Wstęp

Sortowanie jest jednym z częściej występujących problemów w informatyce, dlatego znajomość algorytmów pozwalających na sortowanie wartości jest umiejętnością kluczową. Prawidłowo nadany porządek wśród elementów danych, którymi dysponujemy ułatwia i przyspiesza wiele innych później wykonywanych operacji (np. operacje przeszukiwania i wyszukiwanie odpowiednich elementów). Obecnie znanych jest wiele algorytmów umożliwiających porządkowanie danych różniących się szybkością działania (złożoność czasowa), złożonością pamięciową, stabilnością czy możliwością sortowania w miejscu. Dodatkowo każdy z algorytmów w różny sposób radzi sobie z uporządkowaniem różnego typu danych oraz z różnym rozmiarem danych wejściowych.

W celu wykonania zadania zaimplementowano cztery wybrane algorytmy sortowania (sortowanie przez wybieranie, sortowanie przez wstawianie, sortowanie przez kopcowanie oraz sortowanie szybkie). Każdy z nich zastosowano dla różnego rodzaju rozkładu danych wejściowych (losowego, malejącego, rosnącego oraz V-kształtnego) oraz dla różnej wielkości danych wejściowych (szesnaście różnych wielkości zbioru danych do posortowania). Następnie aby zmierzyć efektywność sortowania w każdym przypadku zmierzono czas, który był potrzebny, aby uporządkować dane. Wyniki przedstawiono w tabelach oraz na wykresach.

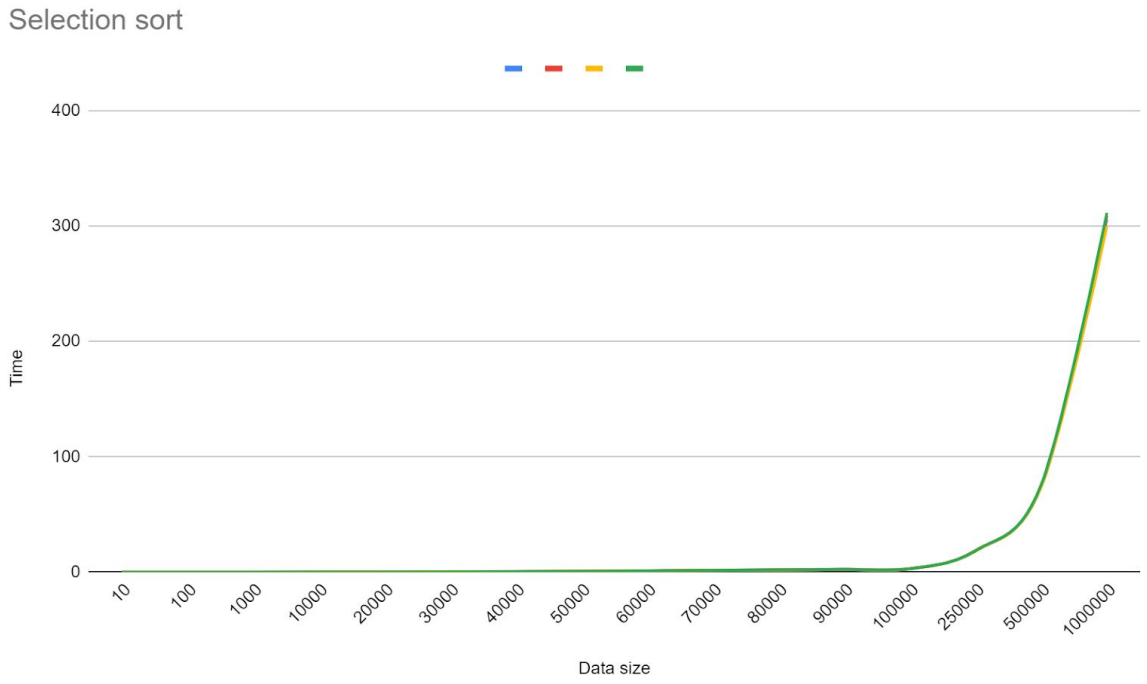
1. Sortowanie przez wybieranie (ang. *selection sort*)

Sortowanie przez wybieranie polega na wyszukiwaniu elementu mającego znaleźć się na danej pozycji, dla przykładu aby uzyskać dane posortowane rosnąco, należy znaleźć element minimalny i zamienić go miejscami z najniższym indeksem tablicy, następny minimalny element znajdzie się na pozycji w następnym indeksie ponieważ w pierwszym indeksie już znajduje się najmniejszy element zestawu danych. Operacja ta jest powtarzana dla każdego elementu tablicy, przez co jego złożoność obliczeniowa wynosi O(n2). Algorytm sortowania przez wybieranie działa z tą samą wydajnością niezależnie od typu rozkładu danych, jednak z niewielką różnicą działa najlepiej dla rozkładu malejącego, a najgorzej dla rozkładu v kształtnego.

Tabela 1. Czas działania algorytmu selection sort (w sekundach) dla różnego typu oraz różnej wielkości danych wejściowych.

	Random	Increasing	Decreasing	V-Shape
10	0	0	0	0
100	0	0	0	0
1000	0	0	0	0,001
10000	0,031	0,031	0,044	0,031
20000	0,119	0,121	0,137	0,122
30000	0,27	0,271	0,267	0,272
40000	0,478	0,484	0,474	0,478

50000	0,76	0,746	0,736	0,744
60000	1,088	1,079	1,058	1,069
70000	1,458	1,569	1,433	1,459
80000	1,986	1,916	1,88	1,902
90000	2,361	2,421	2,43	2,412
100000	2,982	3,006	2,997	2,979
250000	18,625	18,654	18,342	18,597
500000	74,543	75,145	73,824	75,69
1000000	305,878	308,825	300,049	311,552



Wykres 1. Algorytm selection sort - wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.
[kolor niebieski - rozkład losowy (ang. random), kolor czerwony - rozkład rosnący, kolor żółty - rozkład malejący, kolor zielony - rozkład V- kształtny]

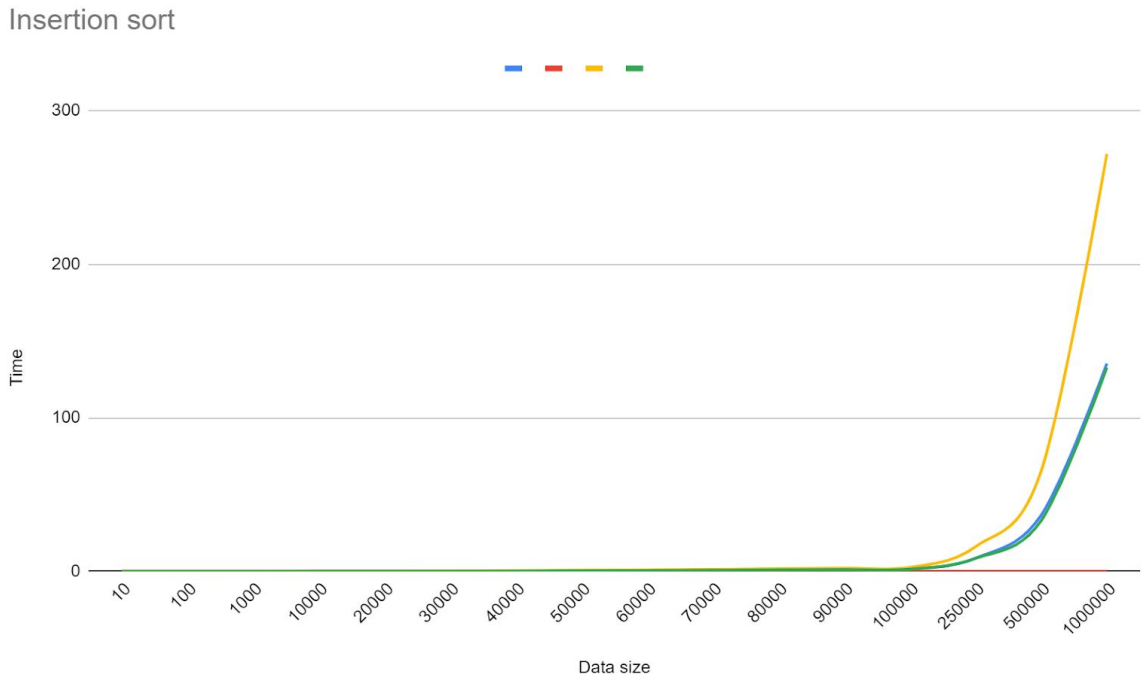
2. Sortowanie przez wstawianie (ang. *insertion sort*)

Sortowanie przez wstawianie polega na wstawieniu danego elementu na odpowiednią pozycję co czyni ten algorytm efektywnym dla małych zestawów danych bądź posortowanych tablic. Rozpatrując sytuację gdzie dane są dwa zestawy danych, tablica posortowana i tablica nieposortowana należy wybrać dowolny element ze zbioru nieposortowanego i umieścić go w odpowiednim miejscu tablicy posortowanej i należy

wykonać to dla każdego elementu tablicy nieposortowanej (Odzwierciedla to sytuację dobierania kart do aktualnego zestawu na ręce) przez co złożoność tego algorytmu wynosi O(n2). Natomiast algorytm działa znacznie efektywniej dla zestawów losowych, rosnących czy rozkładu v kształtnego niż sortowanie przez wybieranie i jest to różnica ponad dwukrotna. Dla zestawu już posortowanego algorytm kończy swoje działanie prawie natychmiastowo. Zestaw z rozkładem malejącym działa już w podobnym czasie co sortowanie przez wybieranie, różnica wynosi tylko około 30 sekund na korzyść sortowania przez wstawianie i czyni ten algorytm najmniej wydajnym dla tego rozkładu danych.

Tabela 2. Czas działania algorytmu insertion sort (w sekundach) dla różnego typu oraz różnej wielkości danych wejściowych.

	Random	Increasing	Decreasing	V-Shape
10	0	0	0	0
100	0	0	0	0
1000	0,001	0	0,001	0
10000	0,014	0	0,026	0,014
20000	0,057	0	0,103	0,064
30000	0,143	0	0,233	0,124
40000	0,23	0	0,407	0,21
50000	0,368	0	0,647	0,332
60000	0,48	0	0,922	0,474
70000	0,727	0	1,258	0,657
80000	0,961	0	1,651	0,846
90000	1,212	0	2,1	1,068
100000	1,39	0	2,543	1,362
250000	8,565	0	16,059	8,381
500000	36,533	0,001	65,277	33,042
1000000	135,288	0,001	271,81	132,681



Wykres 2. Algorytm insertion sort - wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.
[kolor niebieski - rozkład losowy (ang. random), kolor czerwony - rozkład rosnący, kolor żółty - rozkład malejący, kolor zielony - rozkład V- kształtny]

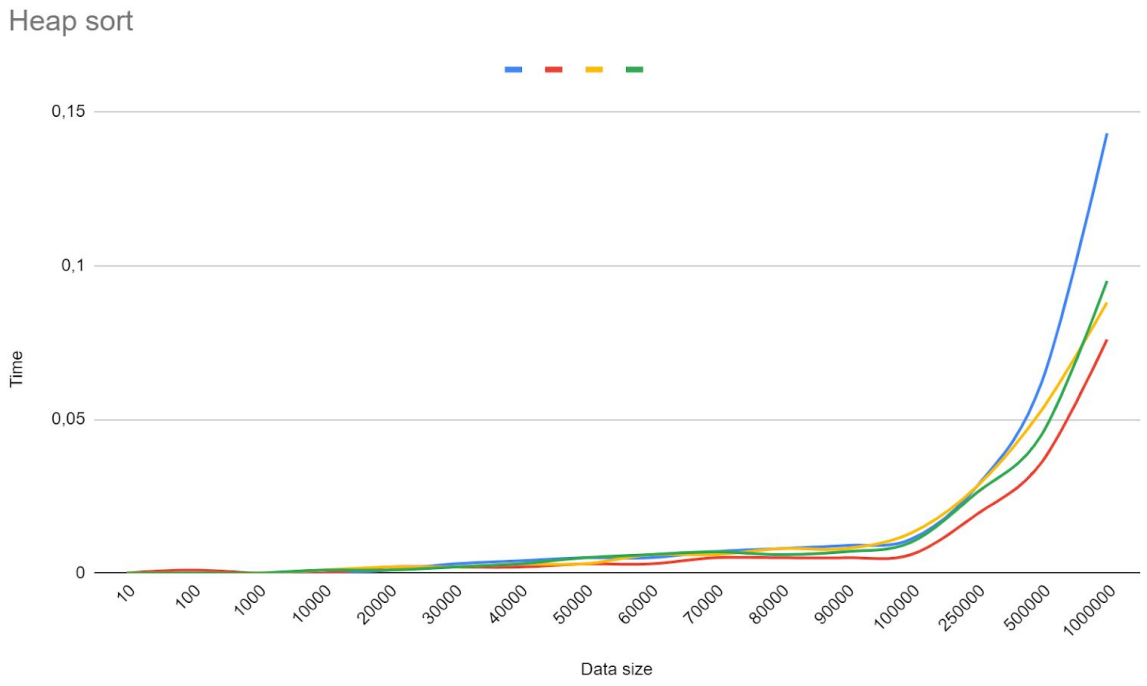
3. Sortowanie przez kopcowanie (ang. *heap sort*)

Sortowanie przez kopcowanie korzysta ze struktury kopca w celu utrzymania danego rozkładu danych. Kopiec jest to zupełne drzewo binarne w którym elementy rozkładają się od najmniejszego w korzeniu do największego w ostatnim liściu bądź odwrotnie w zależności od potrzeby uzyskania danego rozkładu danych. Mechanizm działania tego sortowania nie różni się od sortowania przez wybieranie ale wykorzystywa struktura danych wymusza pewne ograniczenia ilości porównywanych elementów czego wynikiem jest złożoność obliczeniowa O(nlogn). Wydajność tego algorytmu dla poszczególnych rozkładów danych jest porównywalna, wyróżnia się jedynie wydajność dla losowego rozkładu danych ale różnica jest pomijalnie mała, choć nadal czyni ten algorytm najmniej wydajnym dla losowego rozkładu danych, natomiast najbardziej efektywny jest dla zestawu już posortowanego.

Tabela 3. Czas działania algorytmu heap sort (w sekundach) dla różnego typu oraz różnej wielkości danych wejściowych.

	Random	Increasing	Decreasing	V-Shape
10	0	0	0	0
100	0	0,001	0	0
1000	0	0	0	0

10000	0	0	0,001	0,001
20000	0,001	0,002	0,002	0,001
30000	0,003	0,002	0,002	0,002
40000	0,004	0,002	0,003	0,003
50000	0,005	0,003	0,003	0,005
60000	0,005	0,003	0,006	0,006
70000	0,007	0,005	0,006	0,007
80000	0,008	0,005	0,008	0,006
90000	0,009	0,005	0,008	0,007
100000	0,011	0,006	0,013	0,01
250000	0,028	0,019	0,028	0,026
500000	0,062	0,036	0,053	0,045
1000000	0,143	0,076	0,088	0,095



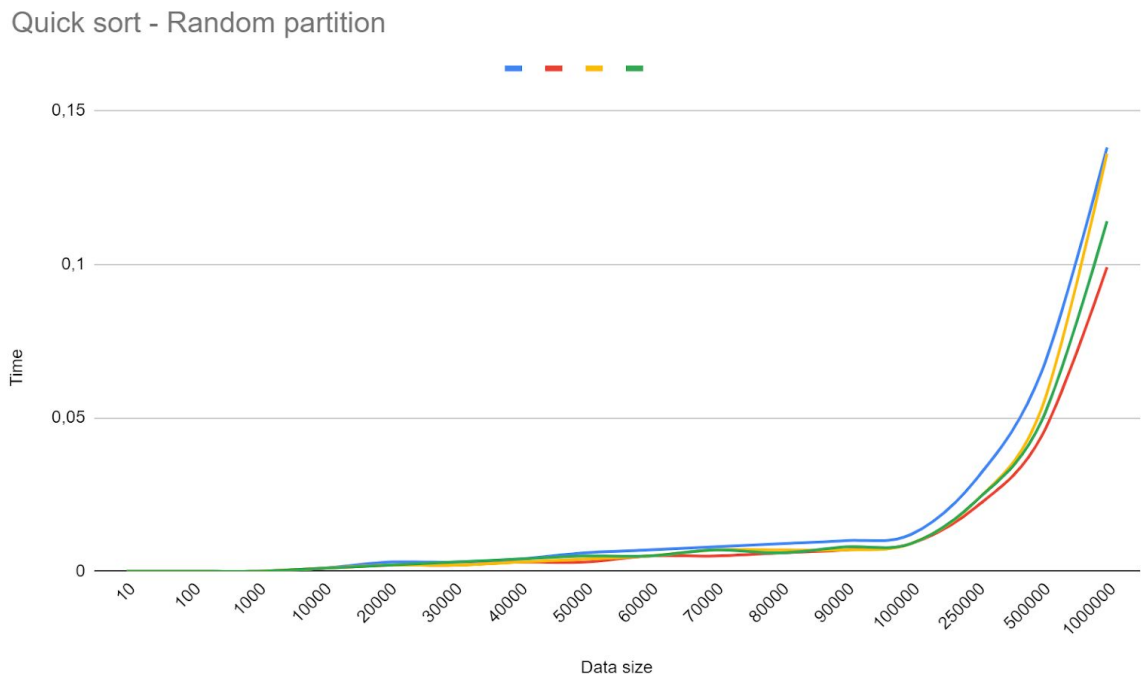
Wykres 3. Algorytm heap sort - wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu.
[kolor niebieski - rozkład losowy (ang. random), kolor czerwony - rozkład rosnący, kolor żółty - rozkład malejący, kolor zielony - rozkład V- kształtny]

4. Sortowanie szybkie (ang. *quick sort*)

Sortowanie szybkie polega na wyborze dowolnego elementu rozdzielającego z sortowanej tablicy, elementy mniejsze znajdą się w jednej tablicy, a elementy większe znajdą się w drugiej, procedura ta powtarzana jest rekurencyjnie dla każdej utworzonej tablicy do momentu uzyskania tablic jednoelementowych które następnie są ze sobą łączone zgodnie z danym rozkładem danych. Średnia złożoność obliczeniowa tego algorytmu wynosi O(nlogn) co widać w poniższej tabeli, natomiast przypadek pesymistyczny to O(n2) co sprawia że algorytm może być podatny na niektóre zestawy danych. Wydajność algorytmu dla implementacji algorytmu z losowym elementem rozdzielającym jest prawie jednakowa dla każdego ze sprawdzonych zestawów danych, najlepiej działa dla rosnącego rozkładu danych, a najgorzej dla zestawu losowego.

Tabela 4. Czas działania algorytmu quick sort w wariancie z losowym elementem rozdzielającym (w sekundach) dla różnego typu oraz różnej wielkości danych wejściowych.

	Random	Increasing	Decreasing	V-Shape
10	0	0	0	0
100	0	0	0	0
1000	0	0	0	0
10000	0,001	0,001	0,001	0,001
20000	0,003	0,002	0,002	0,002
30000	0,003	0,002	0,002	0,003
40000	0,004	0,003	0,003	0,004
50000	0,006	0,003	0,004	0,005
60000	0,007	0,005	0,005	0,005
70000	0,008	0,005	0,007	0,007
80000	0,009	0,006	0,007	0,006
90000	0,01	0,007	0,007	0,008
100000	0,012	0,009	0,009	0,009
250000	0,03	0,021	0,023	0,023
500000	0,065	0,044	0,053	0,049
1000000	0,138	0,099	0,136	0,114



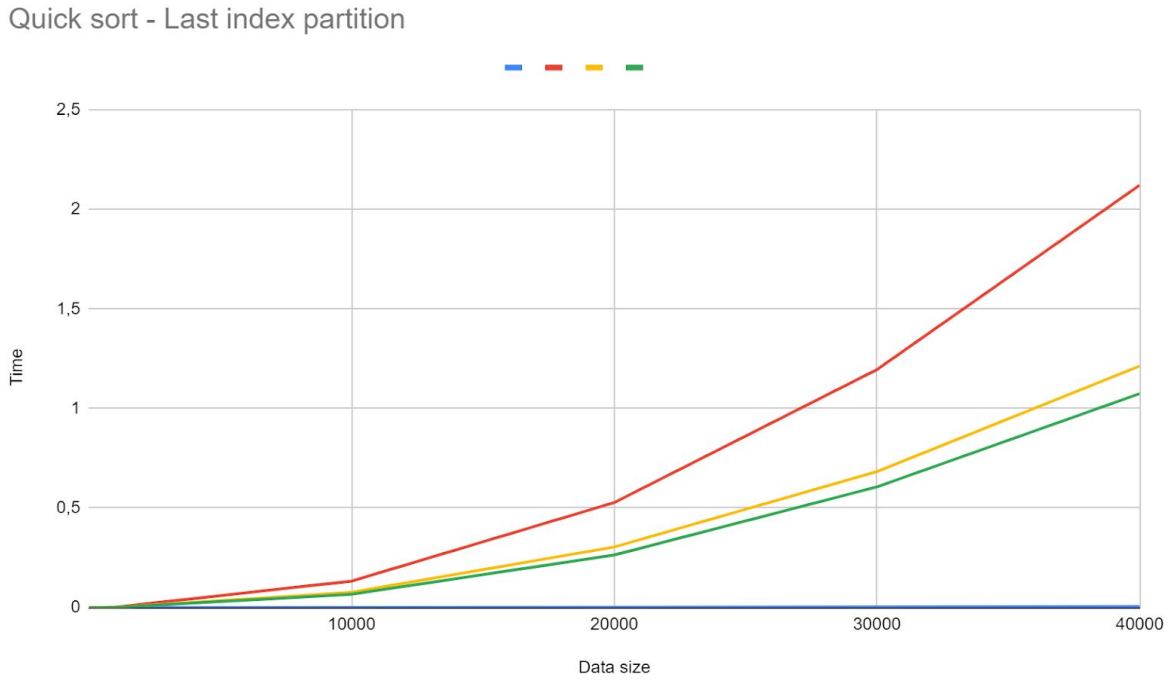
Wykres 4. Algorytm quick sort w wariancie z losowym elementem rozdzielającym - wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu. [kolor niebieski - rozkład losowy (ang. random), kolor czerwony - rozkład rosnący, kolor żółty - rozkład malejący, kolor zielony - rozkład V- kształtny]

W przypadku implementacji z ostatnim elementem rozdzielającym wydajność znacznie się zmniejsza. Na call stacku odkłada się zbyt wiele elementów co utrudnia zbadanie wydajności dla większej ilości elementów niż 40.000 (Błąd naruszenia pamięci) oraz znacznie to spowalnia działanie algorytmu. Dla losowego rozkładu danych ta implementacja algorytmu nie różni się wydajnością od implementacji z losowym elementem rozdzielającym i jest to najlepszy przypadek dla tej implementacji. Pesymistyczny przypadek działania tego algorytmu widoczny jest w rozkładzie danych posortowanych oraz w rozkładzie v-kształtnym.Ta implementacja algorytmu działa najgorzej dla rozkładu rosnącego, ponieważ na call stacku odkłada się taka liczba odwołań do funkcji ile jest elementów w sortowanej tablicy.

Tabela 5. Czas działania algorytmu quick sort w wariancie z ostatnim elementem rozdzielającym (w sekundach) dla różnego typu oraz różnej wielkości danych wejściowych.

	Random	Increasing	Decreasing	V-Shape
10	0	0	0	0
100	0	0	0	0
1000	0	0,002	0,001	0,001

10000	0,001	0,133	0,077	0,067
20000	0,002	0,527	0,304	0,264
30000	0,003	1,195	0,683	0,606
40000	0,005	2,121	1,213	1,074



Wykres 5. Algorytm quick sort w wariancie z ostatnim elementem rozdzielającym - wykres zależności rozmiaru danych/rozkładu danych od szybkości [sekundy] działania algorytmu. [kolor niebieski - rozkład losowy (ang. random), kolor czerwony - rozkład rosnący, kolor żółty - rozkład malejący, kolor zielony - rozkład V- kształtny]

Podsumowanie

Podsumowując, każdy z zaprezentowanych wyżej algorytmów różni się efektywnością wykonywania operacji sortowania dla różnego rozkładu i wielkości danych wejściowych.

O ile różnicę w szybkości sortowania poszczególnych algorytmów nie są aż tak istotne dla użytkownika w przypadku małej ilości danych podanych na wejściu, stają się one zauważalne w przypadku dużego zbioru elementów. Dla przykładu sortowanie danych złożonych z miliona elementów algorytm sortowania przez wybieranie, będącym najwolniej działającym spośród nich, posortował je średnio w czasie 306,6 sekund (5,11 minut). Natomiast dane o tym samym rozmiarze algorytm działający najszybciej, sortowanie przez kopcowanie, uporządkował jedynie średnio w czasie 0,1005 sekundy, co daje oszczędność czasu ponad 5 minut. Tak duże różnice nie są natomiast zauważalne w przypadku małych zbiorów danych wejściowych.

Znajomość zachowania się algorytmów dla różnego rozkładu i rozmiaru danych, którymi dysponujemy, jest istotne i umożliwi dobranie odpowiedniego sposobu sortowania, aby tą operację przeprowadzić jak najbardziej efektywnie.