Algorytmy Sortujące

SPRAWOZDANIE I

PAWEŁ KOLEC 155873

Wprowadzenie:

W niniejszym sprawozdaniu przeprowadzono badania efektywności siedmiu popularnych algorytmów sortowania w języku Python 3 na pięciu typach danych generowanych losowo. Testy zostały przeprowadzone dla dziesięciu różnych rozmiarów danych, aby dokładnie zbadać, który z algorytmów działa najszybciej dla danych wejściowych różnych typów, a także jak zmienia się czas działania w zależności od rozmiaru danych wejściowych.

W ramach badania przetestowano siedem algorytmów sortowania: Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort, Shell Sort oraz Bubble Sort. Każdy z tych algorytmów został przetestowany dla pięciu różnych typów danych generowanych losowo: pełni losowego, rosnącego, malejącego, A-kształtnego oraz V-kształtnego. Dziesięć różnych rozmiarów danych, na które przeprowadzono testy, to 100, 500, 1000, 2500, 5000, 10000, 20000, 30000, 40000 oraz 50000.

Celem badania było wybranie najlepszego algorytmu sortowania dla każdego typu danych oraz zbadanie, jak złożoność czasowa każdego z algorytmów wpływa na czas działania.

Wyniki analizy czasowej każdego algorytmu sortowania przedstawiono w formie wykresów, które pozwalają na łatwe porównanie wyników. Analiza złożoności czasowej każdego algorytmu pozwala na określenie, jak zmienia się czas działania w zależności od rozmiaru danych wejściowych. Badania zostały przeprowadzone przy użyciu środowiska PyCharm (x64) na systemie Microsoft Windows 11 Home z procesorem Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz.

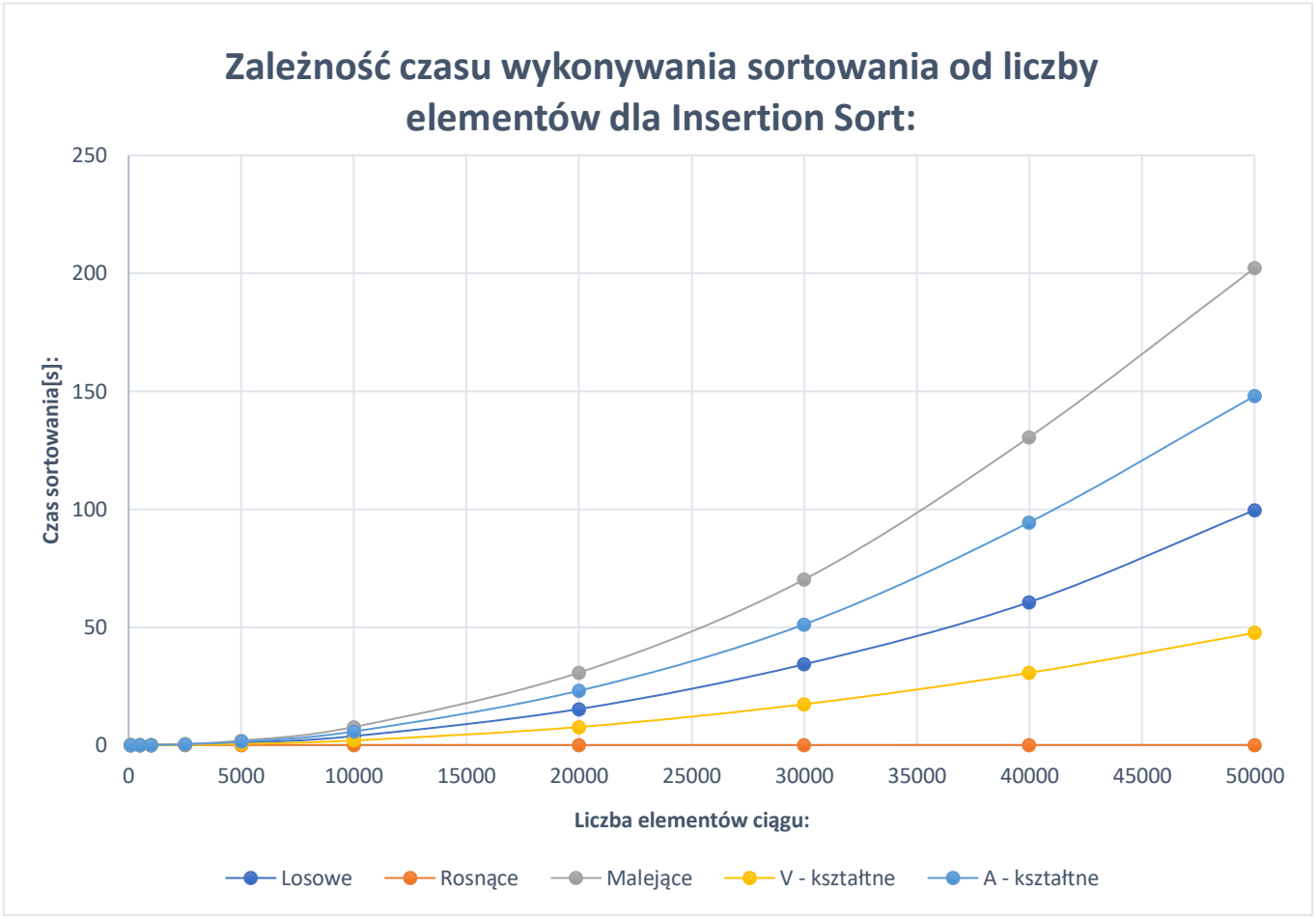
Zgodnie z celem badania, wyniki przedstawione w sprawozdaniu pozwalają na określenie, który z algorytmów sortowania działa najlepiej dla różnych typów danych wejściowych oraz jak złożoność czasowa wpływa na czas działania każdego z algorytmów.

Insertion Sort

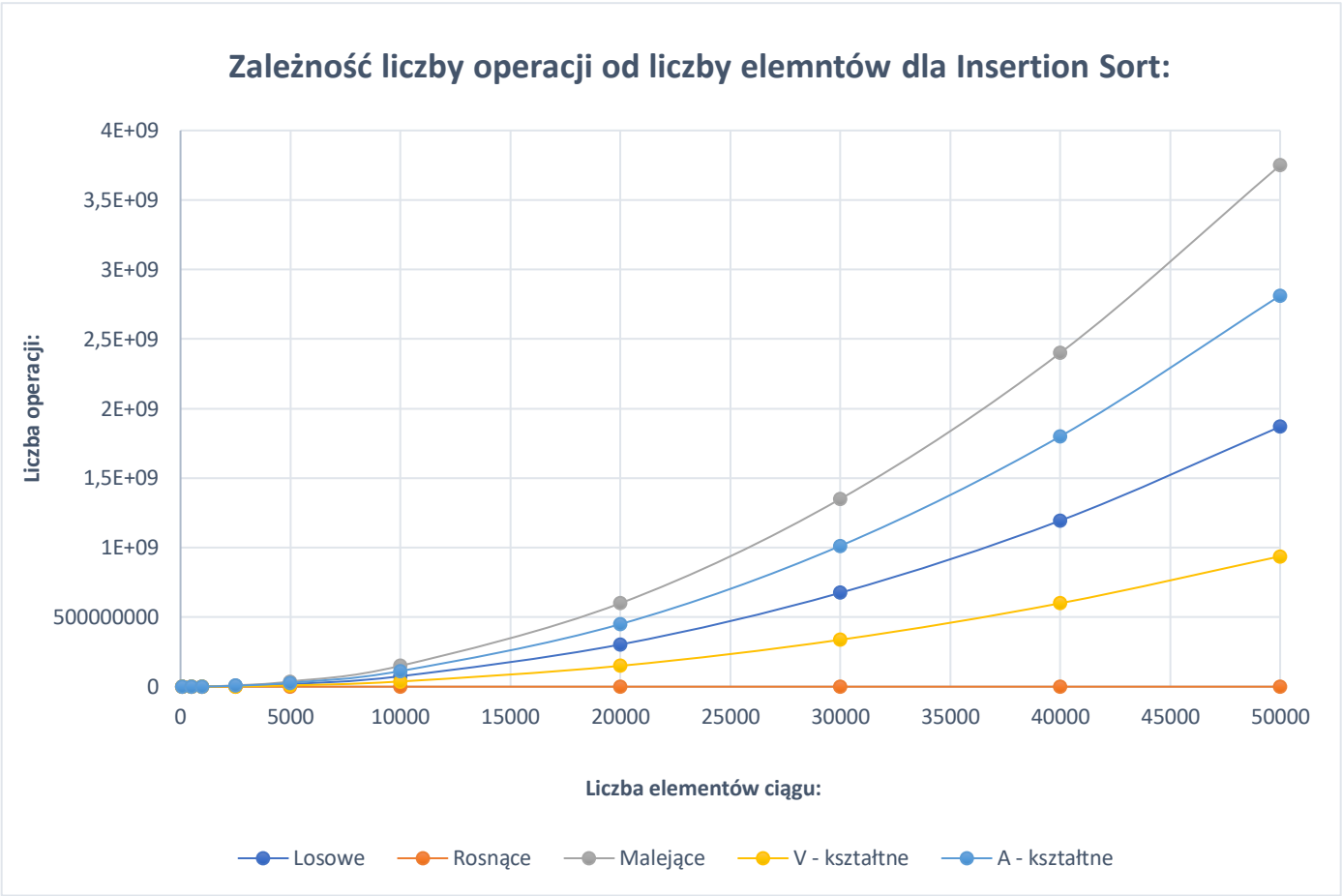
Opis:

Insertion Sort to algorytm sortowania, który porównuje każdy element z pozostałymi elementami i wstawia element w odpowiednie miejsce w posortowanym ciągu. Algorytm ten ma optymistyczną złożoność czasową O(n), co oznacza, że jest wydajny dla małych zbiorów danych.

Zależność czasu wykonywania sortowania od liczby elementów dla Insertion Sort:



Zależność liczby operacji od liczby elementów dla Insertion Sort:



Złożoność:

Algorytm sortowania Insertion Sort jest prosty i wydajny dla małych zbiorów danych, ponieważ ma optymistyczną złożoność O(n). Jednak jego pesymistyczna złożoność to O(n^2), co oznacza, że dla dużych zbiorów danych może być wolny. Algorytm porównuje każdy element z pozostałymi elementami, a następnie wstawia element w odpowiednie miejsce w posortowanym ciągu.

Wnioski:

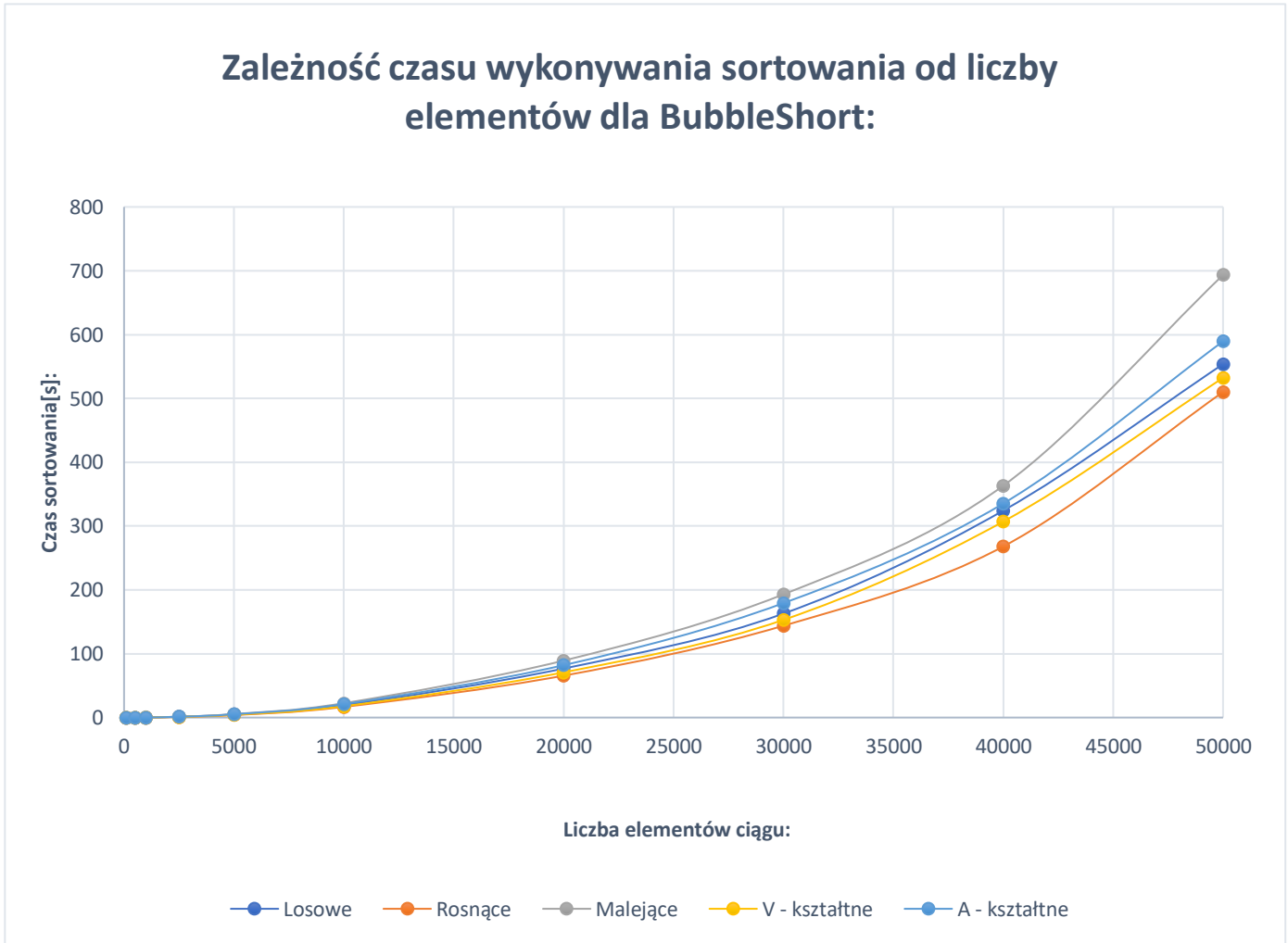
Algorytm sortowania przez wstawianie Insertion Sort jest stosunkowo prosty w implementacji, ale niestety działa bardzo wolno dla większych zbiorów danych.

Bubble Sort:

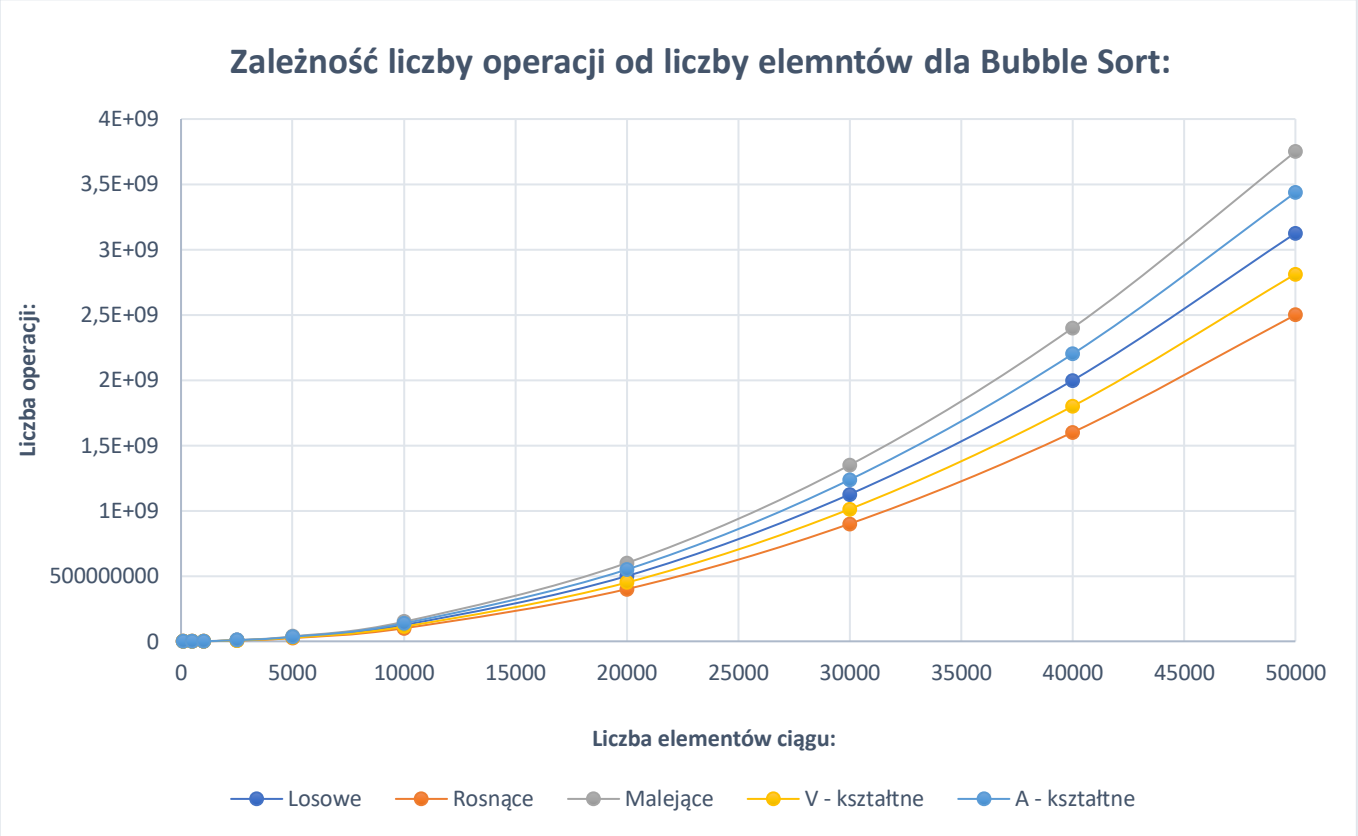
Opis:

Bubble sort to algorytm sortowania porządkujący elementy w tablicy poprzez porównywanie kolejnych par wartości i zamienianie ich miejscami, jeśli są one w złej kolejności. Algorytm kontynuuje tę operację aż do momentu, gdy cała tablica zostanie posortowana. Bubble sort jest jednym z najprostszych algorytmów sortowania, ale ma złożoność czasową O(n^2), co oznacza, że nie jest wydajny dla dużych zbiorów danych.

Zależność czasu wykonywania sortowania od liczby elementów dla Bubble Sort:



Zależność liczby operacji od liczby elementów dla Bubble Sort:



Złożoność:

Bubble sort jest prostym algorytmem sortowania, jednak ma złożoność czasową O(n^2) dla każdego przypadku. Bez względu na kolejność elementów w tablicy, algorytm wykonuje w przybliżeniu n^2/2 porównań i n^2/2 zamian, co prowadzi do takiej samej złożoności czasowej.

Wnioski:

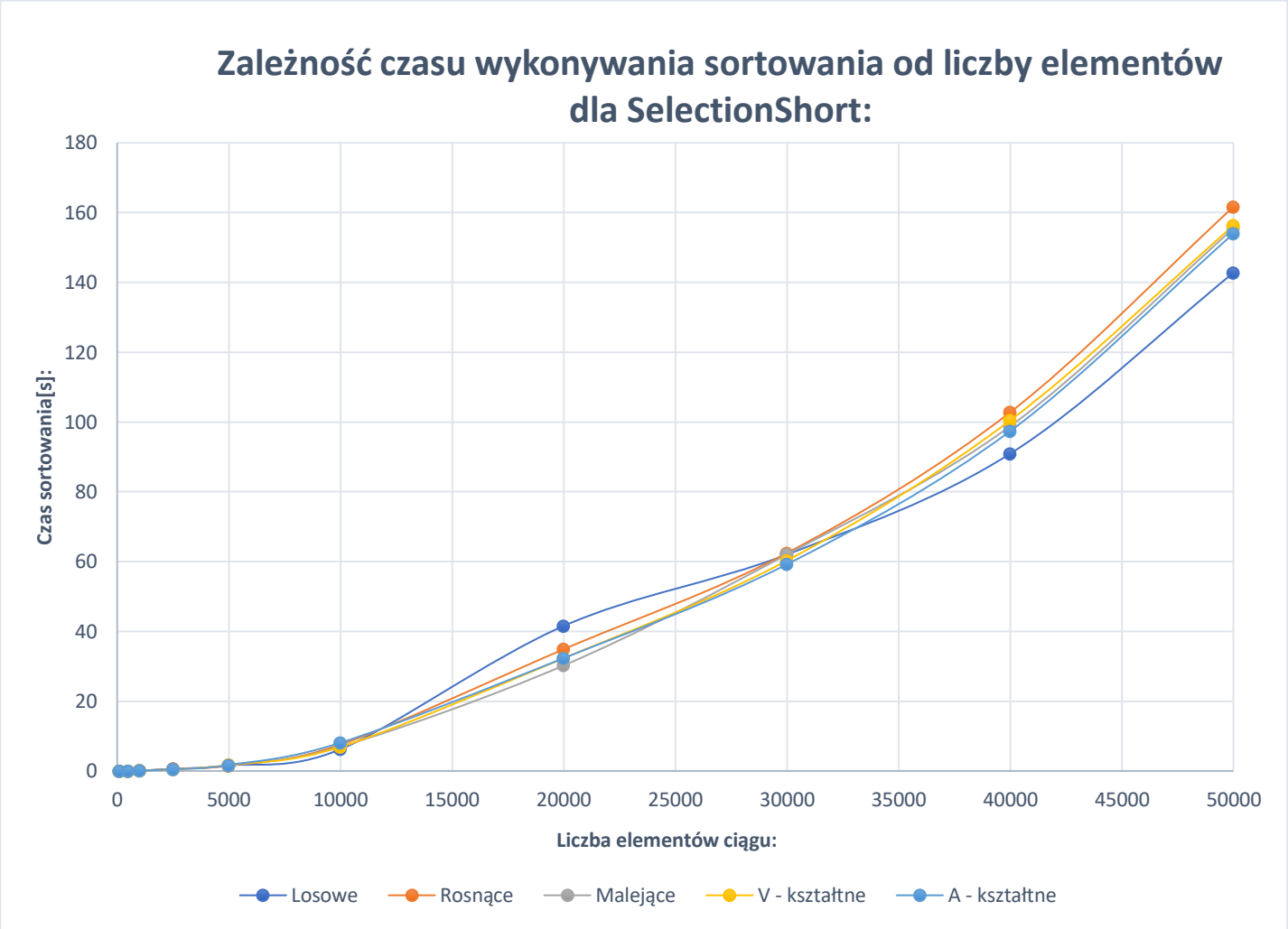
Algorytm sortowania Bubble Sort jest stosunkowo prosty w implementacji, ale niestety działa bardzo wolno niezależnie od typu danych wejściowych.

Selection Sort:

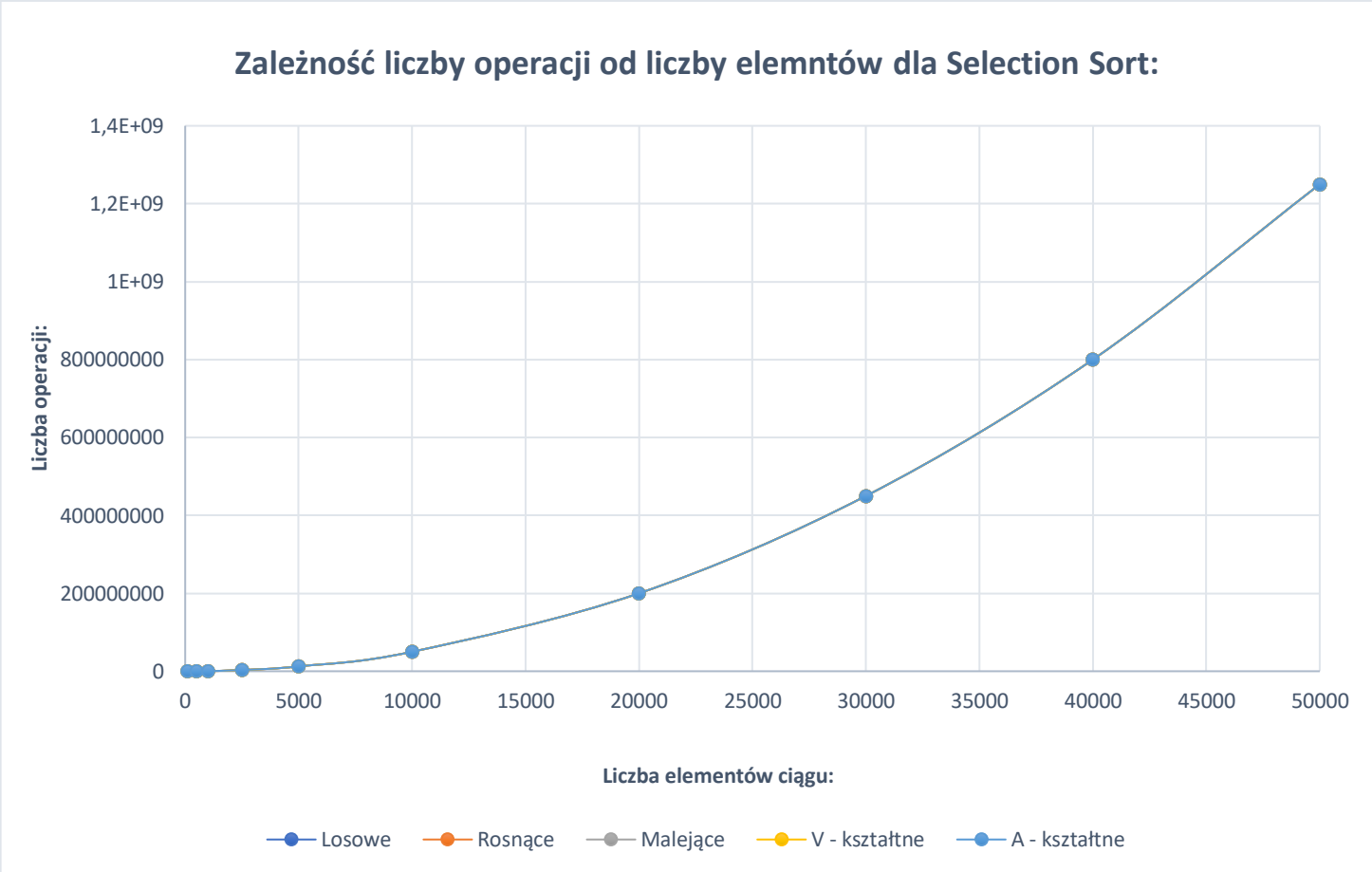
Opis:

Selection Sort to kolejny algorytm sortowania, który polega na znajdowaniu w tablicy najmniejszego elementu i zamianie go z pierwszym elementem, a następnie powtarzaniu tej operacji dla pozostałych elementów. Algorytm wykonuje n-1 iteracji, gdzie n to liczba elementów w tablicy, co daje złożoność czasową O(n^2). Selection Sort jest prosty w implementacji, ale mało wydajny dla dużych zbiorów danych, ponieważ wymaga dużo porównań i zamian, nawet gdy tablica jest już częściowo posortowana.

Zależność czasu wykonywania sortowania od liczby elementów dla Selection Sort:



Zależność liczby operacji od liczby elementów dla Selection Sort:



Złożoność:

Selection Sort ma złożoność czasową O(n^2), co oznacza, że dla n-elementowej tablicy algorytm wykonuje n^2 operacji. Złożoność przestrzenna Selection Sorta wynosi O(1), ponieważ algorytm sortuje elementy w miejscu, bez potrzeby tworzenia nowych struktur danych. Selection Sort nie jest wydajny dla dużych zbiorów danych, ponieważ wymaga dużo porównań i zamian elementów, nawet gdy tablica jest już częściowo posortowana.

Wnioski:

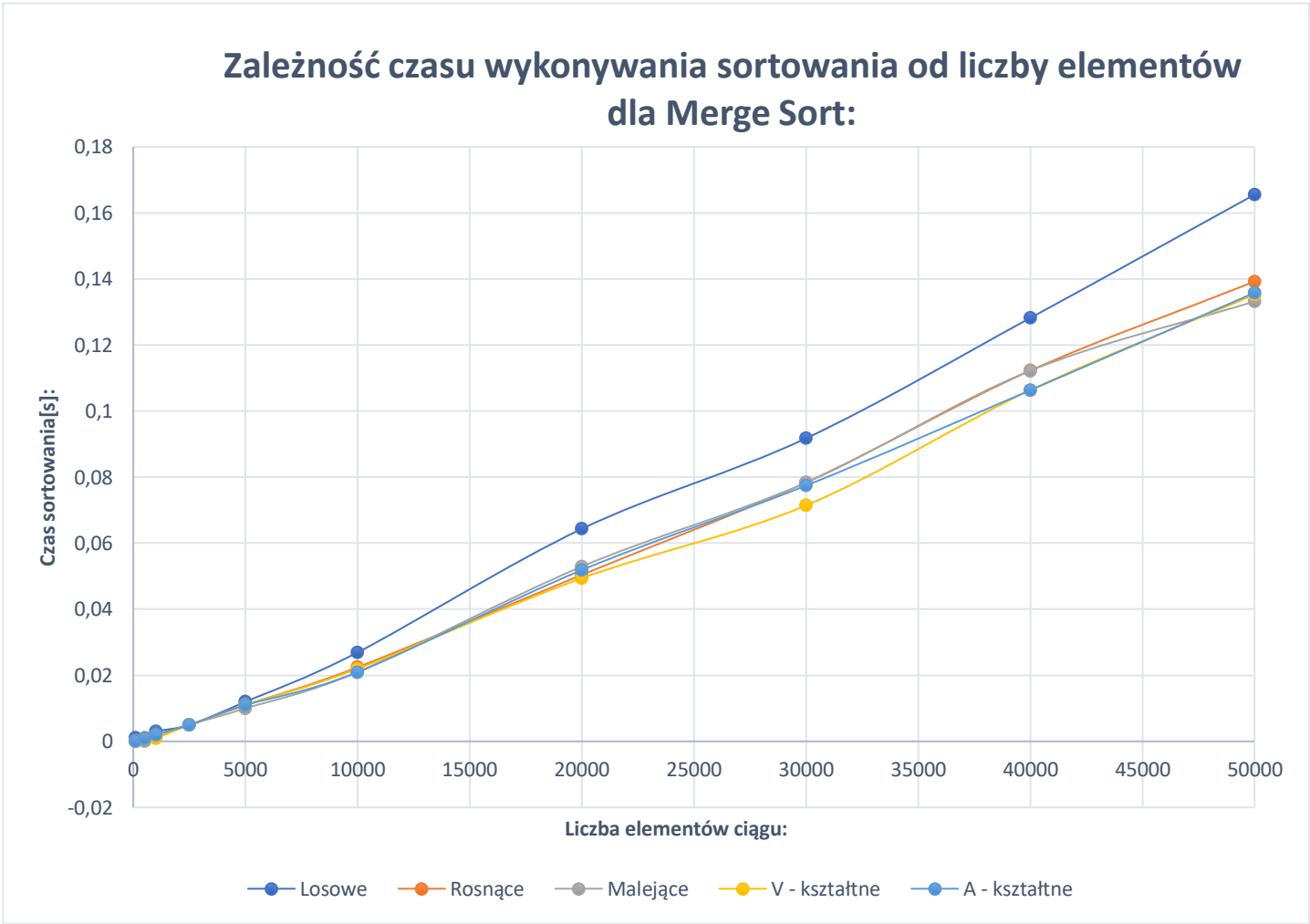
Selection Sort może być używany do sortowania niewielkich zbiorów danych lub do sortowania danych, które już są częściowo posortowane, co zmniejsza ilość operacji, które muszą być wykonane.

Merge Sort

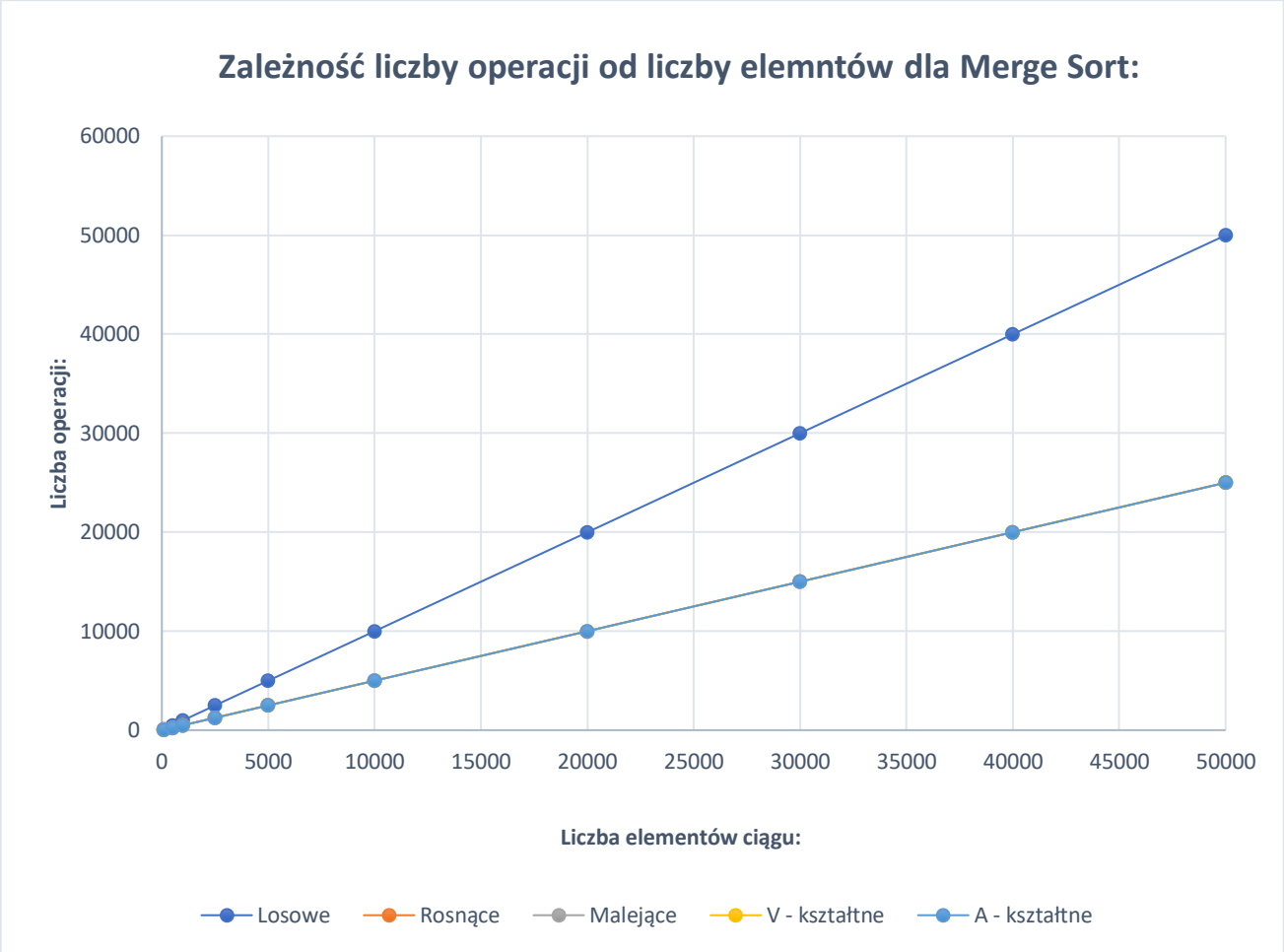
Opis:

Algorytm sortowania Merge Sort to jeden z najskuteczniejszych algorytmów sortowania o złożoności O(n log n). Polega on na dzieleniu zbioru danych na mniejsze podzbiory, a następnie scaleniu ich w sposób posortowany. Algorytm działa dla każdej liczby elementów i jest stabilny, co oznacza, że porządek równorzędnych elementów nie ulega zmianie.

Zależność czasu wykonywania sortowania od liczby elementów dla Merge Sort:



Zależność liczby operacji od liczby elementów dla Merge Sort:



Złożoność:

Algorytm Merge Sort ma złożoność czasową O(n log n) dla wszystkich przypadków. Jest to stabilny algorytm sortowania, co oznacza, że porządek równorzędnych elementów nie ulega zmianie.

Wnioski:

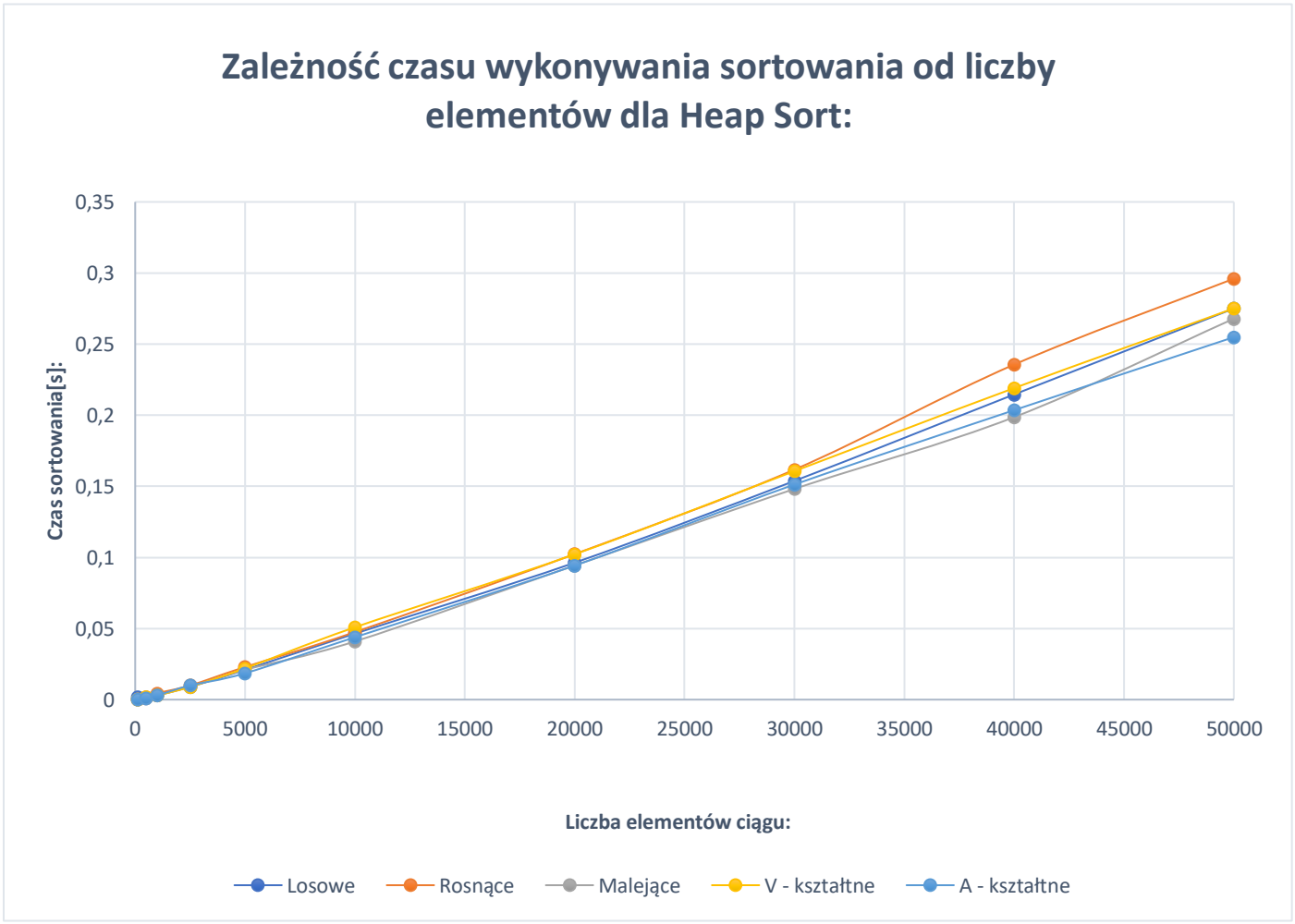
Merge Sort to bardzo wydajny algorytm sortowania. W związku z tym, że Merge Sort wymaga przechowywania elementów w pamięci podręcznej, jego wykorzystanie może być ograniczone w przypadku sortowania dużych zbiorów danych, gdzie może wystąpić brak pamięci.

Heap Sort

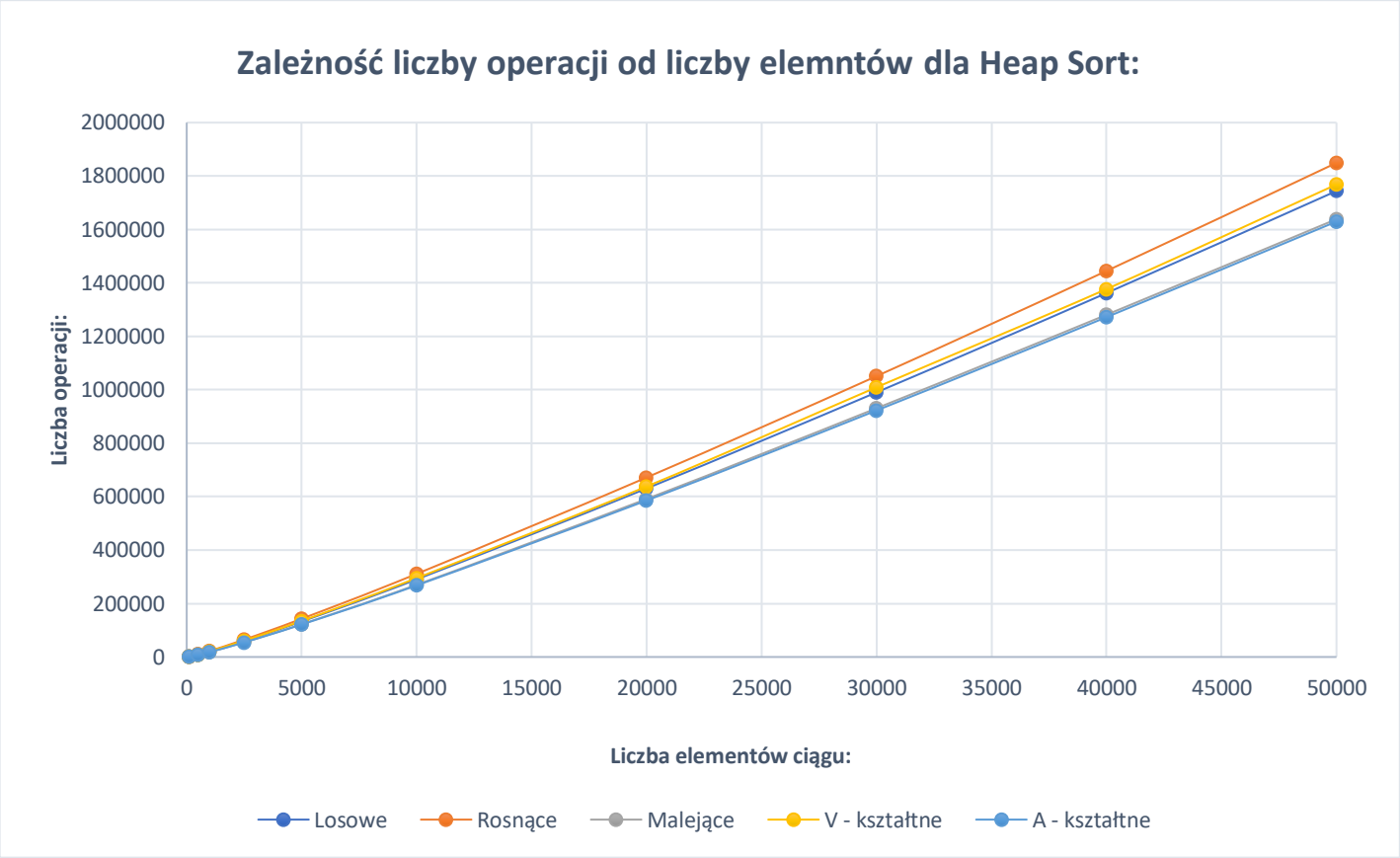
Opis:

Algorytm sortowania Heap Sort wykorzystuje strukturę danych drzewa binarnego, aby sortować elementy. Jego złożoność to O(n log n), co oznacza, że jest to szybki i skuteczny algorytm. Algorytm działa dla każdej liczby elementów i jest stabilny.

Zależność czasu wykonywania sortowania od liczby elemntów dla Heap Sort:



Zależność liczby operacji od liczby elementów dla Heap Sort:



Złożoność:

Złożoność czasowa algorytmu Heap Sort wynosi O(n log n) we wszystkich przypadkach. Jest to stabilny algorytm sortowania. Jest to stabilny algorytm sortowania, co oznacza, że porządek równorzędnych elementów nie ulega zmianie.

Wnioski:

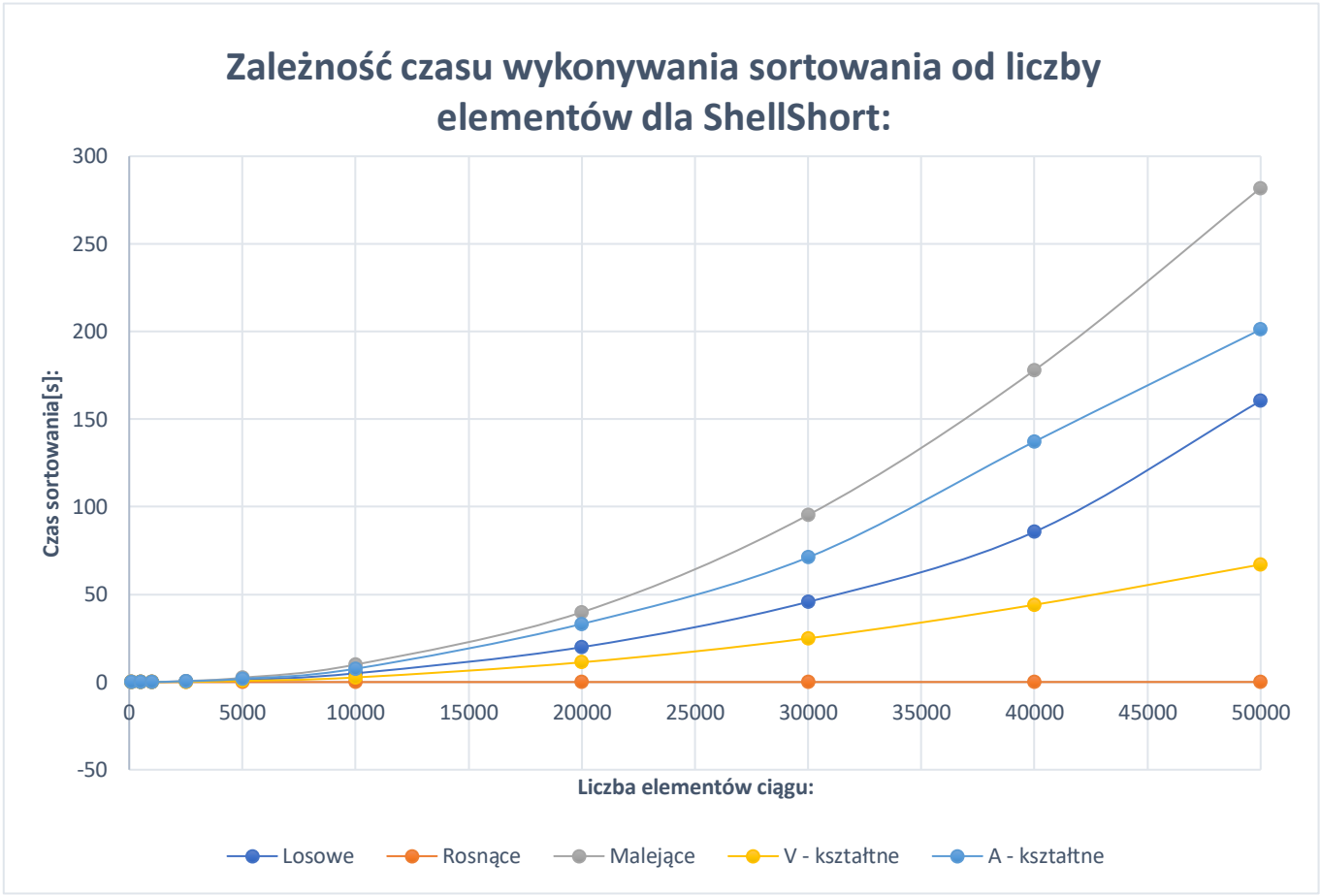
Podsumowując, algorytm Heap Sort jest bardzo skutecznym sposobem sortowania, który działa dla każdej liczby elementów i jest stabilny. Jest to jedna z najszybszych metod sortowania. W praktyce, algorytm Heap Sort często jest wykorzystywany do sortowania dużych zbiorów danych, w szczególności tam, gdzie wymagana jest stabilność sortowania.

Shell Sort

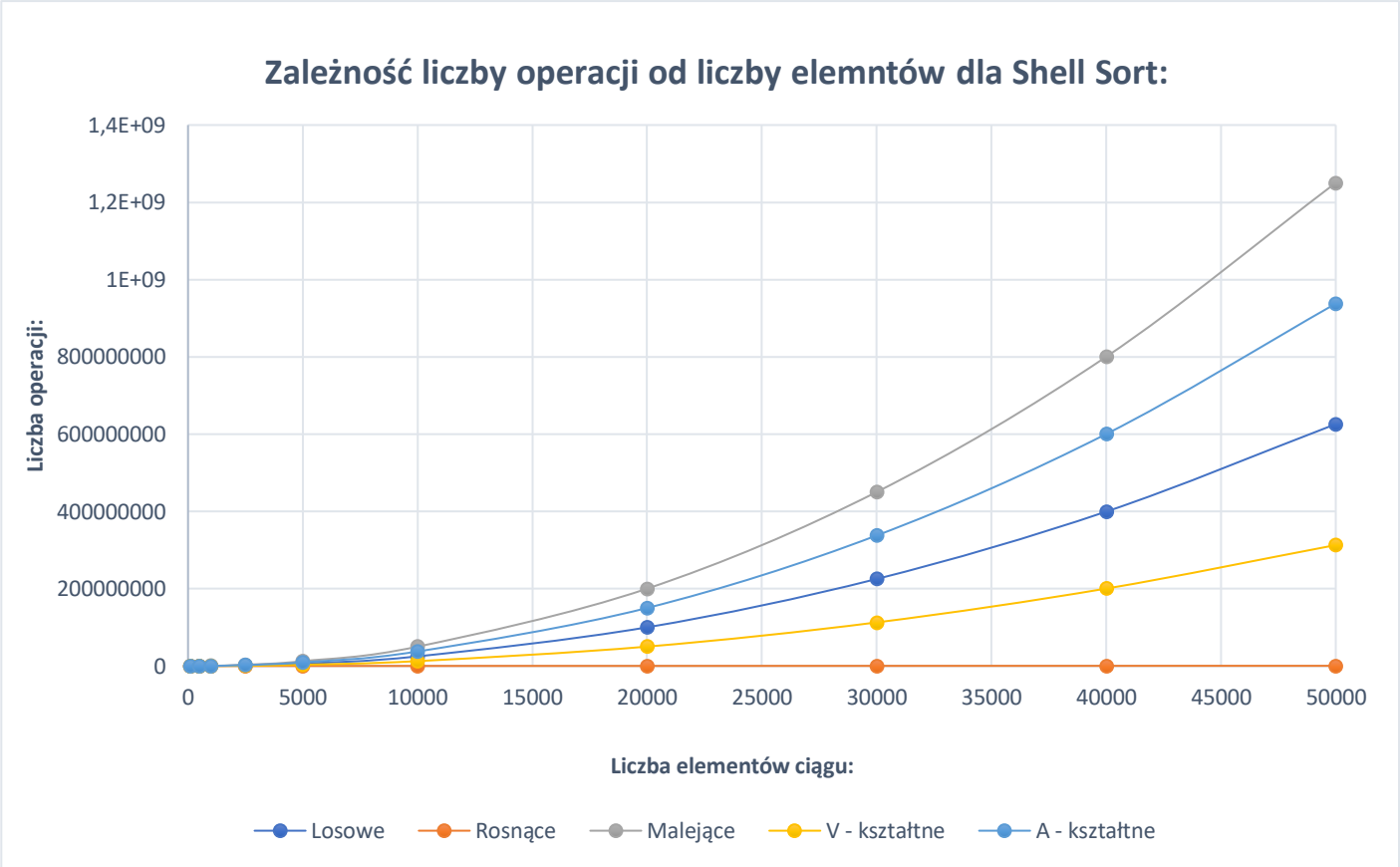
Opis:

Algorytm sortowania Shell Sort jest rozwinięciem Insertion Sort i opiera się na dzieleniu zbioru danych na mniejsze podzbiory. Złożoność średnia algorytmu to O(n^(3/2)), co oznacza, że jest on szybszy niż Insertion Sort dla większych zbiorów danych. Algorytm ma pesymistyczną złożoność O(n^2), co oznacza, że może być wolny dla niektórych zbiorów danych.

Zależność czasu wykonywania sortowania od liczby elementów dla Shell Sort:



Zależność liczby operacji od liczby elementów dla Shell Sort:



Złożoność:

Optymistyczna złożoność czasowa algorytmu Shell Sort zależy od sekwencji długości podzbiorów, ale zazwyczaj jest lepsza niż dla Insertion Sort. Średnia złożoność czasowa wynosi O(n^(3/2)), a pesymistyczna złożoność to O(n^2).

Wnioski:

Algorytm Shell Sort jest bardziej wydajny niż Insertion Sort dla większych zbiorów danych, ale jego złożoność czasowa zależy od wyboru ciągu kroków. Dlatego ważne jest dobieranie odpowiedniego ciągu, który pozwoli na uzyskanie jak najmniejszej złożoności.

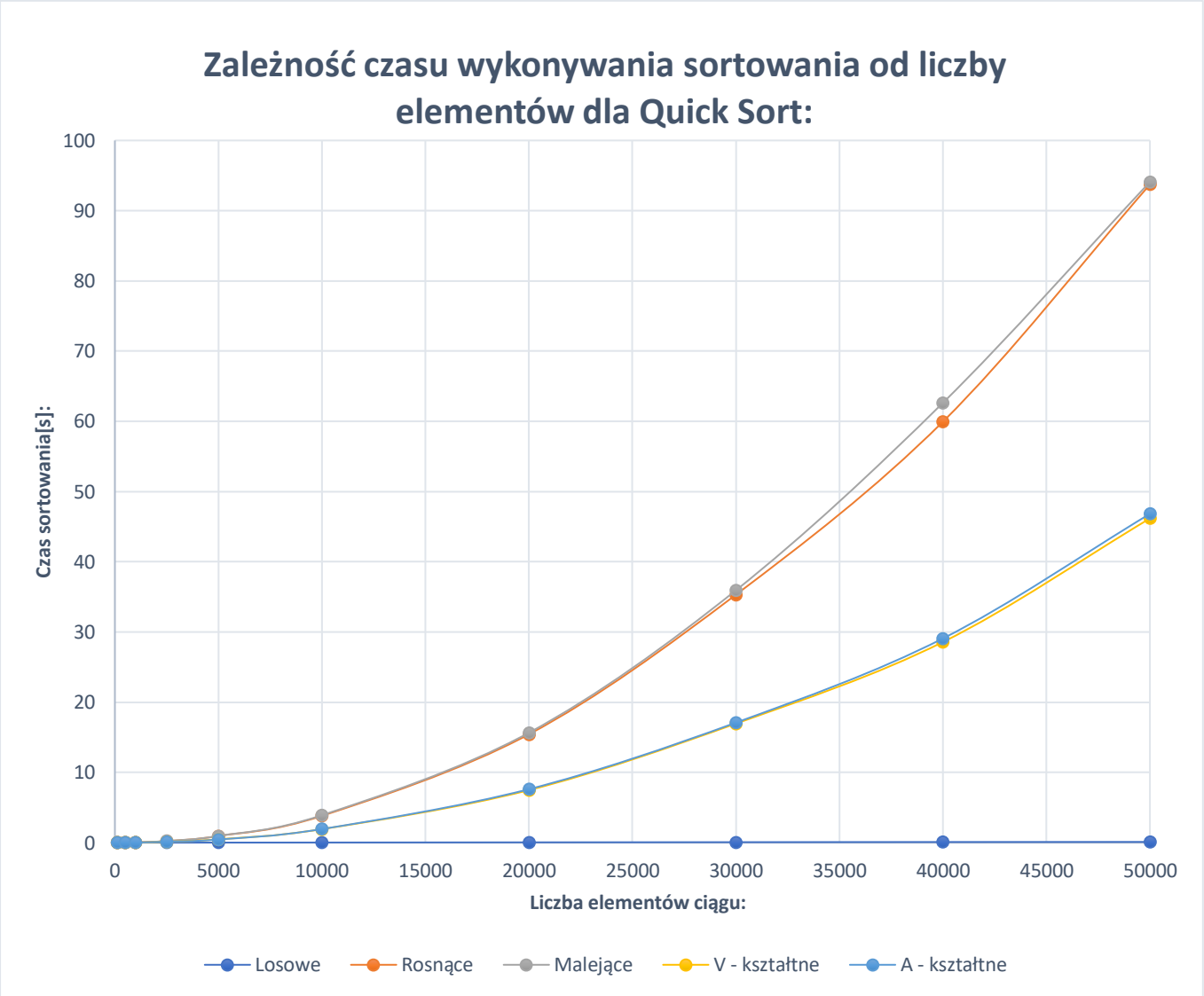
Niestety, Shell Sort nie jest stabilny, co oznacza, że kolejność równorzędnych elementów może ulec zmianie w procesie sortowania. Mimo to, jest to nadal skuteczny algorytm sortowania, szczególnie w przypadku dużej liczby elementów do posortowania.

Quick Sort

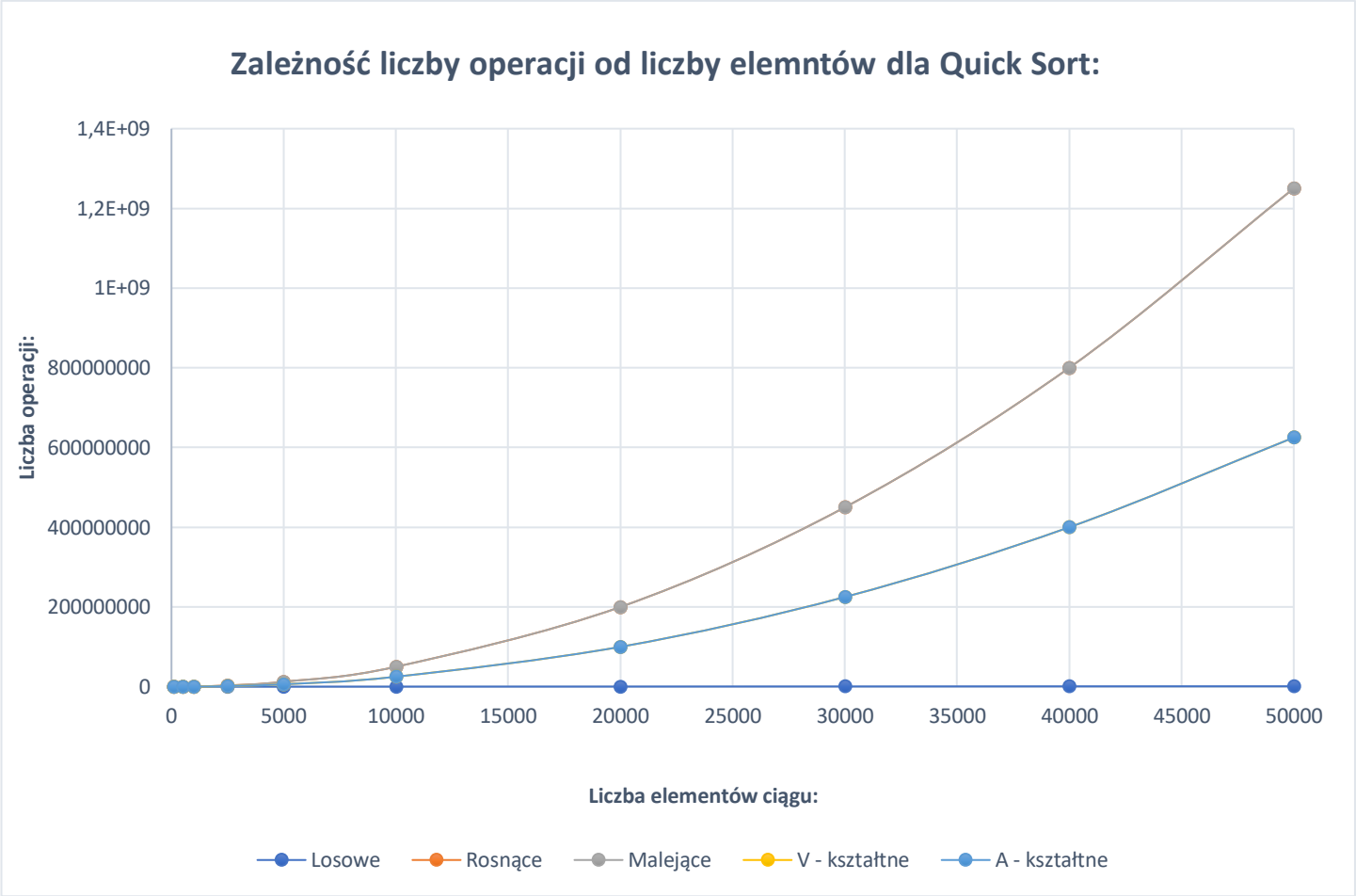
Opis:

Algorytm sortowania Quick Sort to jeden z najszybszych algorytmów sortowania o złożoności średniej O(n log n). Algorytm opiera się na podziale zbioru danych na mniejsze podzbiory, a następnie sortowaniu ich rekurencyjnie. Pesymistyczna złożoność algorytmu to O(n^2), ale może być zmniejszona poprzez wybór losowego pivota lub pivota będącego środkowym elementem ciągu.

Zależność czasu wykonywania sortowania od liczby elementów dla Quick Sort:



Zależność liczby operacji od liczby elementów dla Quick Sort:



Złożoność:

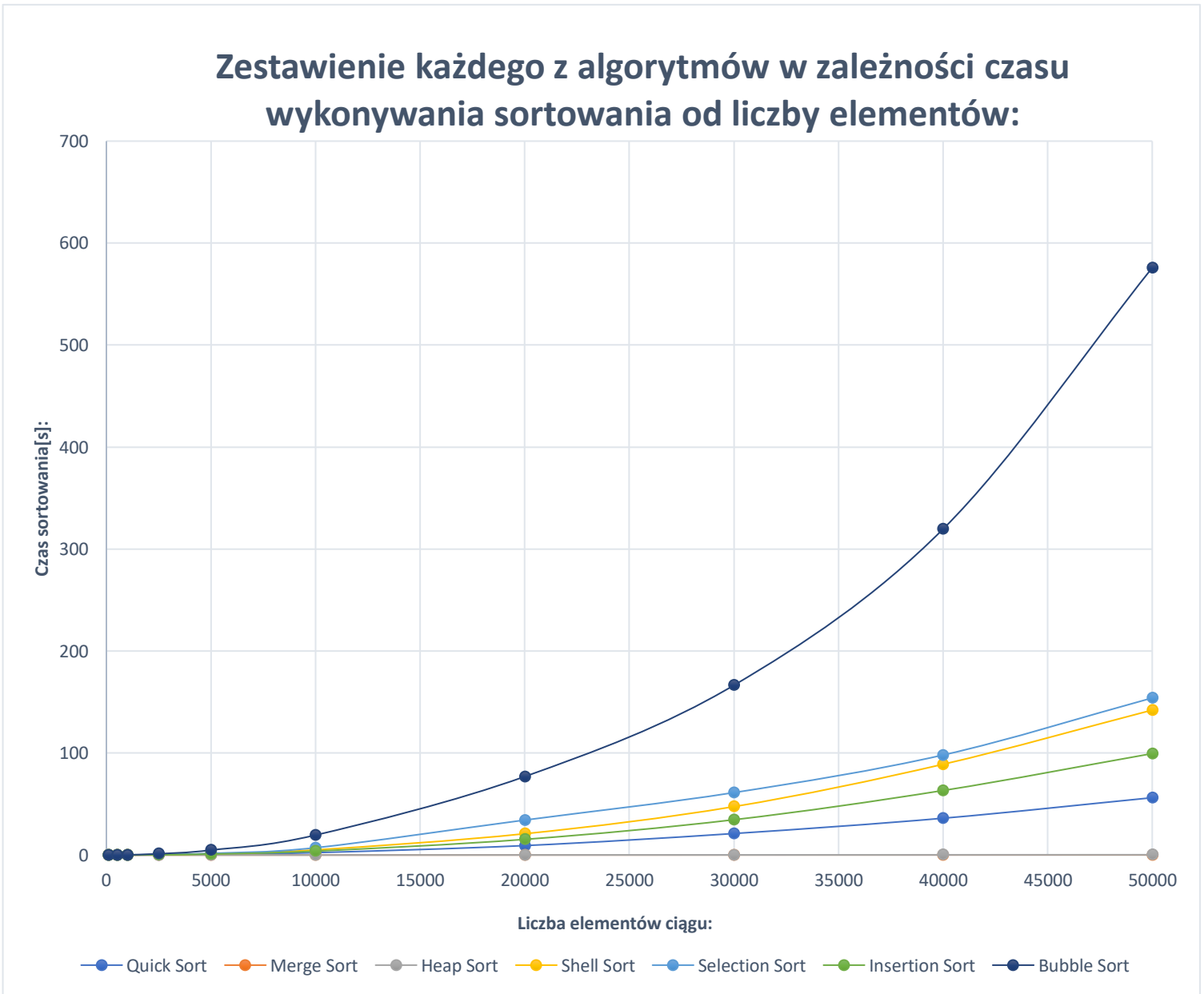
Średnia złożoność czasowa algorytmu Quick Sort wynosi O(n log n), ale pesymistyczna złożoność to O(n^2). Złożoność ta zależy od wyboru pivota, dlatego wybieranie losowego lub środkowego elementu ciągu może zmniejszyć pesymistyczną złożoność czasową algorytmu.

Wnioski:

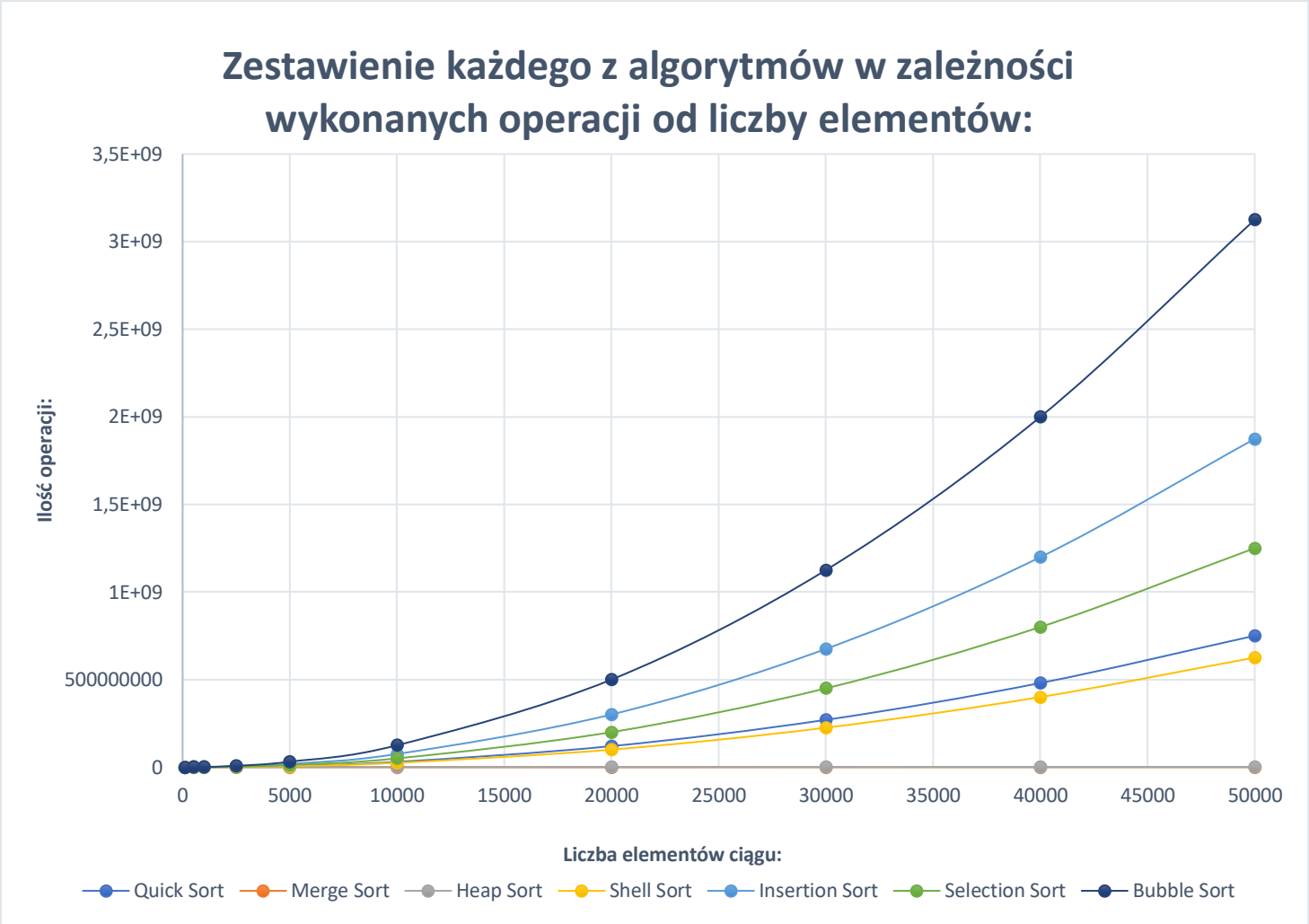
Quick Sort to jedna z najszybszych i najbardziej efektywnych algorytmów sortowania danych. Algorytm jest również bardzo elastyczny i może być łatwo zoptymalizowany, aby działał dla różnych typów danych. Jednakże, Quick Sort ma również pewne wady, takie jak słaba wydajność w przypadku najgorszego scenariusza, który wymaga czasu O(n^2) oraz niskiej stabilności, co może wpłynąć na zachowanie kolejności identycznych elementów. Mimo to, Quick Sort pozostaje jednym z najpopularniejszych algorytmów sortowania i jest często stosowany w różnych aplikacjach.

Zestawienie każdego z algorytmów w zależności czasu wykonywania sortowania od liczby elementów:

Na niniejszych wykresach przedstawiono zestawienie czasów sortowania algorytmów dla różnych typów danych, w tym losowych, rosnących, malejących, A-kształtnych i V-kształtnych. Przeprowadzono sortowanie dla różnej liczby elementów i przedstawiono wyniki w postaci wykresu, który ukazuje jak czas sortowania rośnie wraz ze wzrostem liczby elementów. W analizie porównawczej skupiono się na sześciu popularnych algorytmach sortowania: Merge Sort, Heap Sort, Bubble Sort, Selection Sort, Insertion Sort, Shell Sort oraz Quick Sort. Celem jest zidentyfikowanie, który z algorytmów jest najbardziej efektywny dla określonego typu danych i w jakiej liczbie elementów. Wyniki te mogą być pomocne w wyborze najlepszego algorytmu sortowania dla pewnej liczby elementów.



Zestawienie każdego z algorytmów w zależności wykonanych operacji od liczby elementów:

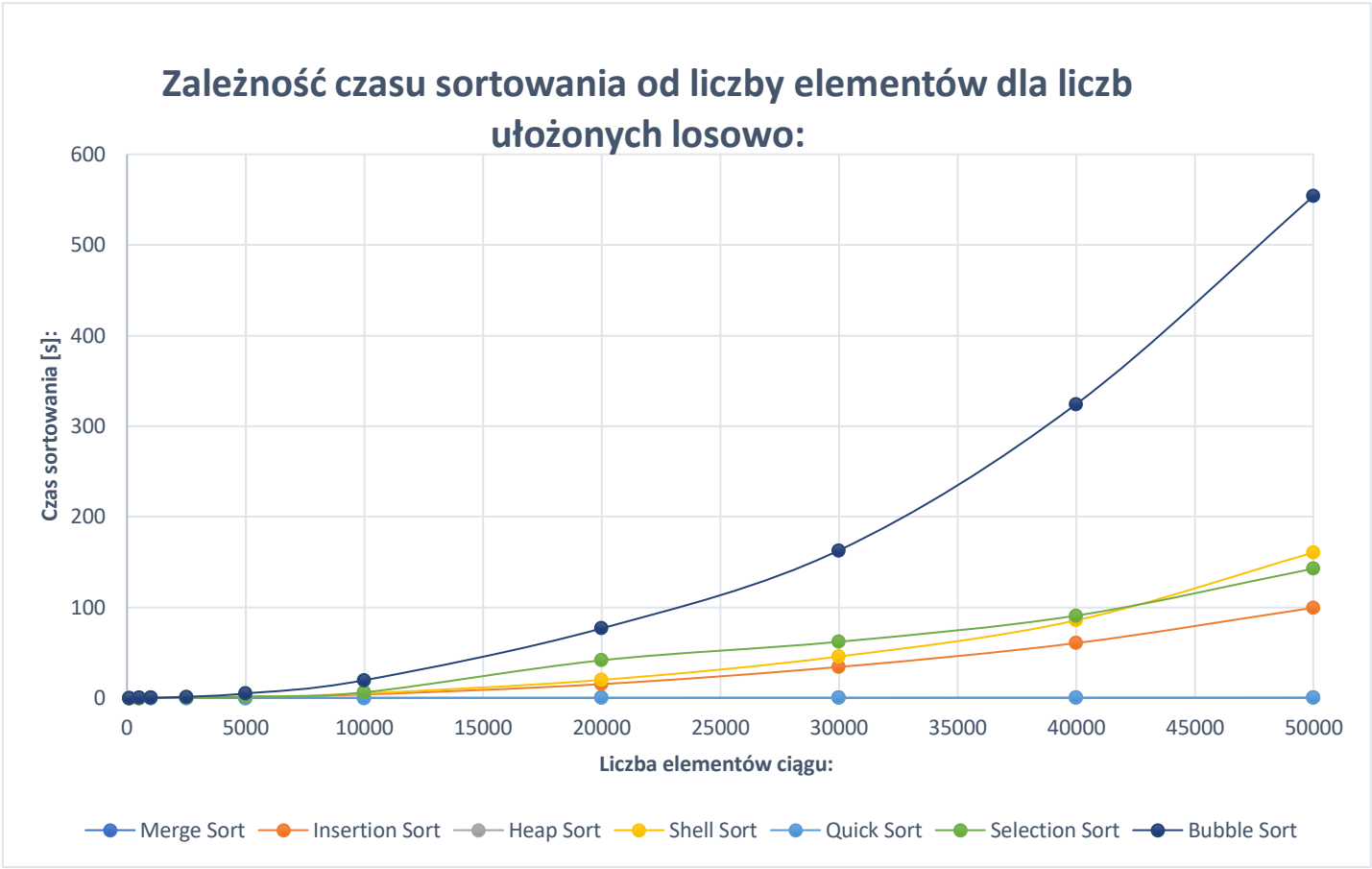


Wnioski:

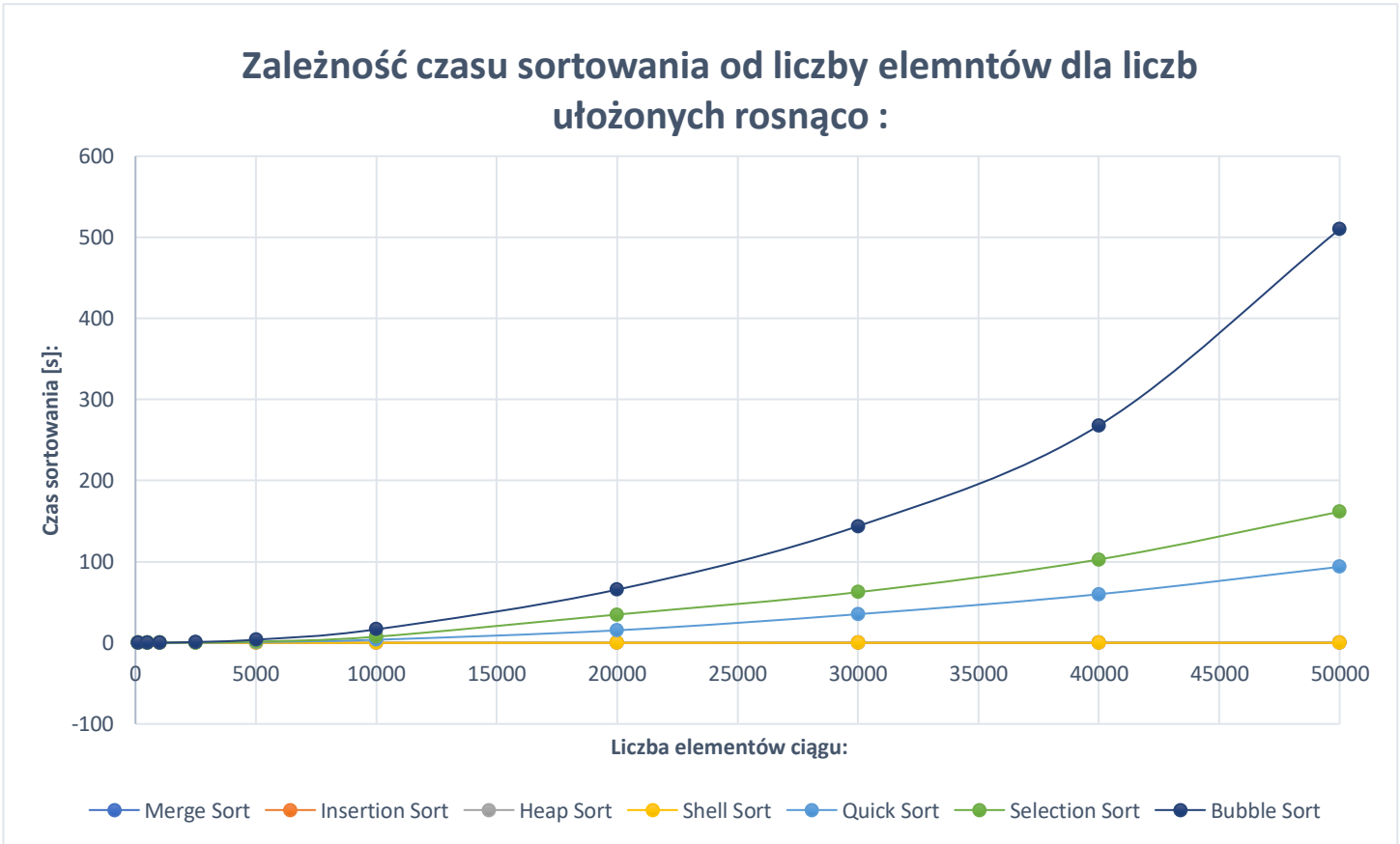
Analiza wyników przedstawionych na wykresie pozwala na wyciągnięcie kilku ważnych wniosków dotyczących efektywności poszczególnych algorytmów sortowania. Po pierwsze, większość algorytmów (Merge Sort, Heap Sort, Insertion Sort, Shell Sort, Bubble, Shell Sort, Quick Sort) radzi sobie podobnie dobrze dla różnych typów danych, w tym dla danych losowych, rosnących, malejących, A-kształtnych i V-kształtnych. Po drugie, Insertion Sort wyróżnia się znacznie gorszymi wynikami, zwłaszcza w przypadku dużych ilości danych, co wskazuje na to, że jest to algorytm stosunkowo mało wydajny dla dużych zbiorów danych.

Wnioski te mogą mieć praktyczne zastosowanie w wyborze optymalnego algorytmu sortowania dla konkretnych zastosowań. Dla mniejszych zbiorów danych, algorytm Insertion Sort może być wystarczająco szybki i nie wymagać zastosowania bardziej złożonych algorytmów. Jednak dla większych zbiorów danych, zastosowanie bardziej wydajnych algorytmów, takich jak Merge Sort, Heap Sort czy Quick Sort, może znacznie przyspieszyć sortowanie i zwiększyć ogólną wydajność systemu.

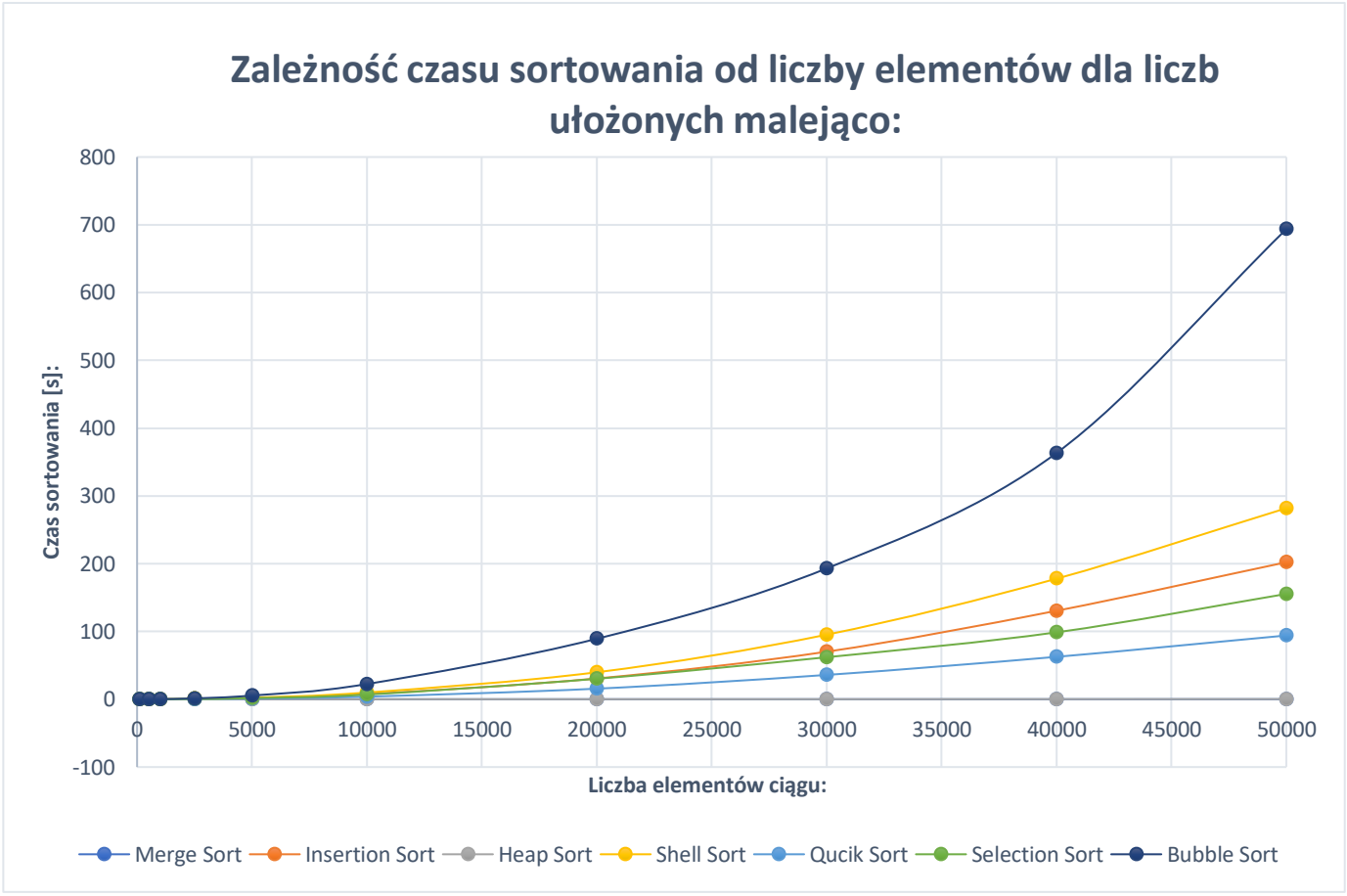
Zależność czasu sortowania od liczby elementów dla liczb ułożonych losowo:



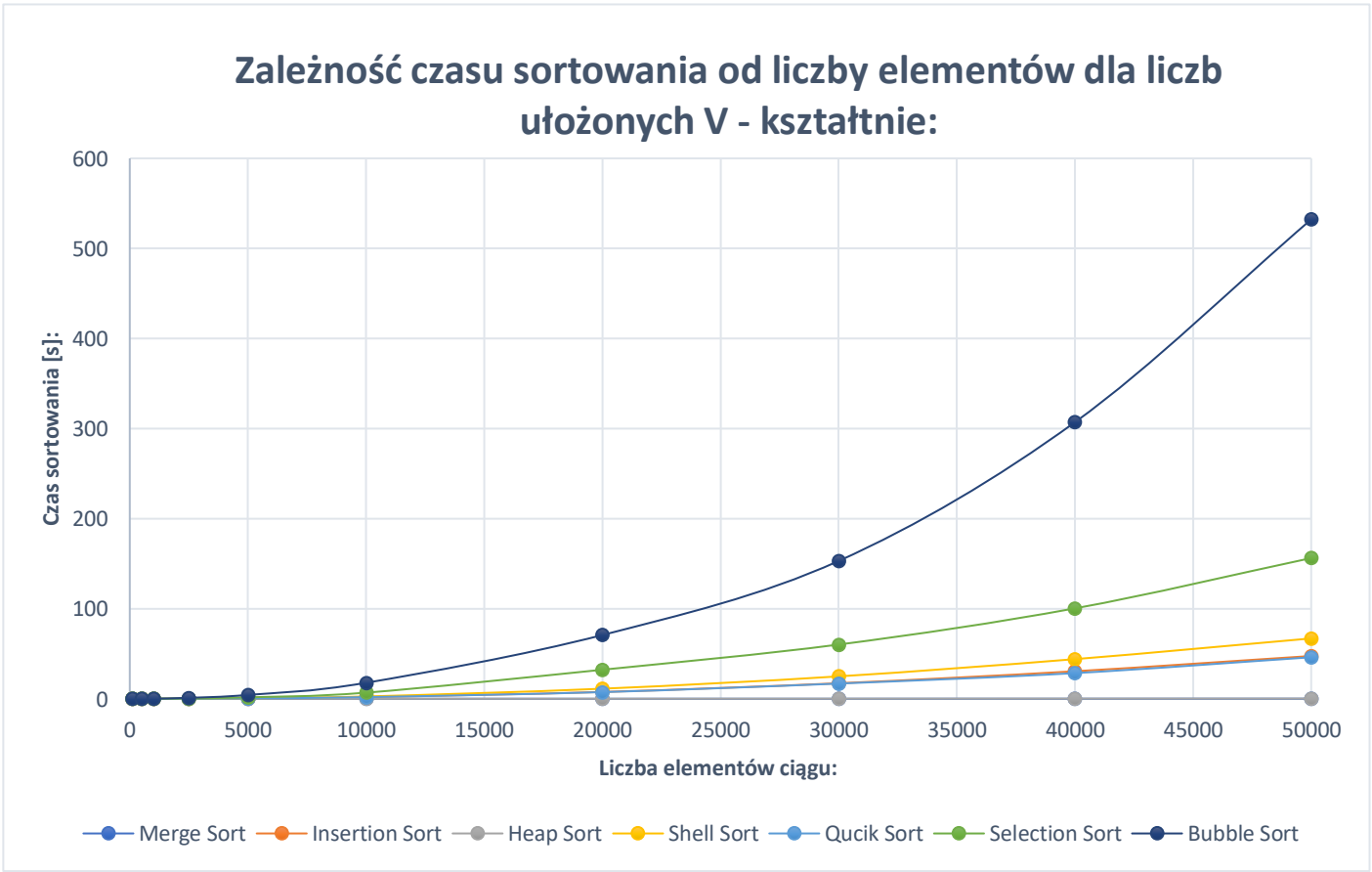
Zależność czasu sortowania od liczby elementów dla liczb ułożonych rosnąco :



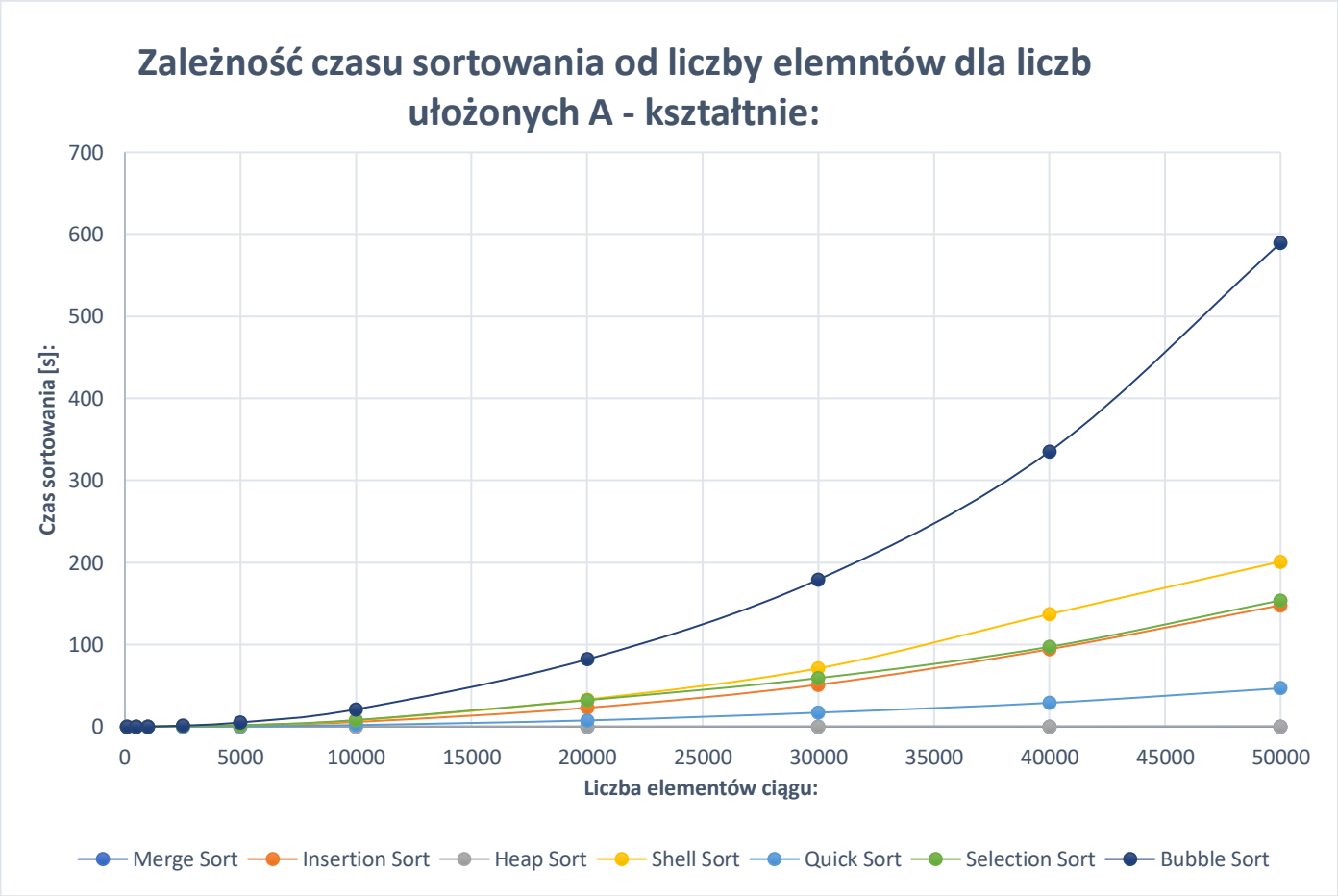
Zależność czasu sortowania od liczby elementów dla liczb ułożonych malejąco:



Zależność czasu sortowania od liczby elementów dla liczb ułożonych V – kształtnie:



Zależność czasu sortowania od liczby elementów dla liczb ułożonych A - kształtnie:



Wnioski:

Wydajność algorytmów sortujących jest znacznie uzależniona od sposobu ułożenia danych wejściowych. Algorytmy takie jak Merge Sort, Heap Sort i Quick Sort działają bardzo dobrze dla danych losowych, ale mogą działać gorzej dla danych ułożonych w sposób nieregularny, takich jak A-kształtne lub V-kształtne. Algorytmy sortowania o złożoności czasowej O(n^2), takie jak Insertion Sort i Selection Sort, Bubble Sort oraz Shell Sort mogą działać lepiej dla danych ułożonych w sposób nieregularny. Dla danych ułożonych w sposób rosnący, algorytmy sortowania zwykle działają najlepiej, a dla danych malejących - najgorzej. Wszystkie algorytmy sortowania mają zwykle złożoność czasową O(nlogn) lub O(n^2), w zależności od algorytmu i sposobu ułożenia danych wejściowych.

Wnioski:

Na podstawie przeprowadzonych testów sortowania losowych, rosnących, malejących, A-kształtnych i V-kształtnych danych przy użyciu różnych algorytmów sortowania, można zauważyć, że wybór algorytmu do sortowania danych ma kluczowe znaczenie dla osiągnięcia maksymalnej wydajności. W zależności od charakterystyki danych, niektóre algorytmy są bardziej skuteczne niż inne.

Przeprowadzone testy pokazują, że algorytmy o niższej złożoności obliczeniowej, takie jak Insertion Sort i Shell Sort, Bubble Sort oraz Selection Sort są bardziej skuteczne w sortowaniu mniejszych zbiorów danych. Natomiast w przypadku większych zbiorów danych, algorytmy o wyższej złożoności obliczeniowej, takie jak Quick Sort, Heap Sort i Merge Sort, są bardziej skuteczne.

Warto również zauważyć, że różnice w czasie wykonywania algorytmów o takiej samej złożoności obliczeniowej są znaczące, szczególnie w przypadku dużych zbiorów danych. Dlatego też, aby zoptymalizować proces sortowania, należy dobrać odpowiedni algorytm w zależności od charakterystyki danych wejściowych.

Wnioskiem z przeprowadzonych testów jest to, że wybór odpowiedniego algorytmu do sortowania danych ma kluczowe znaczenie dla uzyskania maksymalnej wydajności. Zaleca się zatem staranne dobieranie algorytmu w celu zaoszczędzenia cennych zasobów i osiągnięcia jak najlepszych wyników sortowania danych.