# Sprawozdanie - Algorytmy Sortowania

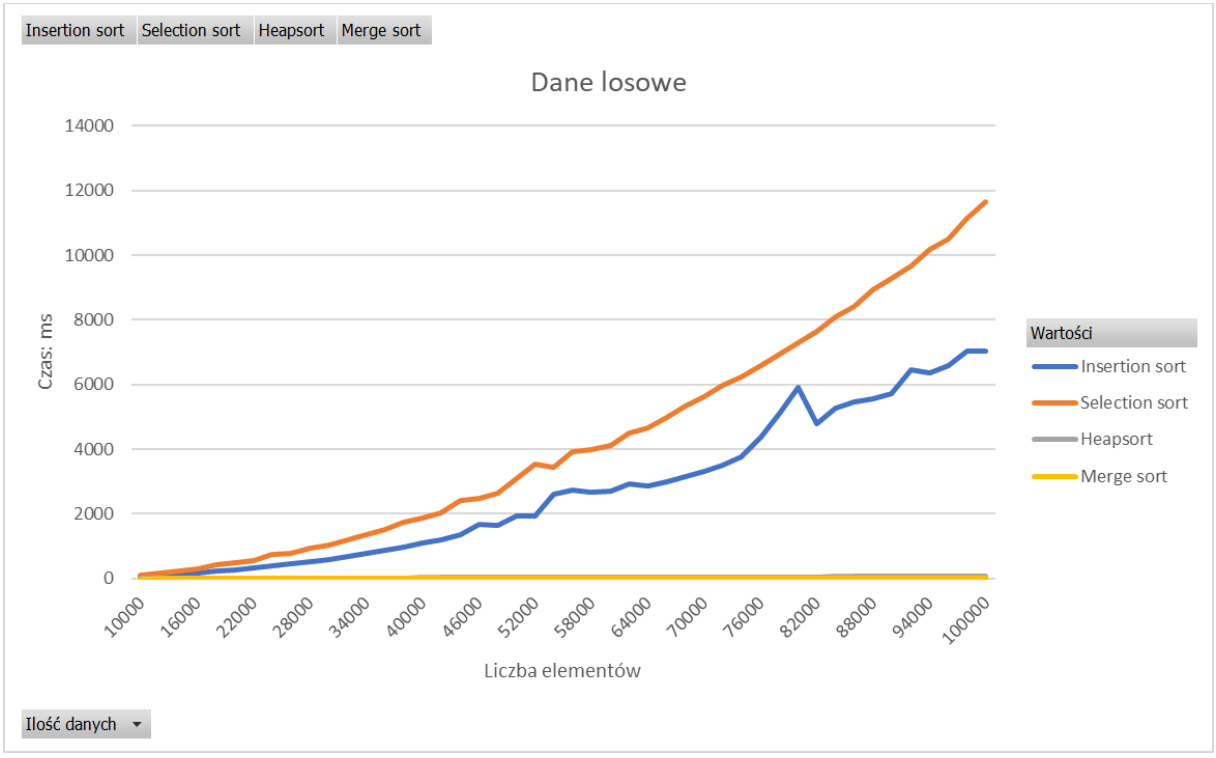
Szymon Szymankiewicz, Nikolas Szwargot

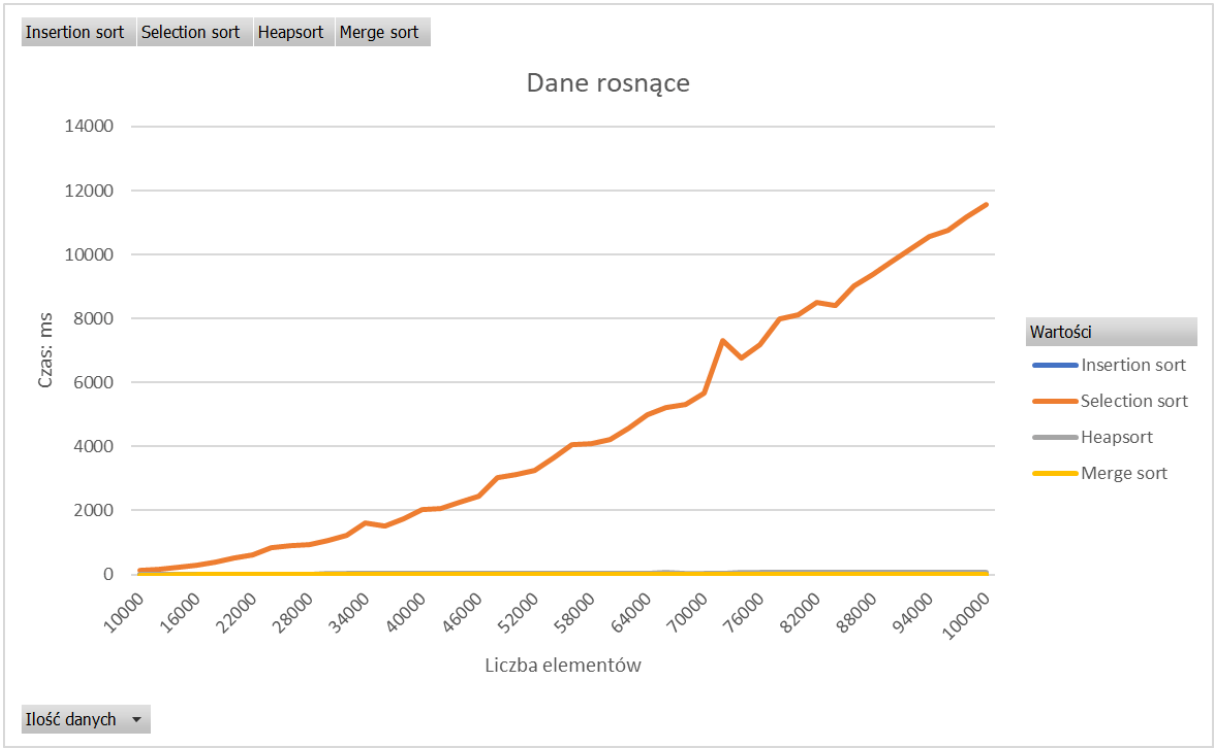
Lab 11, Informatyka, Semestr 2

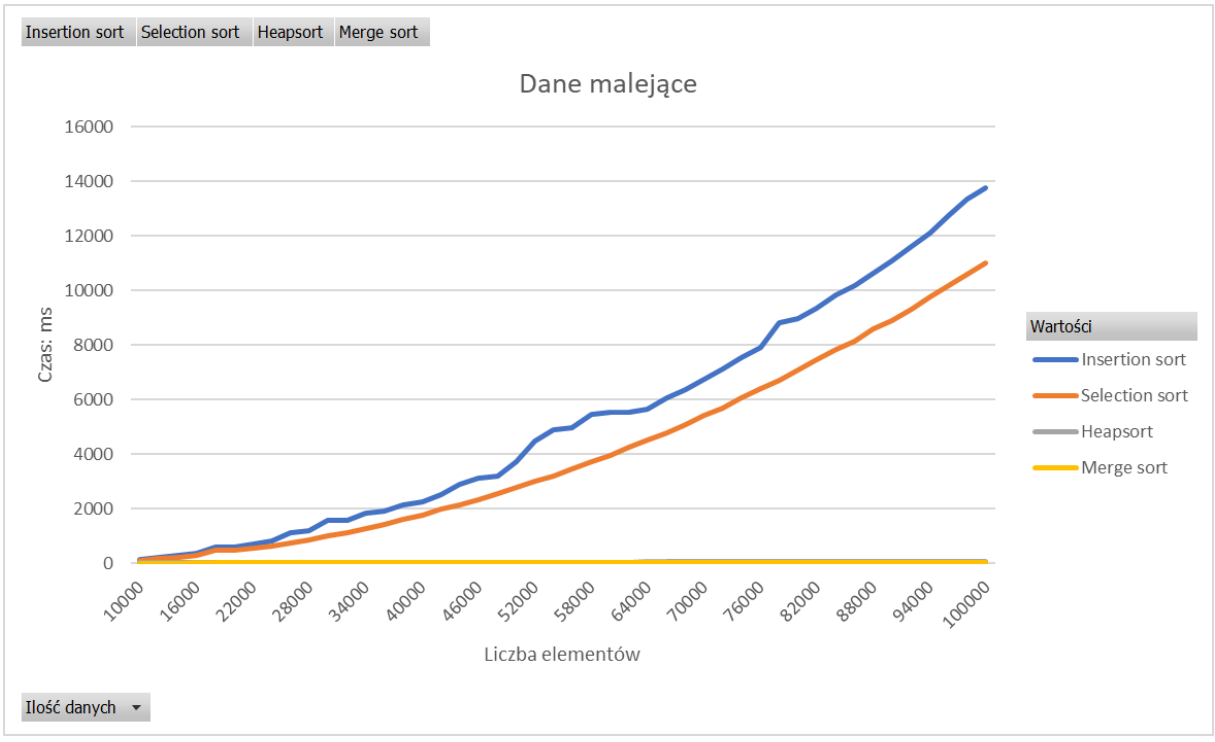
## Wstęp

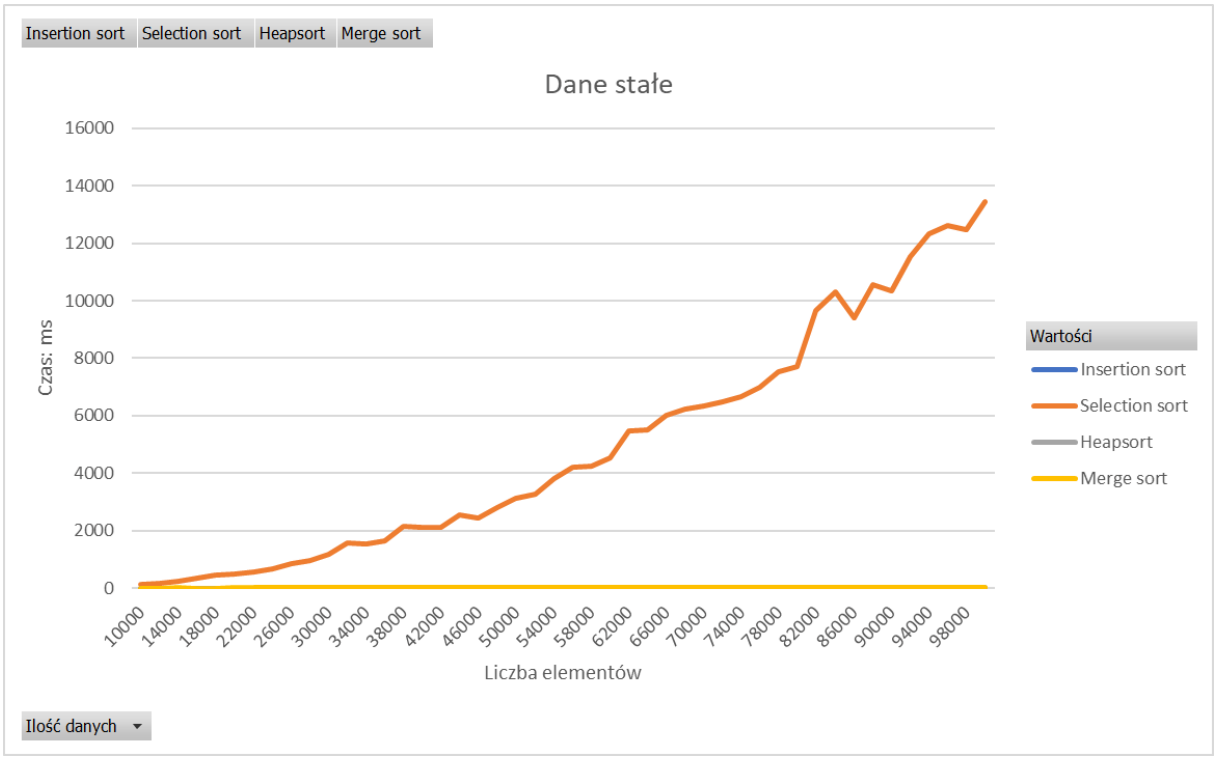
Celem naszego projektu jest analiza pięciu różnych algorytmów sortowania (Heapsort, Selection sort, Merge sort oraz Insertion sort) ze względu na tablicę liczb całkowitych generowanych w pięciu różnych postaciach (malejąca, rosnąca, losowa, v-kształtna oraz stała).i Jak również badane będzie działanie różnych kluczy dal algorytmu Quick Sort dla tablicy danych podanych w postaci A-kształtnej. Głównym celem było określenie szybkości oraz efektywności algorytmów jak i również porównanie tych aspektów z każdym z badanych algorytmów. Programy użyte do stworzenia wykresów zostały napisane w języku C++. W naszych wykresach rozpoczęliśmy od liczby 10 000 elementów do posortowania, a wykresy kończą się razem z progiem 100 000, krok co 2 000 (razem 45 punktów pomiarowych). Aby uzyskać pełen, czytelny widok wykresów czas wykonywania algorytmów został zmierzony w milisekundach.

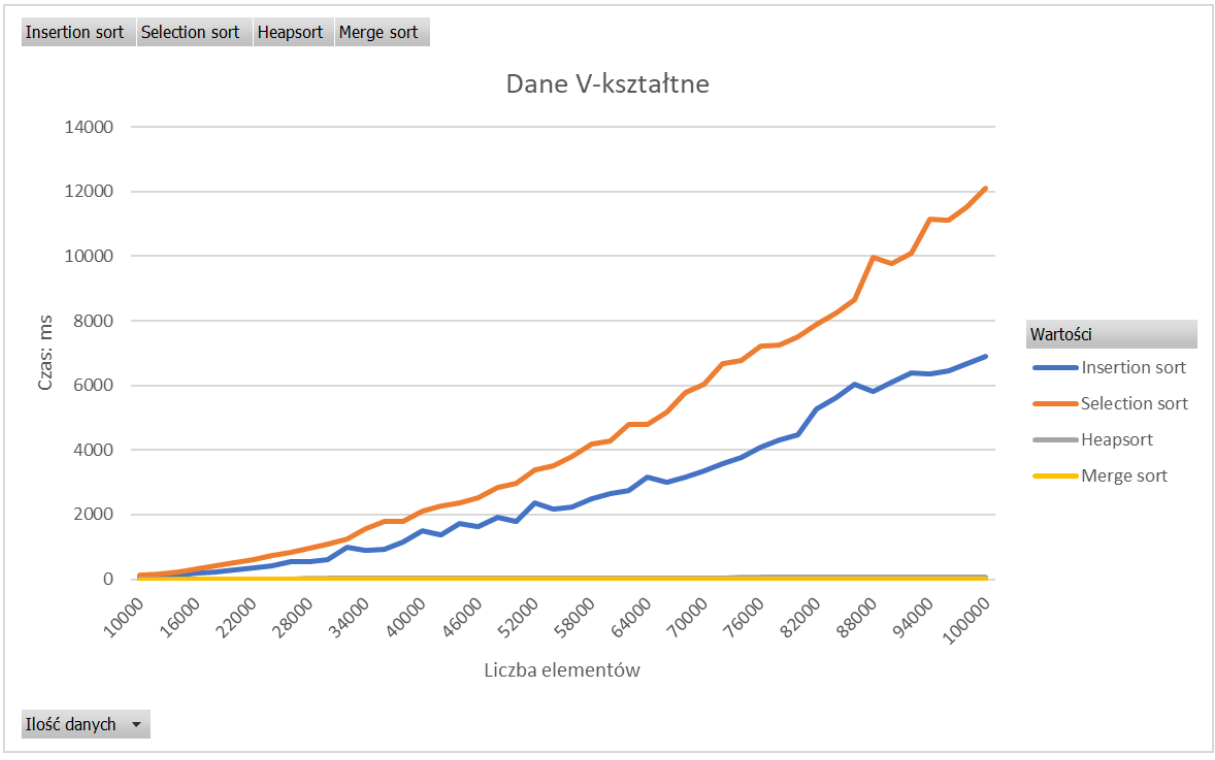
## Szybkość 4 sortowań dla tablicy liczb całkowitych generowanych w różnej postaci




Insertion sort
Selection sort
Heapsort
Merge sort
Dane rosnące
14000
12000
10000
8000
6000
4000
2000
0
Czas: ms
10000
16000
22000
28000
34000
40000
46000
52000
58000
64000
70000
76000
82000
88000
94000
100000
Liczba elementów
Wartości
Insertion sort
Selection sort
Heapsort
Merge sort
Ilość danych


Insertion sort
Selection sort
Heapsort
Merge sort
Dane malejące
16000
14000
12000
10000
8000
6000
4000
2000
0
Czas: ms
10000
16000
22000
28000
34000
40000
46000
52000
58000
64000
70000
76000
82000
88000
94000
100000
Liczba elementów
Wartości
Insertion sort
Selection sort
Heapsort
Merge sort
Ilość danych


Insertion sort
Selection sort
Heapsort
Merge sort
Dane stałe
16000
14000
12000
10000
8000
6000
4000
2000
0
Czas: ms
10000
14000
18000
22000
26000
30000
34000
38000
42000
46000
50000
54000
58000
62000
66000
70000
74000
78000
82000
86000
90000
94000
98000
Liczba elementów
Wartości
Insertion sort
Selection sort
Heapsort
Merge sort
Ilość danych


Insertion sort
Selection sort
Heapsort
Merge sort
Dane V-kształtne
14000
12000
10000
8000
6000
4000
2000
0
Czas: ms
10000
16000
22000
28000
34000
40000
46000
52000
58000
64000
70000
76000
82000
88000
94000
100000
Liczba elementów
Wartości
Insertion sort
Selection sort
Heapsort
Merge sort
Ilość danych

# Wnioski

**Generowanie liczb w postaci losowej** – na samym początku podczas analizowania zachowań każdego z sortowań należy wziąć pod uwagę to, że dane generowane losowo mogą pokazywać wyniki, które nie zgadzają się ze sobą patrząc na kolejne próby (np. po ciągłym rośnięciu czas wykonywania może zmaleć). Mimo tego na pierwszy rzut oka można zauważyć, że w przypadku danych losowych Insertion Sort oraz Selection Sort radzą sobie dużo gorzej niż Heap Sort i Merge Sort, których czas jest bardzo mały w prównaniu do dwóch pierwszych algorytmów. Powodem tego może być złożoność obliczeniowa w przypadku danych losowych (przypadek typowy). Pokazuje to również różnicę w działaniu złożoności obliczeniowej $O(n^2)$ (Insertion Sort oraz Selection Sort) od złożoności $O(n\log(n))$(Heap Sort i Merge Sort).

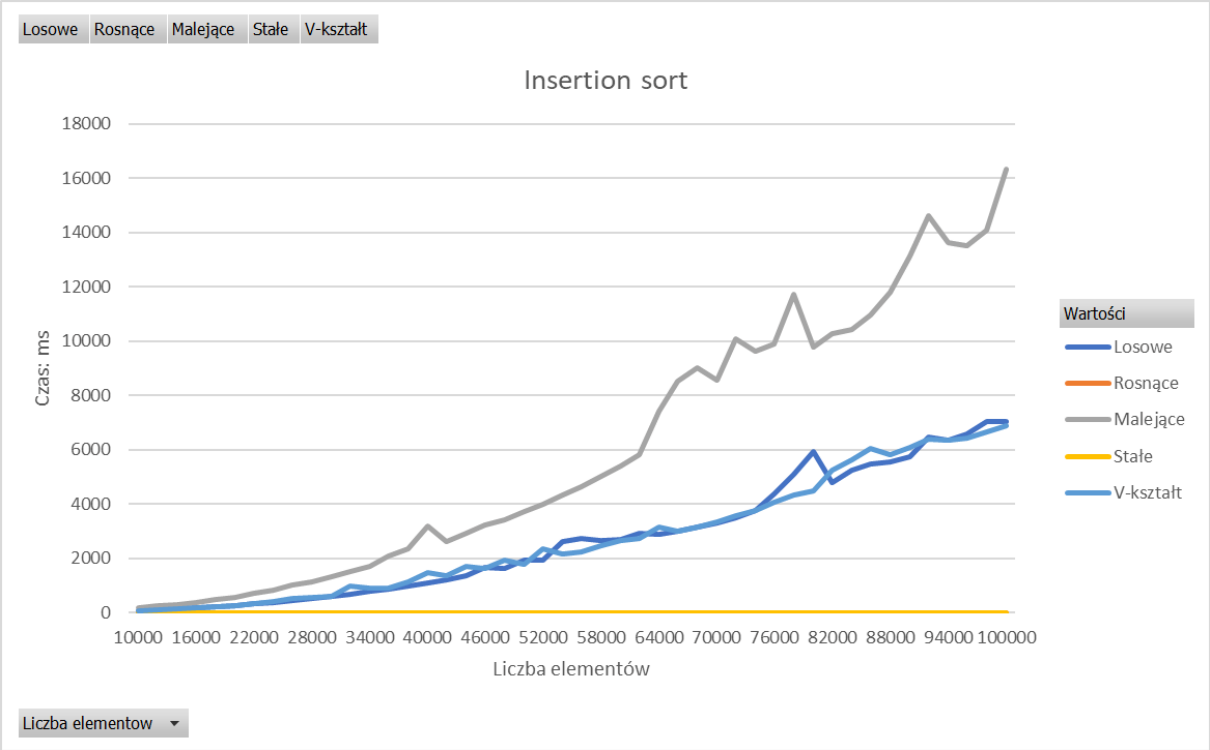
**Generowanie liczb w postaci rosnącej** – przypadek rosnący dokładnie ukazuje nam jaką przewagę w szybkości działania algorytmu posiada Insertion Sort nad algorytmem Selection Sort. Gdy podczas generowania liczb w postaci losowej obie złożoności wynoszą $O(n^2)$, tak w tym przypadku Insertion Sort zmienia swoją złożoność do najlepszego przypadku równego $O(n)$, co znacznie przyspiesza czas wykonania.

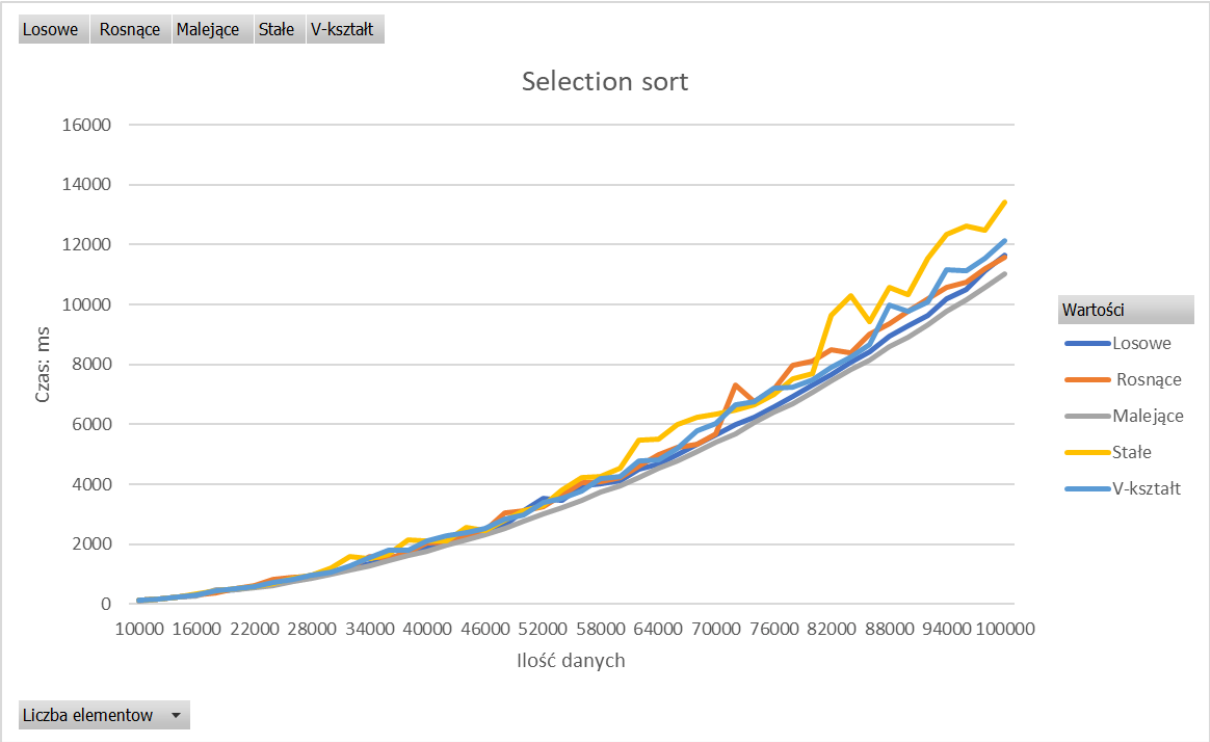
**Generowanie liczb w postaci malejącej** – wykres przedstawiający tablice z danymi malejącymi może przypominać nam wykres z danymi podanymi w postaci losowej – wszystkie złożoności wynoszą dokładnie tyle samo, lecz występująca różnica w szybkości sortowania przez wstawianie i sortowania przez wybieranie może być znacząca w stwierdzeniu, który z nich radzi sobie lepiej. Wykres ten można uznać za bardziej zaufany zważając na aspekt bardzo dużej losowości w nadmienionym wykresie z danymi losowymi przez co porównanie prędkości wykonania algorytmów staje się mniej rzetelne i wiarygodne.

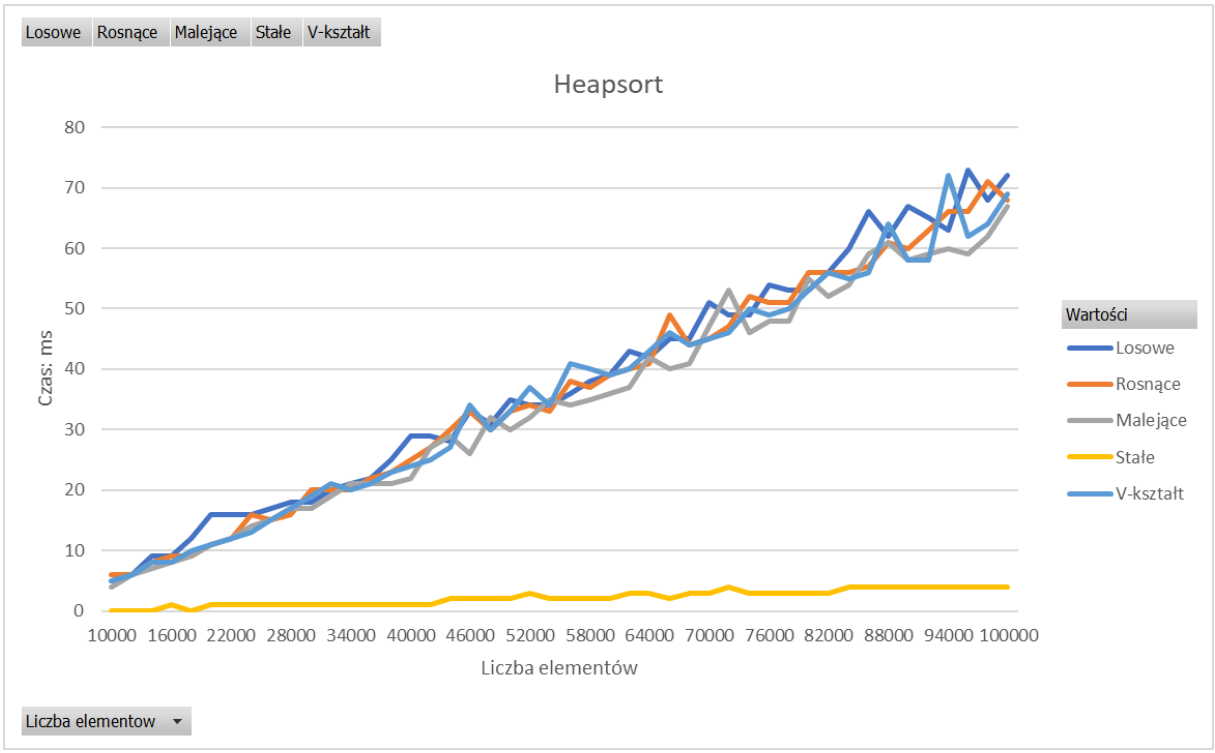
**Generowanie liczb w postaci stałej** – podczas przechodzenia algorytmów przez tablice liczb w postaci stałej, złożoność obliczeniowa sortowań: Heap sort oraz Insertion sort wyrównuje się do wartości $O(n)$. Z powodu, że algorytm sortowania Selection Sort jest stały w swojej złożoności, która w porównaniu do reszty algorytmów jest dość duża $O(n^2)$, dlatego we wszystkich pięciu wykresach można zauważyć szybki wzrost w czasie wykonywania tego algorytmu.

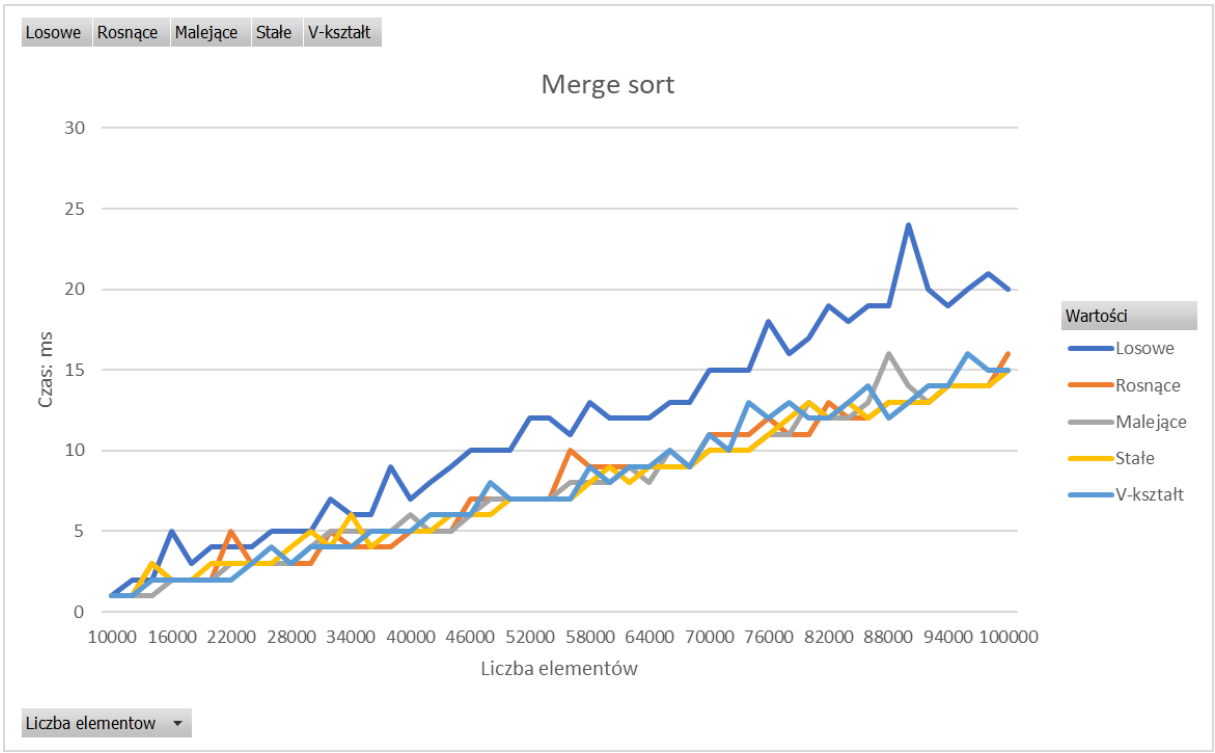
**Generowanie liczb w postaci V-kształtnej** – przy danych V-kształtnych (jak i również w wykresach opisanych powyżej) nadal widać przewagę w szybkości obliczeniowej sortowań: Heap Sort oraz Merge Sort. Tutaj również można zauważyć nasz wniosek o szybkości obliczeniowej dla Insertion Sort przy danych rosnących, ponieważ przy tablicy w postaci V-kształtnej dane najpierw maleją, a następnie rosną (w tym miejscu podczas takich tablic Insertion Sort zyskuje większą przewagę nad Selection Sort).

# Efektywność 4 sortowań dla tablicy liczb całkowitych generowanych w różnej postaci




Losowe
Rosnące
Malejące
Stałe
V-kształt
Heapsort
80
70
60
50
40
30
20
10
0
Czas: ms
10000 16000 22000 28000 34000 40000 46000 52000 58000 64000 70000 76000 82000 88000 94000 100000
Liczba elementów
Wartości
Losowe
Rosnące
Malejące
Stałe
V-kształt
Liczba elementow


Losowe
Rosnące
Malejące
Stałe
V-kształt
Merge sort
30
25
20
15
10
5
0
Czas: ms
10000 16000 22000 28000 34000 40000 46000 52000 58000 64000 70000 76000 82000 88000 94000 100000
Liczba elementów
Wartości
Losowe
Rosnące
Malejące
Stałe
V-kształt
Liczba elementow

# Wnioski

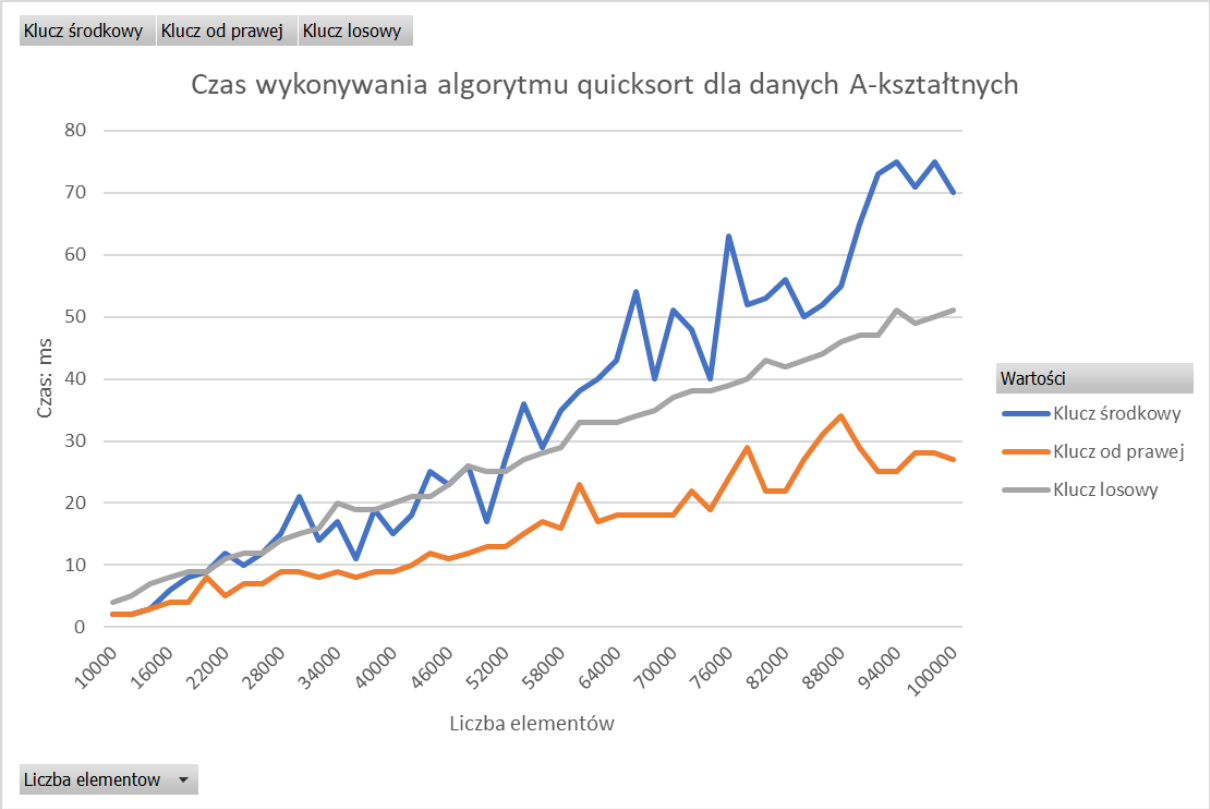
**Insertion Sort** – czyli sortowanie przez wstawianie. Jest to najwolniejszy z naszych 4 badanych algorytmów. Sortowanie odbywa się w miejscu. Algorytm Insertion Sort bazuje na podziale podanej tablicy na 2 części (uporządkowana oraz nieuporządkowana). W przypadku tego sortowania złożoność obliczeniowa nie jest jednakowa. Na powyższym wykresie zauważyć można, że Insertion sort działa zdecydowanie lepiej w przypadku optymistycznym (czyli dla danych rosnących oraz stałych). Dzieje się tak, ponieważ w przypadku optymistycznym złożoność obliczeniowa wynosi $O(n)$, lecz w przypadku typowym jak i pesymistycznym złożoność obliczeniowa zwiększa się do $O(n^2)$. Można dojść do wniosku, że algorytm sortowania przez wstawianie nadaje się do wstępnie uporządkowanych tablic. W przypadku złożoności pamięciowej algorytm nie wykorzystuje żadnej dodatkowej struktury danych, więc złożoność pamięciowa wynosi $O(1)$. Jest to algorytm stabilny.

**Selection Sort** – czyli sortowanie przez wybieranie. Jest to jedna z prostszych metod sortowania, jednak jak pokazuje czas wykonywania każdego z algorytmów Selection sort nie jest najbardziej efektywną i najszybszą z metod sortowania. Sortowanie odbywa się w miejscu. Złożoność obliczeniowa dla takiego algorytmu jest stała i wynosi $O(n^2)$. Złożoność pamięciowa również w przypadku tego sortowania wynosi $O(1)$ (nie jest potrzebna dodatkowa struktura danych), a algorytm działa niestabilnie.

**Heapsort** – czyli sortowanie przez kopcowanie. Jest to jeden z szybkich algorytmów, który wykorzystuje kopiec binarny (drzewo binarne o określonym porządku). Algorytm sortuje się w miejscu. Złożoność obliczeniowa takiego algorytmu wynosi $O(n\log(n))$, tylko dla przypadku typowego oraz pesymistycznego. Z wykresu można zauważyć, że dla wartości stałych, czas na przejście algorytmu jest znacznie mniejszy. Dzieje się tak dlatego, ponieważ algorytm potrzebuje złożoności obliczeniowej równej tylko $O(n)$, ponieważ w drzewie nie dokonują się żadne zmiany. Heap Sort jest to algorytm niestabilny.

**Merge Sort** – czyli sortowanie przez scalanie. Sortowanie nie odbywa się w miejscu. Jest to rekurencyjny algorytm sortowania o złożoności obliczeniowej równej $O(n\log(n))$. Można zauważyć, że Merge Sort utrzymuje swoją złożoność dla wszystkich rodzajów generowania danych. Jest to jeden z szybszych badanych algorytmów. Dla Merge Sort'a złożoność pamięciowa wynosi $O(n)$. Algorytm działa stabilnie.

# Efektywność algorytmu Quicksort w wersji rekurencyjnej z różnymi sposobami wyboru klucza




## Wnioski

**Quick Sort** – czyli sortowanie szybkie, które korzysta z techniki dziel i zwyciężaj. Średnia złożoność obliczeniowa wynosi $O(n\log(n))$, natomiast w pesymistycznej wersji zwiększa się do $O(n^2)$. Średnia złożoność pamięciowa poprzez używanie przez algorytm stosu wywołań rekurencyjnych wynosi $O(\log(n))$.

**Klucz skrajnie prawy** – z porównania trzech kluczy do sortowania można wywnioskować, że jest to klucz, który razem z samym algorytmem działa najszybciej w porównaniu do reszty badanych kluczy. Sortowanie odbywa się w miejscu.

**Klucz środkowy co do położenia** – jest to klucz, który jak można zauważyć po licznych nagłych skokach jest najmniej stabilny oraz również najmniej efektywny. Sortowanie również odbywa się w miejscu.

**Klucz losowo wybrany** – jest to klucz, który ukazuje średnią efektywność algorytmu, a sortowanie tego algorytmu odbywa się w miejscu. Złożoność obliczeniowa również w tym przypadku wynosi $O(n\log(n))$.

# Podsumowanie

Po wykonaniu wykresów i przedstawieniu wniosków można zauważyć, że najczęstsza złożoność obliczeniowa sortowań to: $O(n^2)$, $O(\log(n))$, $O(n)$. Taka kolejność wartości złożności obliczeniowych przedstawia również kolejność tych złożoności od tych, które są najmniej efektywne do najszybszych i najbardziej efektywnych. Wyniki pomiarów wskazują również, że najlepszym z badanych algorytmów jest Merge Sort, między innymi przez swoją złożoność, jak i fakt dobrego radzenia sobie z różnorodnym sposobem podania liczb całkowitych w tablicy do posortowania.