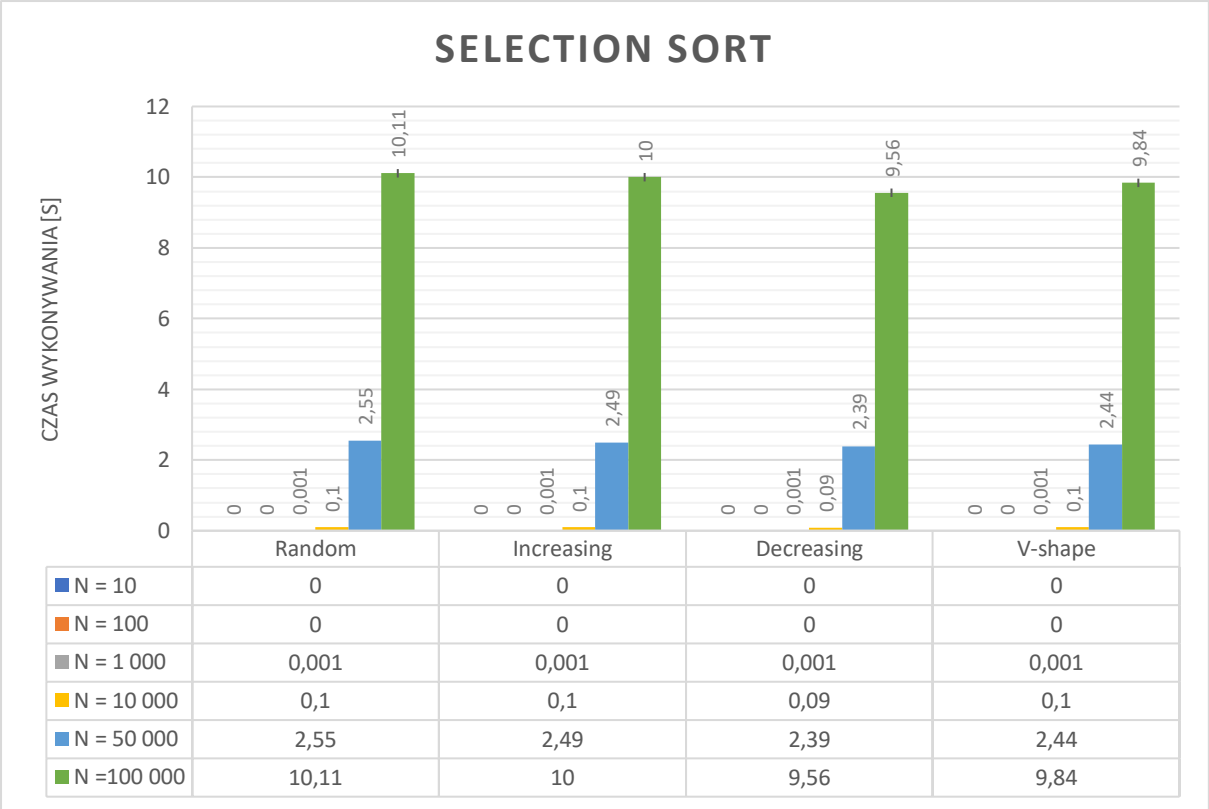
1. Czas działania poszczególnych algorytmów:



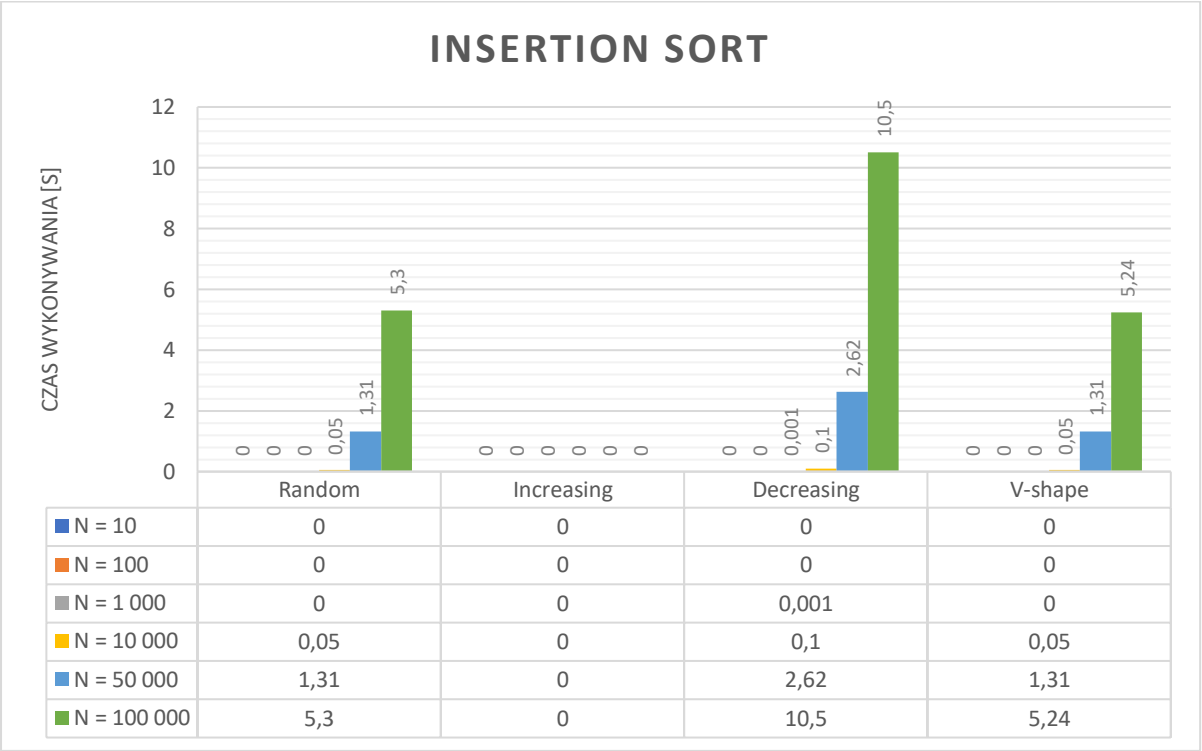
	Random	Increasing	Decreasing	V-shape
N = 10	0	0	0	0
N = 100	0	0	0	0
N = 1 000	0,001	0,001	0,001	0,001
N = 10 000	0,1	0,1	0,09	0,1
N = 50 000	2,55	2,49	2,39	2,44
N =100 000	10,11	10	9,56	9,84

Algorytm **selection sort** zachował się bardzo podobnie przy każdym z badanych przypadków. Niezależnie od wstępnego ułożenia danych w tabeli, można zauważyć wykładniczy wzrost ilości czasu potrzebnego do wykonania algorytmu względem wzrostu ilości danych. Przykład: porównując przypadki **n = 10^3** oraz **n = 10^4** widać, że stosunek ilości danych jest równy **10**, jednak stosunek ilości wymaganego czasu do wykonania algorytmu wyniósł 100. Ta sama zależność występuje między n=10^4 oraz 10^5. Idąc tym tropem, posortowanie **miliona** danych zajmie ok. **16,5 minuty**.

Wniosek:

Niezależnie od wstępnego ułożenia danych, zachowanie algorytmu jest takie same.

Złożoność obliczeniowa w najgorszym przypadku: $\Theta(n^2)$



	Random	Increasing	Decreasing	V-shape
N = 10	0	0	0	0
N = 100	0	0	0	0
N = 1 000	0	0	0,001	0
N = 10 000	0,05	0	0,1	0,05
N = 50 000	1,31	0	2,62	1,31
N = 100 000	5,3	0	10,5	5,24

Algorytm **insertion sort** zachowuje się różnie dla poszczególnych sposobów początkowego rozkładu danych:

a) Rozkład losowy oraz rozkład v-kształtny: wzrostu ilości czasu wymaganego do zakończenia algorytmu rośnie błyskawicznie wraz ze wzrostem ilości elementów.
b) Tablica ułożona: algorytm wykonuje go błyskawicznie, ponieważ nie musi przekładać żadnych elementów. Wykonywane jest n-1 porównań.
c) Tablica odwrotnie posortowana: zauważalny jest najwyższy wzrost ilości czasu potrzebnego do wykonania algorytmu w porównaniu ze wzrostem ilości danych do posortowania.

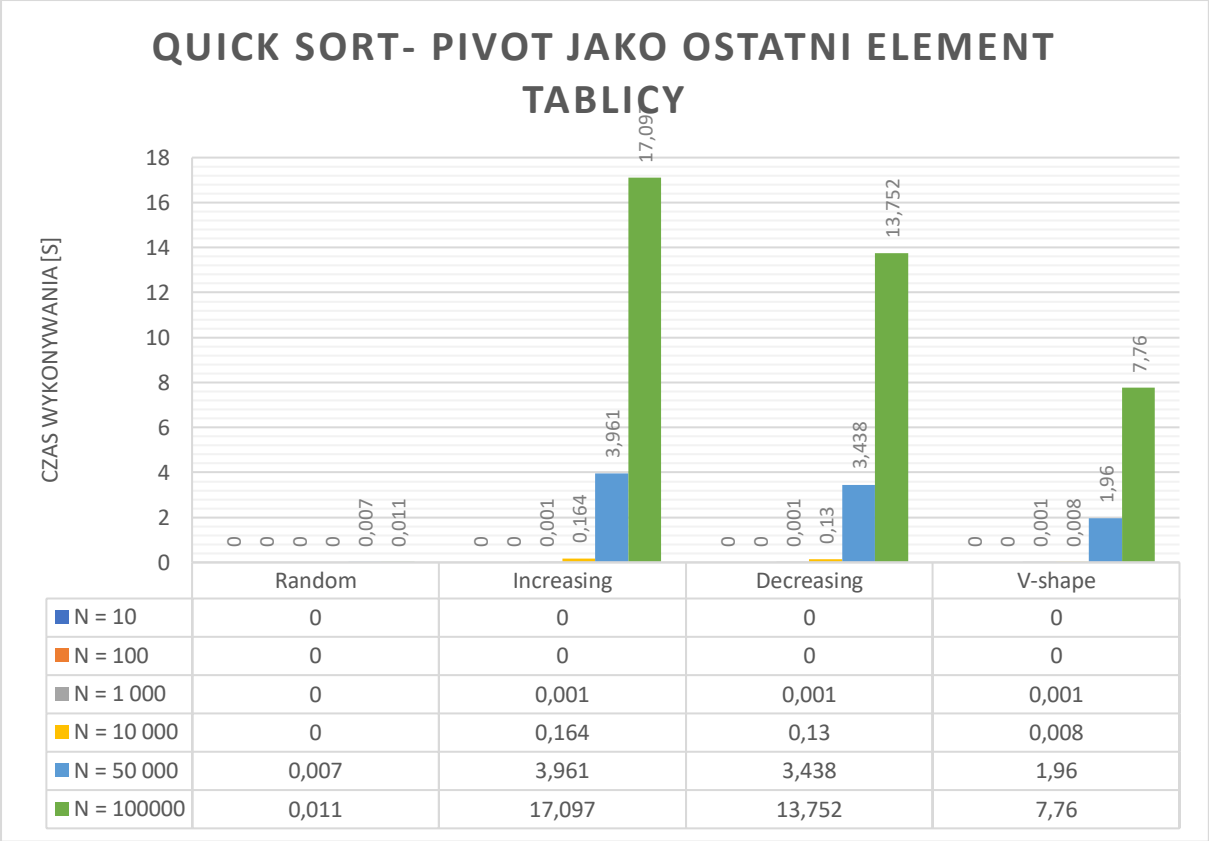
Wniosek:

Wydajność czasowa algorytmu zależy od początkowego rozkładu danych:

Najgorszy przypadek- tablica ułożona odwrotnie.
Najlepszy przypadek- tablica już posortowana.

Złożoność obliczeniowa w najgorszym przypadku: $\Theta(n^2)$

Złożoność obliczeniowa w najlepszym przypadku: $\Theta(n)$



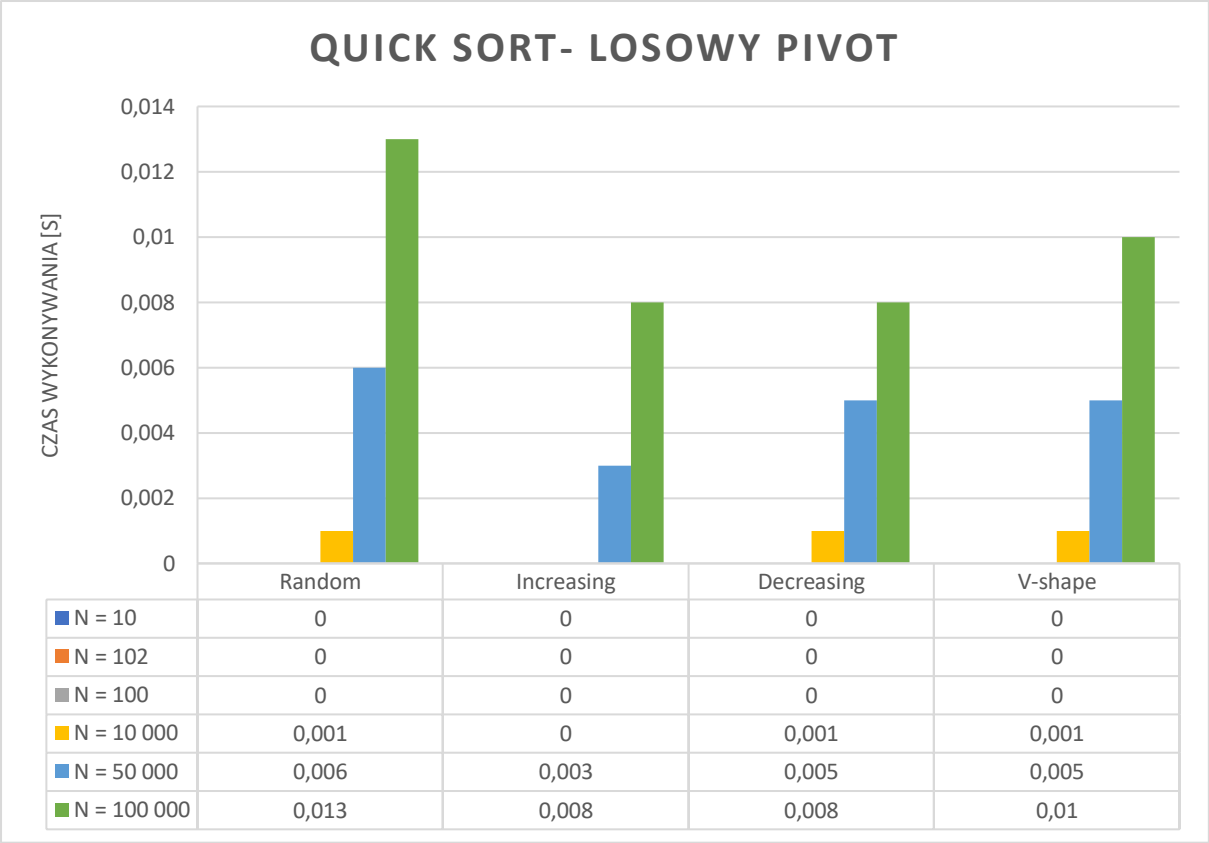
	Random	Increasing	Decreasing	V-shape
N = 10	0	0	0	0
N = 100	0	0	0	0
N = 1 000	0	0,001	0,001	0,001
N = 10 000	0	0,164	0,13	0,008
N = 50 000	0,007	3,961	3,438	1,96
N = 100000	0,011	17,097	13,752	7,76

Algorytm quick sort zachowuje się inaczej zależnie od ustawienia pivotu, więc ten algorytm przedstawiony jest w dwóch różnych wersjach.

W wersji, w której pivot jest ostatnim elementem tablicy poszczególne rozkłady danych mają duży wpływ na wydajność czasową algorytmu:

a) Rozkład losowy charakteryzuje się najkrótszym czasem działania, ponieważ dla 100 tys. elementów, czas działania wyniósł 11 ms, dla 50 tys. liczb, algorytm działał przez 7 ms, a 10 tys. elementów zostało posortowane poniżej 1 ms.
b) Posortowana tablica jest najgorszym przypadkiem. Złożoność czasowa algorytmu rośnie stukrotnie co każdy dziesięciokrotny wzrost liczby elementów. To samo dotyczy tablicy odwrotnie posortowanej oraz tablicy v-kształtnej, jednak w ich wypadku, wzrost jest nieco spowolniony.

Ogólna złożoność obliczeniowa: $\Theta(n \log n)$
Złożoność obliczeniowa w najgorszym przypadku $\Theta(n^2)$
Złożoność obliczeniowa w najlepszym przypadku: $\Theta(n \log n)$ lub $\Theta(n)$



	Random	Increasing	Decreasing	V-shape
N = 10	0	0	0	0
N = 102	0	0	0	0
N = 100	0	0	0	0
N = 10 000	0,001	0	0,001	0,001
N = 50 000	0,006	0,003	0,005	0,005
N = 100 000	0,013	0,008	0,008	0,01

Wersja, w której pivot jest losowany, w tym badaniu sortuje tablice o wiele szybciej. Algorytm ten „zachowuje” się podobnie względem poszczególnych rozkładów początkowych:

a) Wszystkie rosną wykładniczo, lecz od niższej liczby niż poprzedni algorytm.

Wniosek:

Ten eksperyment pokazał, że ten rodzaj quicksorta sortuje najszybciej ze wszystkich omawianych algorytmów.

Ogólna złożoność obliczeniowa: $\Theta(n \log n)$

Złożoność obliczeniowa w najgorszym przypadku $\Theta(n^2)$