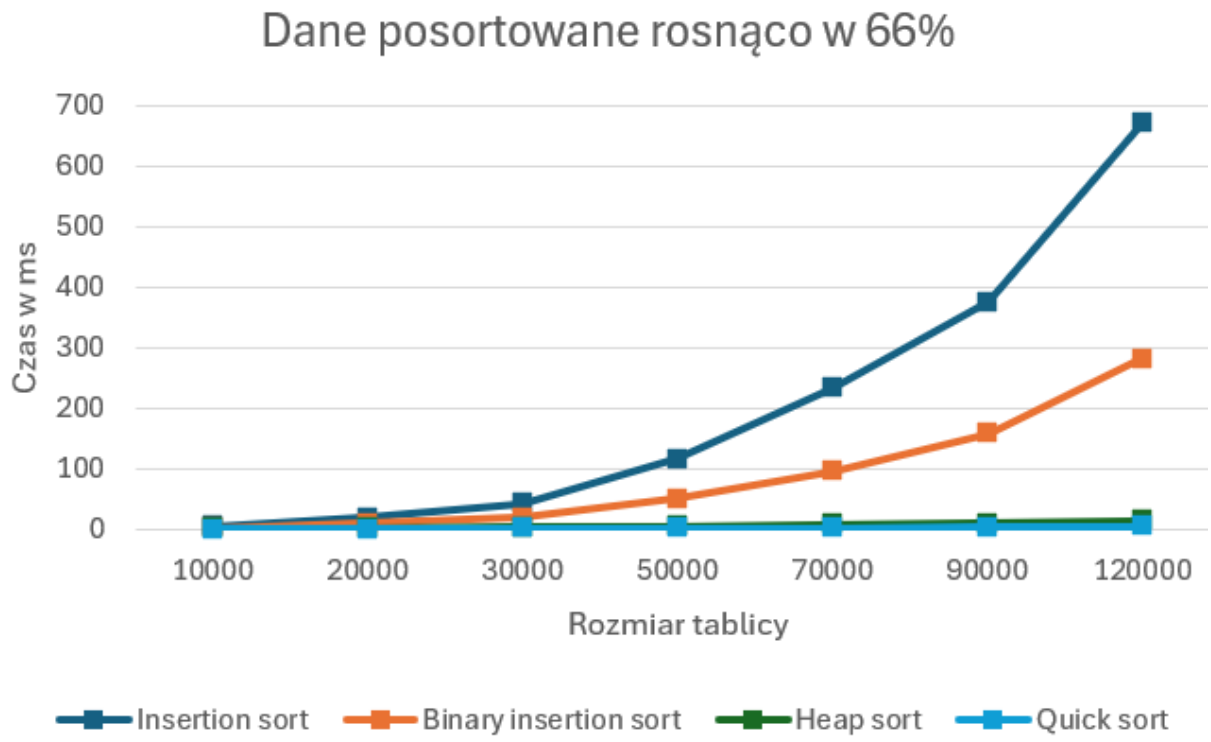
Wykres 11. Wyniki dla danych posortowanych rosnąco w 33%

Wnioski: Podobnie jak poprzednio, sortowanie szybkie wykonuje się najkrócej, a sortowanie przez kopcowanie plasuje się zaraz za nim. Znowu oba te algorytmy wykonują się o wiele szybciej od sortowań przez wstawianie. Zauważalna jest także różnica między sortowaniem przez wstawianie binarne i jego klasyczną wersją.

d) Dane wejściowe posortowane rosnąco w 66%

Rozmiar tablicy	Insertion sort	Binary insertion sort	Heap sort	Quick sort
10000	4,6450	2,3693	0,8569	0,2710
20000	18,2440	8,9084	1,8102	0,5860
30000	41,8480	18,5649	2,8726	0,9700
50000	116,0880	50,0776	5,2876	1,6866
70000	232,8700	96,3717	7,1560	2,3573
90000	376,3130	157,7950	9,4281	3,1244
120000	671,3680	282,2320	13,6123	4,2922

Tabela 12. Wyniki dla danych posortowanych rosnąco w 66%



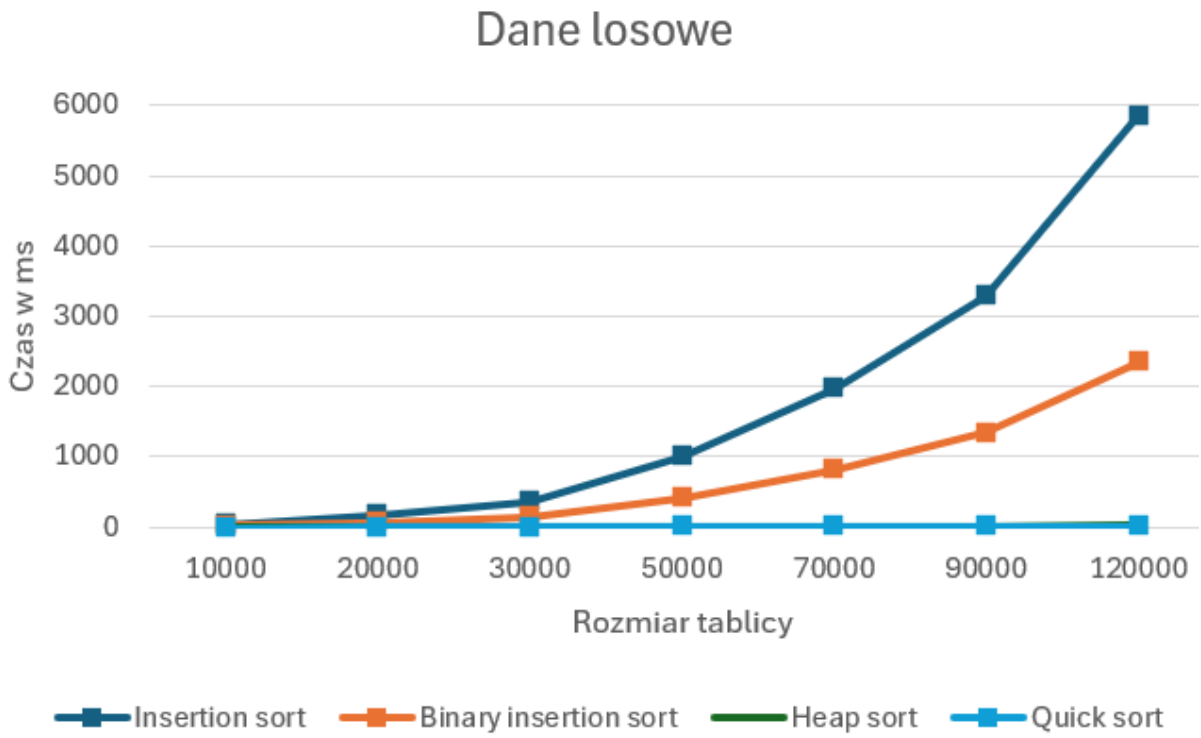
Wykres 12. Wyniki dla danych posortowanych rosnąco w 66%

Wnioski: Podobnie jak poprzednio, sortowanie szybkie i przez kopcowanie wykonują się najszybciej i zauważalna jest widoczna przewaga tych algorytmów przy dużym rozmiarze tablicy. Ponownie sortowanie przez wstawianie binarne jest o wiele szybsze od klasycznej wersji tego sortowania. Widać natomiast, że oba warianty sortowania przez wstawianie wykonują się o wiele szybciej w takim rozkładzie danych, niż w poprzednich dwóch przypadkach.

e) Dane losowe

Rozmiar tablicy	Insertion sort	Binary insertion sort	Heap sort	Quick sort
10000	41,9805	18,4913	1,0499	0,8588
20000	165,7300	68,1987	2,1900	1,3913
30000	358,9930	150,2430	3,3270	2,1573
50000	1002,3100	411,2020	5,8063	3,7811
70000	1966,3300	808,5600	8,4537	5,4018
90000	3281,9900	1333,9600	11,3016	7,3344
120000	5850,9600	2352,7800	16,6261	10,0834

Tabela 13. Wyniki dla danych losowych



Wykres 13. Wyniki dla danych losowych

Wnioski: Sortowanie szybkie ponownie wygrywa. Charakteryzuje się ono wysoką wydajnością we wszystkich testowanych przypadkach. Następne jest sortowanie przez kopcowanie, które również dobrze radzi sobie dla danych losowych i dla dużych rozmiarów tablic. Kolejne jest sortowanie przez wstawianie binarne. Mimo, że gorsze od pozostałych to dalej pozostaje dużo szybsze od ostatniego sortowania jakim jest klasyczne sortowanie przez wstawianie. Znowu sortowanie szybkie i przez kopcowanie w porównaniu do dwóch pozostałych wykonują się niemal natychmiastowo.

6. Podsumowanie i wnioski

Badania te okazały się być bardzo czasochłonnym zadaniem. Przez typ sortowań, a także dosyć wolną jednostkę, na której przeprowadzone zostały badania otrzymanie wyników zajęło dużo czasu.

Zgodnie z oczekiwaniami, sortowanie przez wstawianie osiągało najdłuższe rezultaty, było też najbardziej podatne na zmianę typu danych z int na float. Jego złożoność kwadratowa dla najgorszego i średniego przypadku sprawia, że algorytm ten staje się niepraktycznym, jedynie w momencie gdy tablica zawiera mało danych, lub jest już posortowana rosnąco jest on warty rozważenia. Podobnie jego wersja binarna, mimo że lepsza od klasycznej, dalej zajmuje bardzo dużo czasu, w porównaniu do kolejnych dwóch algorytmów, w przypadku średnim i najgorszym. Kolejnym algorytmem jest sortowanie przez kopcowanie. Uzyskuje ono o wiele lepsze wyniki od poprzednich dwóch, a dodatkowo fakt, że początkowe ułożenie danych nie ma tak dużego znaczenia sprawia, że algorytm ten jest efektywny. Najszybszym algorytmem jednak jest sortowanie szybkie. Uzyskało ono bardzo satysfakcjonujące wyniki we wszystkich testowanych rozkładach danych.

7. Źródła

https://en.wikipedia.org/wiki/Insertion_sort

https://en.wikipedia.org/wiki/Heapsort

https://en.wikipedia.org/wiki/Quicksort