Sprawozdanie
Laboratoria nr 1:
SORTOWANIE

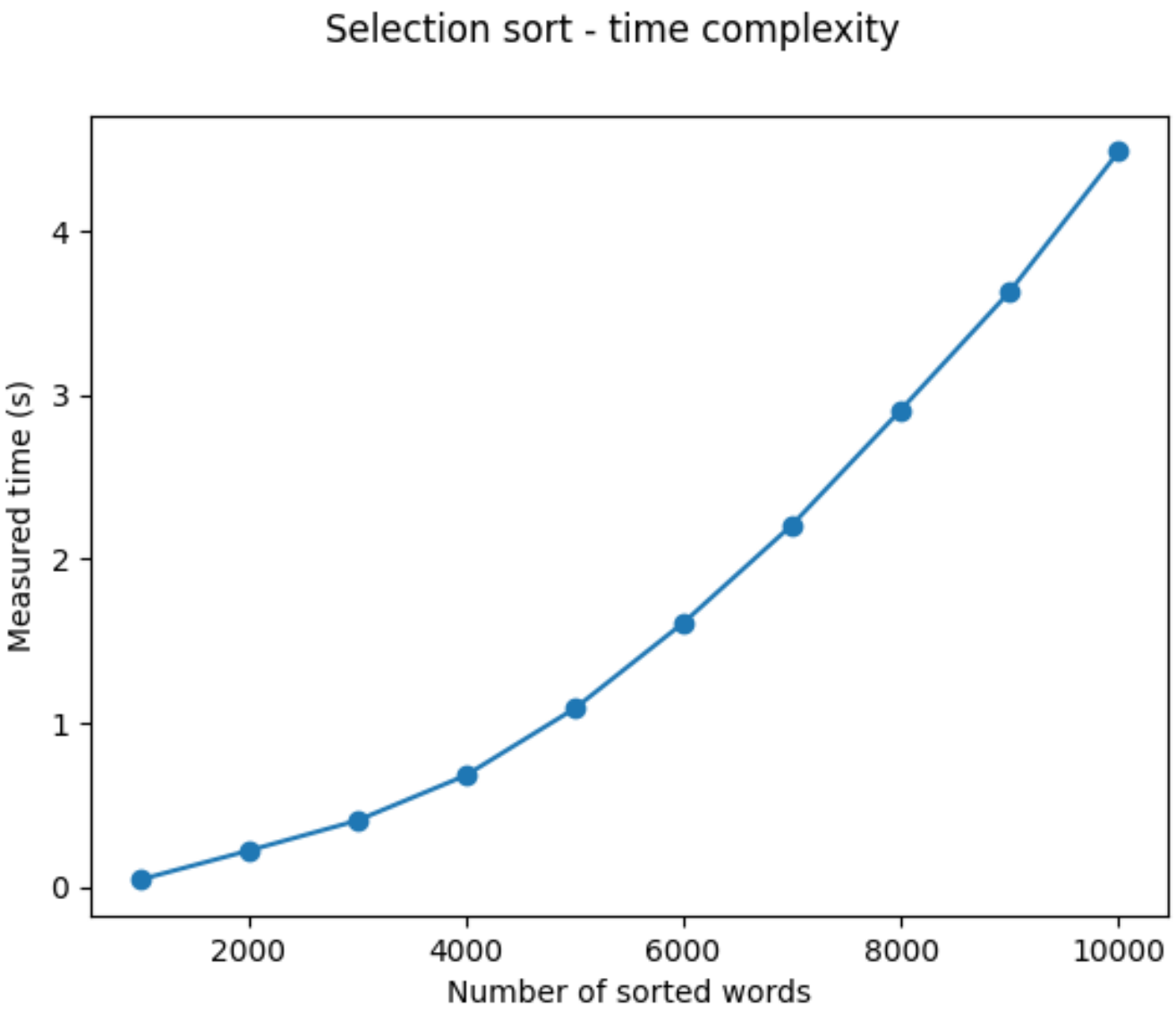
Wykonujący:
Jakub Rozkosz, Hubert Gołębiowski

Podział pracy:
Jakub:
- napisanie algorytmów bubble sort oraz merge sort
- napisanie funkcji do tworzenia wykresów złożoności czasowej pojedyńczych algorytmów
- wykonanie pomiarów prędkości działania wszystkich algorytmów
- wygenerowanie wszystkich wykresów

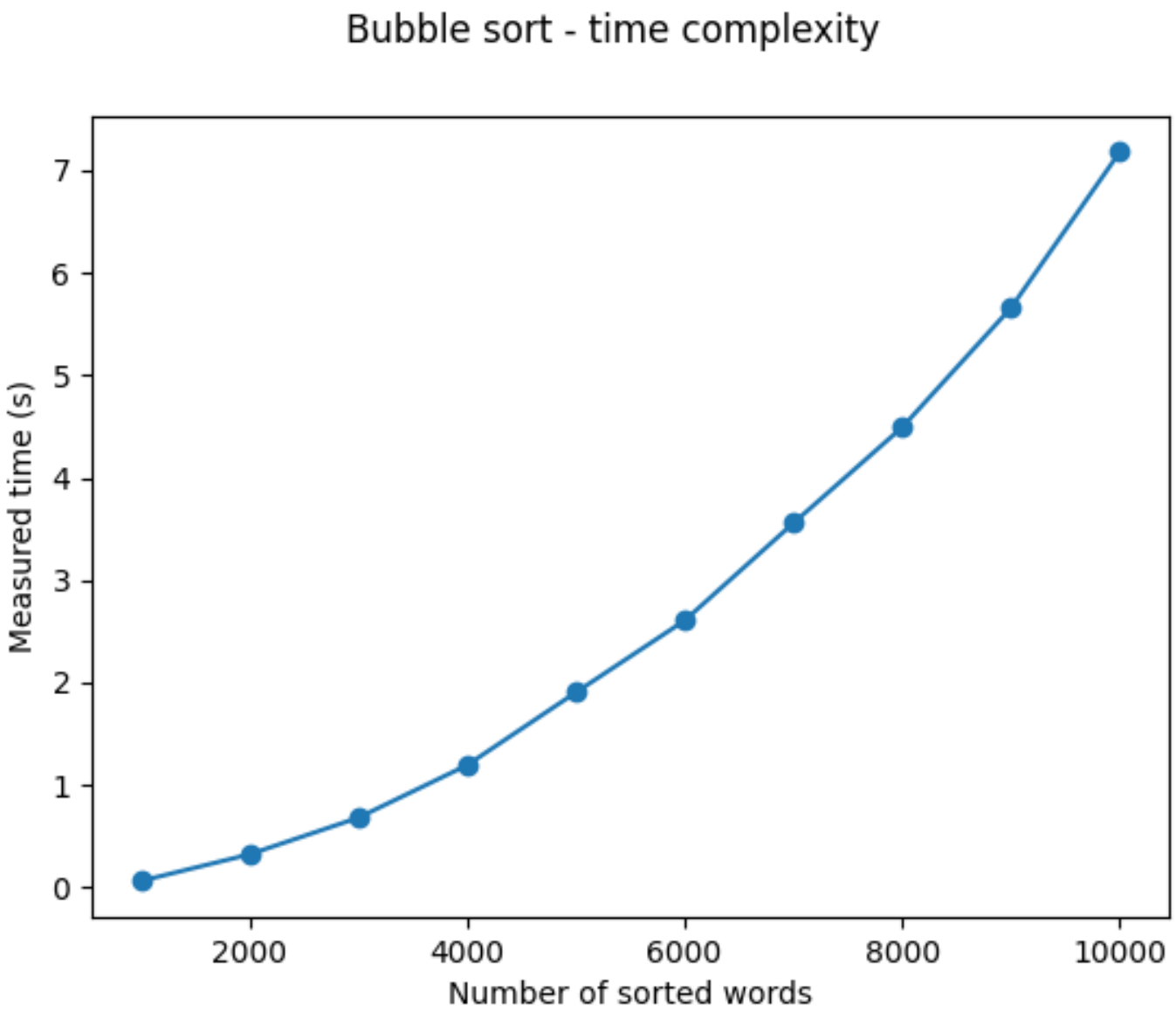
Hubert:
- napisanie algorytmów selection sort oraz quick sort
- napisanie funkcji, która odczytuje plik txt i zwraca listę ze słowami
- napisanie kodu do tworzenia wykresu prezentującego złożoność czasową wszystkich algorytmów
- stworzenie krótkich podsumowań do uzyskanych wyników

Opis projektu:
Aby uniknąć problemów z odczytywaniem pliku tekstowego, pozbyliśmy się z niego wszystkich polskich znaków oraz znaków interpunkcyjnych i zapisaliśmy go w pliku "pan-tadeusz-nowy.txt".
Wszystkie algorytmy pisaliśmy w pythonie w środowisku Visual Studio Code.
Każdy z algorytmów znalazł się w oddzielnym pliku. Dodatkowo stworzyliśmy funkcję 'readfile' do odczytywania pliku tekstowego, który zwraca listę słów o zadanej długości.
W pliku pomiary.py zaimportowaliśmy wszystkie wyżej wymienione pliki. Zawarliśmy w nim funkcję *'draw_a_plot'*, która rysuje i zapisuje do pliku wykres o zadanych wartościach na x-ach oraz y-kach. Do tworzenia wykresów wykorzystaliśmy moduł *pyplot* z biblioteki *matplotlib.*
Przeprowadziliśmy pomiary dla dziesięciu list zawierających od 1000 do 10000 elementów. Do zmierzenia czasu działania funkcji wykorzystaliśmy bibliotekę *timeit.*
Zauważyliśmy, że przy wykonaniu tylko jednego pomiaru dla każdej z list czasami występują pewne nieoczekiwane odchyły, więc aby unormować wyniki wykonywaliśmy pomiar dziesięciokrotnego wykonania funkcji sortowania dla bubble i selection oraz tysiąckrotnego dla merge i quick. Następnie dzieliliśmy te wyniki kolejno przez 10 oraz 1000 i wychodził nam uśredniony czas działania.

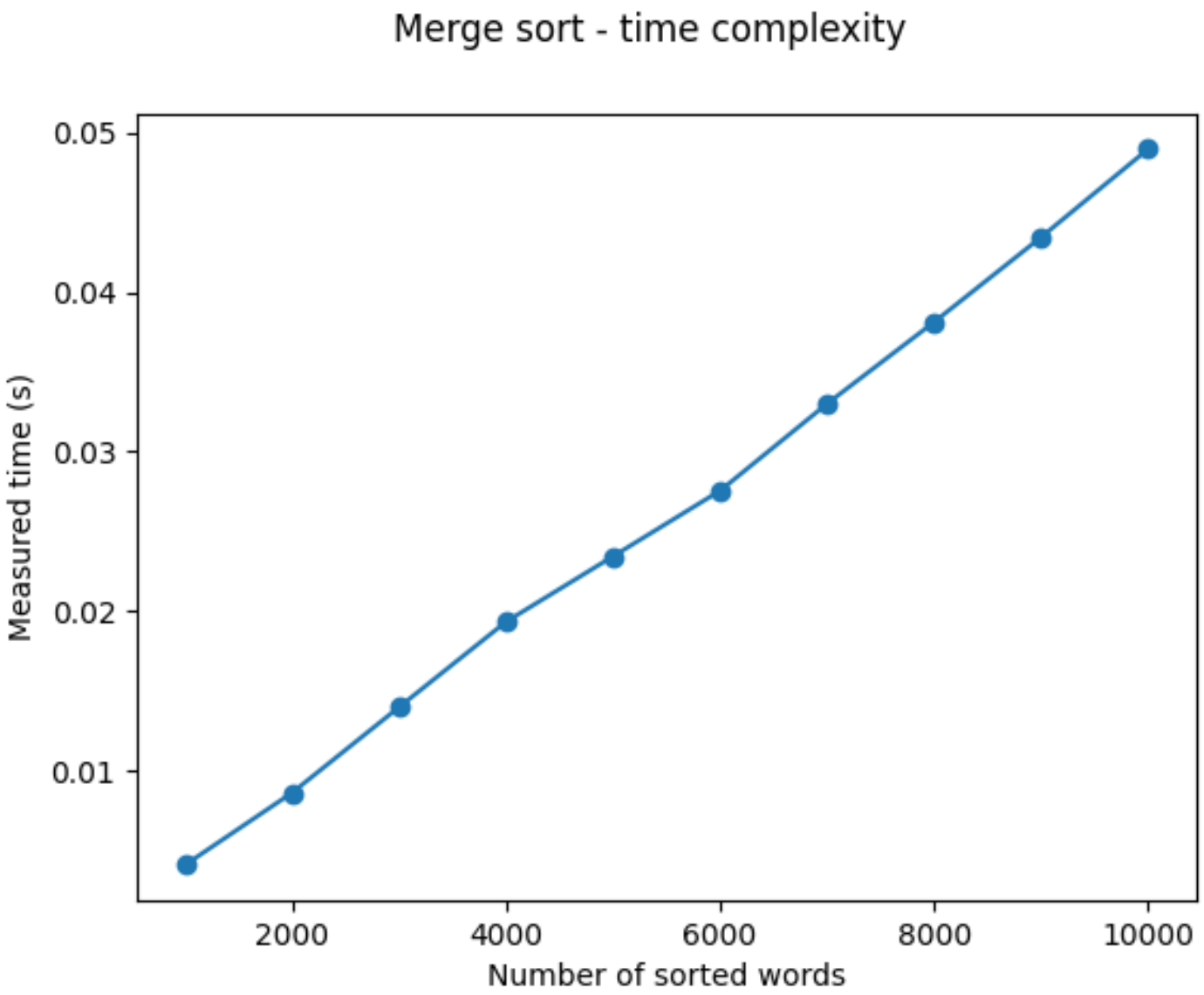
Wyniki:



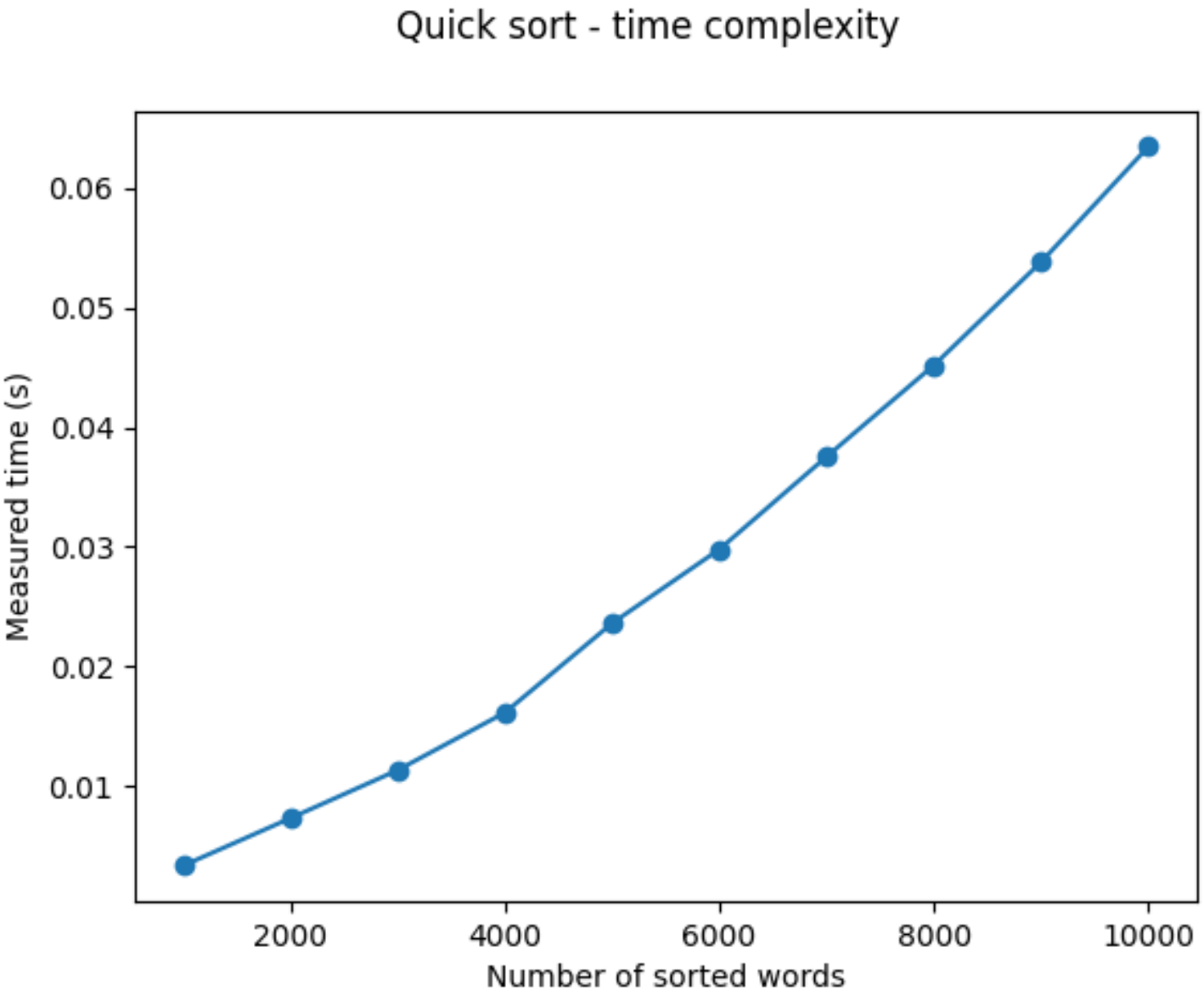
Selection sort po zaimplementowaniu wydawał się powolny, co nie jest zaskakujące, jako, że jest algorytmem o **złożoności kwadratowej**. Potwierdza to wykres funkcji, na którym łatwo zauważyć nieliniowy przyrost, funkcja jest wyraźnie wklęsła. Co ciekawe jest ona też gładka (zbliżona do faktycznej paraboli), co łatwo można wytłumaczyć faktem, że liczba porównań Selection sort'a nie zależy od danych wejściowych, a jedynie od ich ilości.



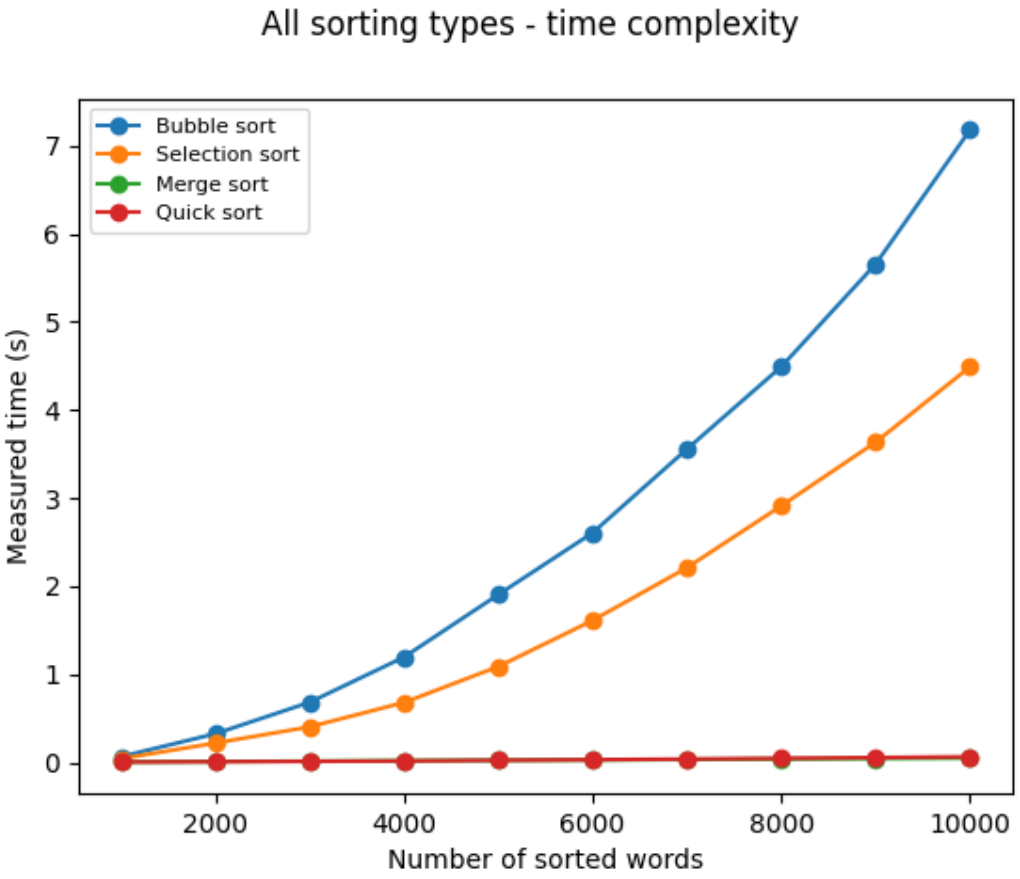
Bubble sort, jak można się było spodziewać, okazał się najgorszym z algorytmów. Tutaj również dobrze widać, że wykres funkcji jest wklęsły, a także, że jest gładki (co można wytłumaczyć analogicznie jak w przypadku Selection sort'a). Obrazek odwzorowuje przebieg funkcji $\mathbf{n^2}$.

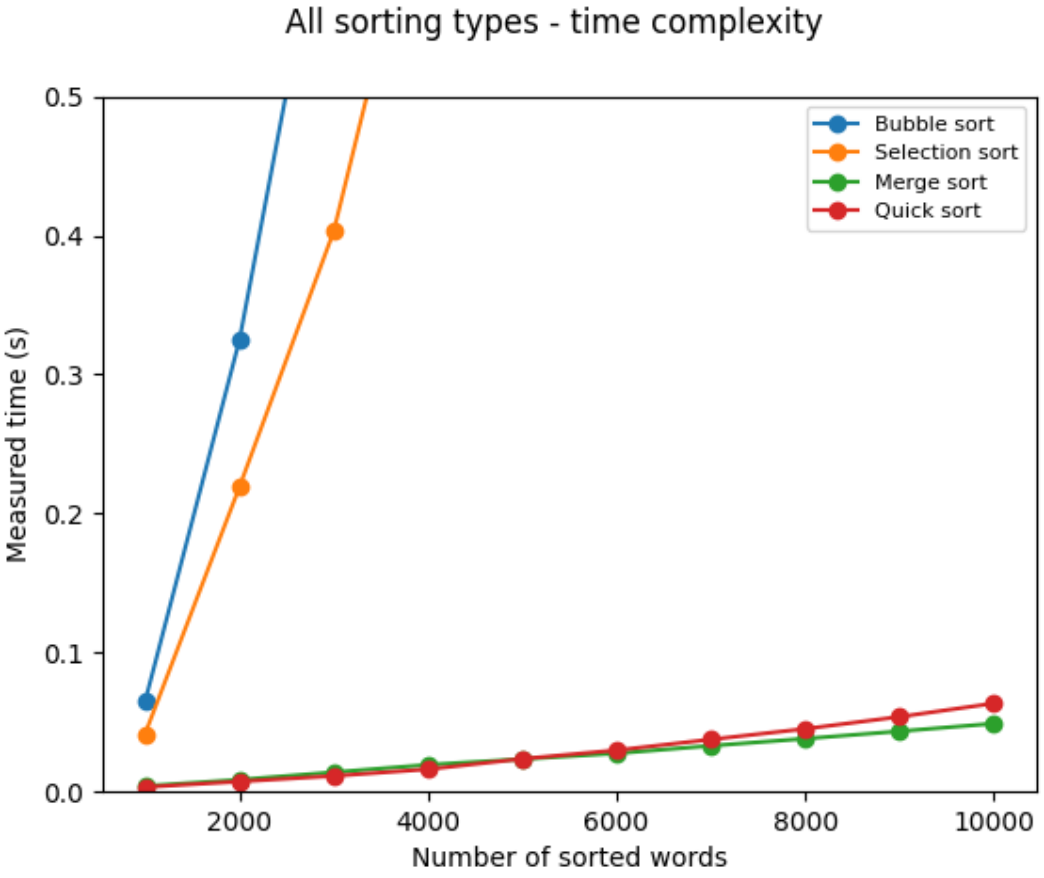


W przypadku Merge sort'a można było odczuć ogromną poprawę w stosunku do poprzednich algorytmów. Stosując ten algorytm wyniki pojawiały się natychmiast, co potwierdza wykres (nawet dla 10tys. danych czas liczony był w setnych sekundy). Co więcej, z wykresu możemy wyczytać, że przyrost był praktycznie liniowy, co nie jest zaskakujące, gdyż funkcje typu **nlogn**, dla niedużych ilości wartości, mają zbliżony przebieg do funkcji liniowych.

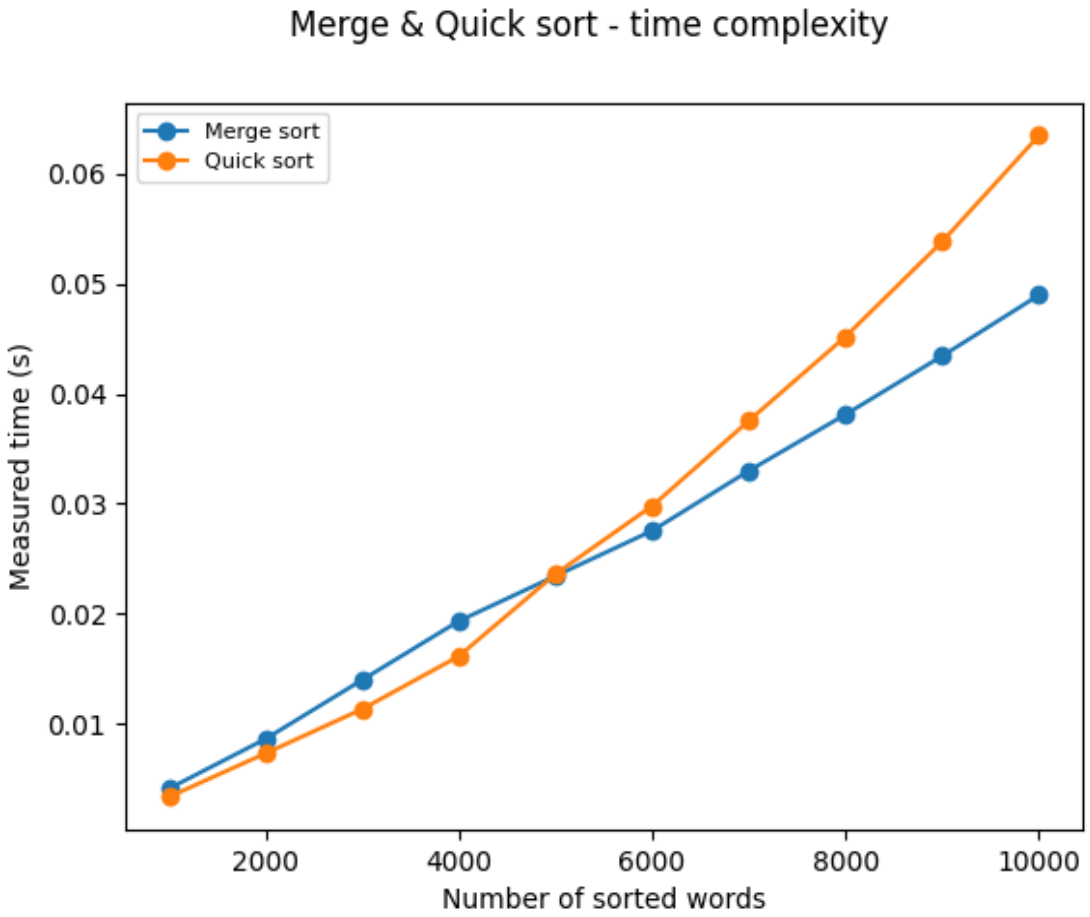


Ostatnim algorytmem był Quick sort i tutaj podobnie jak w przypadku Merge sort'a czas był mierzony w ułamkach sekund. Wykres praktycznie odwzorowuje przebieg funkcji **nlogn** (złożoność czasowa dla średniego przypadku). Tutaj możemy jednak dostrzec lekką wklęsłość, której nie było na poprzednim wykresie. Możliwe, że jest ona dodatkowo spowodowana tym, że w przeciwieństwie do Merge sort'a, który w każdym z przypadków (worst, average, best) ma taką samą złożoność czasową - nlogn, Quick sort w zależności od przypadku ma różną złożoność. Dla *najgorszego przypadku* (tablica z początku posortowana) ma złożoność n^2.


All sorting types - time complexity
Bubble sort
Selection sort
Merge sort
Quick sort
Measured time (s)
0
1
2
3
4
5
6
7
2000
4000
6000
8000
10000
Number of sorted words


All sorting types - time complexity
Bubble sort
Selection sort
Merge sort
Quick sort
Measured time (s)
0.0
0.1
0.2
0.3
0.4
0.5
2000
4000
6000
8000
10000
Number of sorted words

Powyżej mamy porównanie wszystkich algorytmów. Widać na nim wyraźnie przepaść pomiędzy algorytmami o złożoności nlogn, a kwadratowymi. Co więcej, wyraźnie widać, że występuje znaczna różnica pomiędzy Bubble sortem a Selection sortem. Jest to spowodowane tym, że Selection sort wymaga mniej przestawień elementów niż bąbelkowa wersja sortowania. Jeśli natomiast chodzi o różnice pomiędzy algorytmami Merge sort i Quick sort, lepiej prezentują się one na poniższym wykresie:



Dla małych liczb słów oba algorytmy zdają się być podobnie szybkie, jednak wykres Quick sort'a zaczyna rosnąć trochę szybciej, co by sugerowało, że dla zadanych większych wartości może odstawać od Merge sort'a. Metoda scalania jest więc najefektywniejszym algorytmem do sortowania bardzo dużych zbiorów danych. Quick sort dla niefortunnego przypadku musi wykonać dużą liczbę porównań, co wydłuża jego pracę.

Wnioski:

Stosując wybrane algorytmy mogliśmy na własnej skórze przekonać się, jak wolne potrafią być algorytmy o złożoności czasowej n^2, jednocześnie obserwując, że złożoność nlogn jest całkiem zadowalająca. Potwierdziły się też wcześniej nam znane zależności: Bubble sort gorszy od Selection sorta oraz Quick sort gorszy od Merge sort'a. Merge oraz Quick sort okazali się być królami wśród testowanych przez nas algorytmów. Pokazuje to potęgę metody *'divide and conquer'*. Uzyskane wyniki sugerują, że algorytmy zostały zaimplementowane poprawnie.