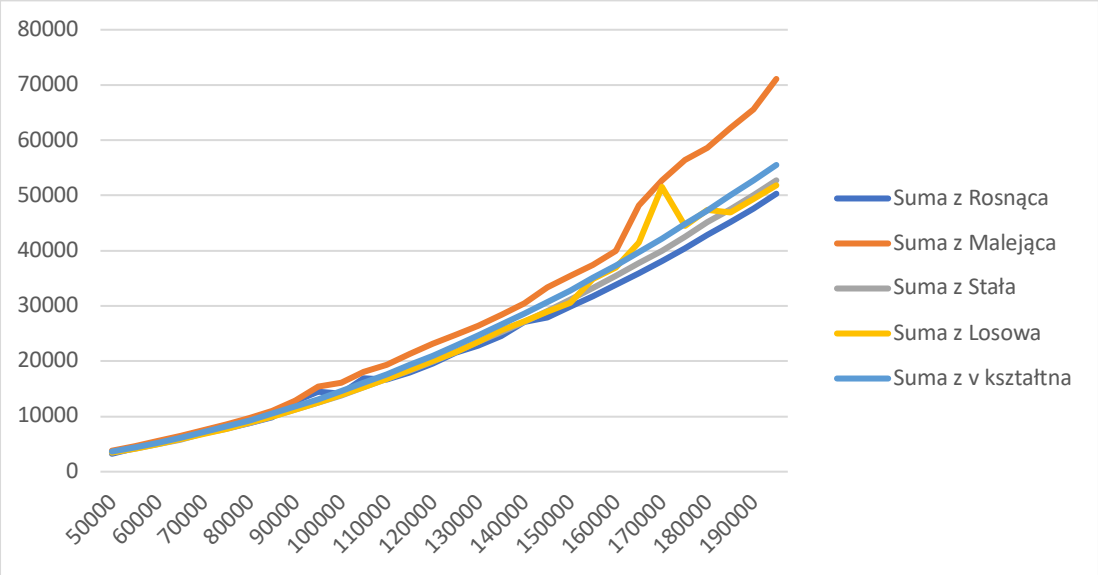
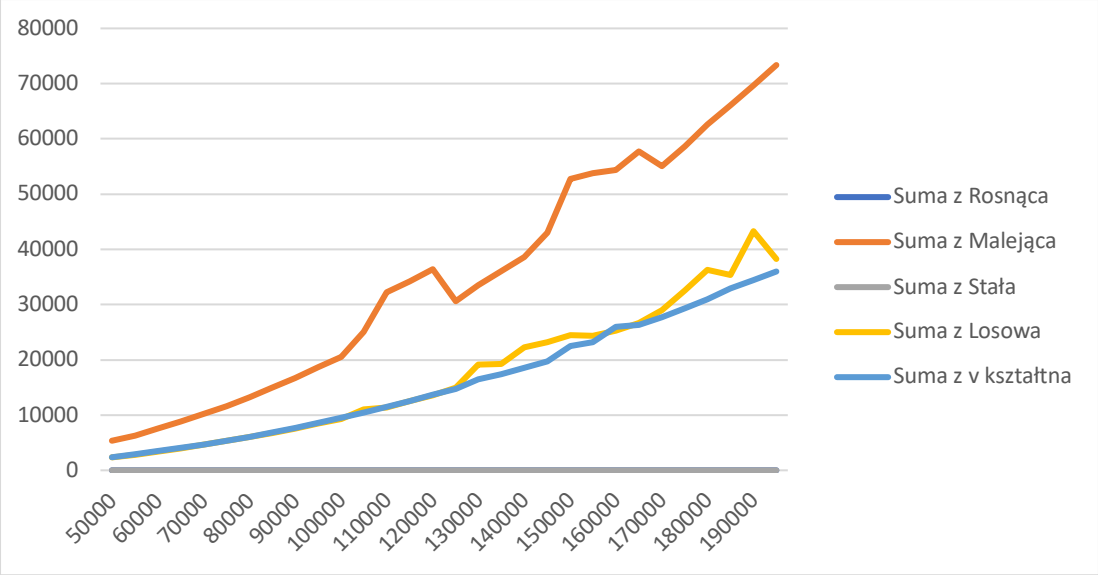
Projekt 3

Projekt zawiera trzy części, które będą poświęcone analizie algorytmów sortujących tablice liczbowe. Za podstawę do badań utworzyliśmy tablice: rosnącą, malejącą, stałą, losową, v-kształtną, A-kształtną o wielkości od 50000 do 200000 liczb, które w następnym etapie zostały poddane sortowaniu przez algorytmy: SelectionSort, InsertionSort, CocktailSort, HeapSort oraz QuickSort . W ramach projektu mierzony został czas wykonywanych sortowań i każde z nich było powtarzane 30 razy zwiększając tablice o 5000.

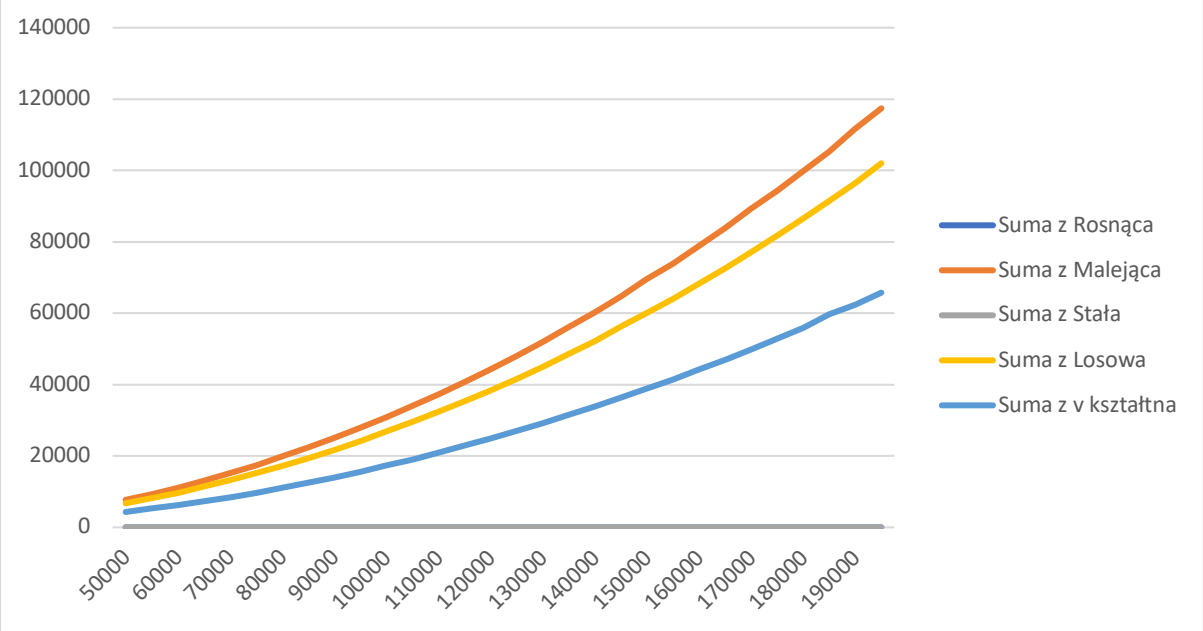
Część I - działanie algorytmów dla różnych typów zbiorów liczbowych



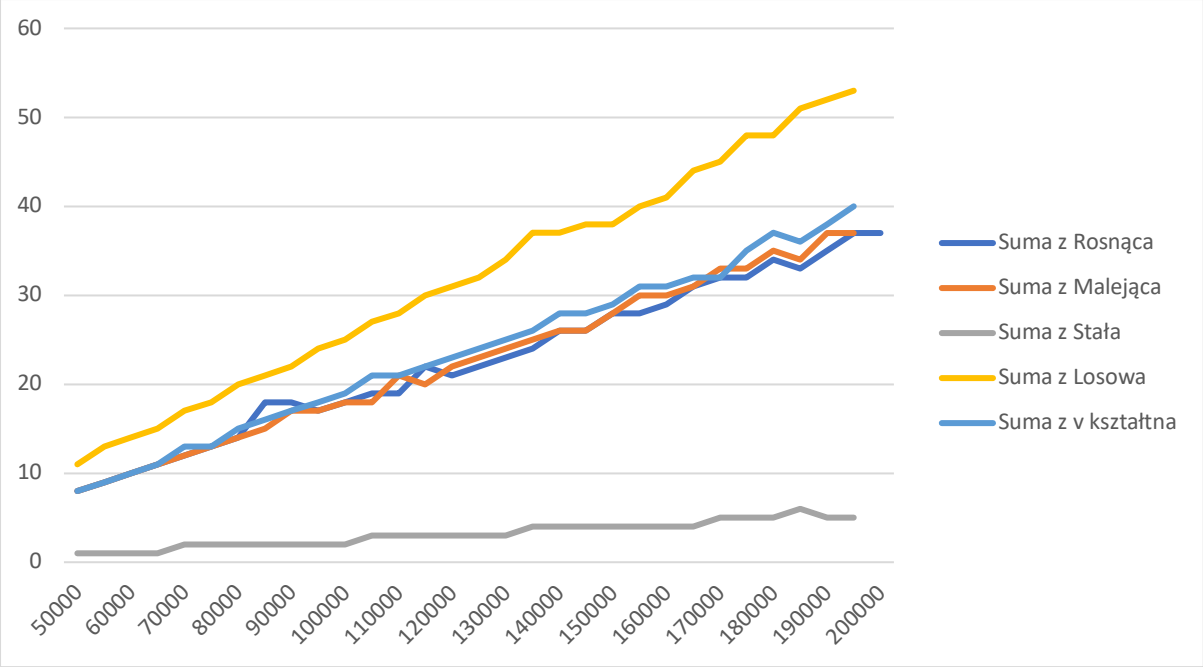
Algorytm SelectionSort jest najwolniejszym a powodem tego jest złożoność O(n^2). Jak widać na wykresie czas działania jest bardzo zbliżony.



W przypadku Algorytmu InsertionSort nie jest już tak jednolity, potrafi bardzo szybko zakończyć proces sortowania. W najgorszym przypadku, którym jest tablica malejąca czas wykonywania sortowania wzrasta n^2.

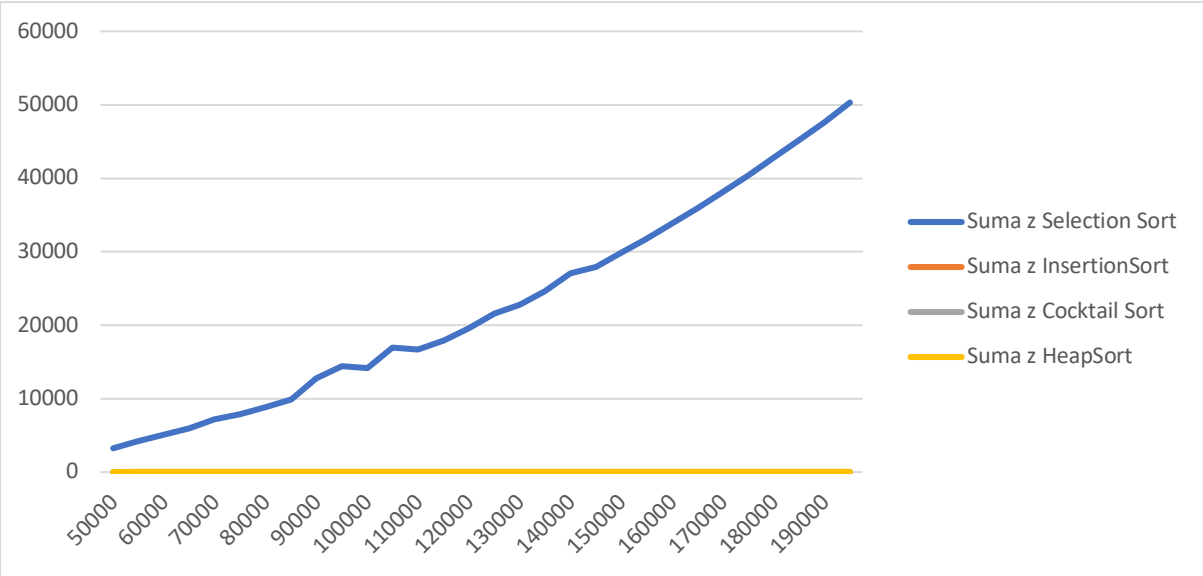


Trzecim algorytmem jest CocktailSort który ma podobną złożoność do poprzedniego algorytmu natomiast jego czas sortowania są dłuższe.

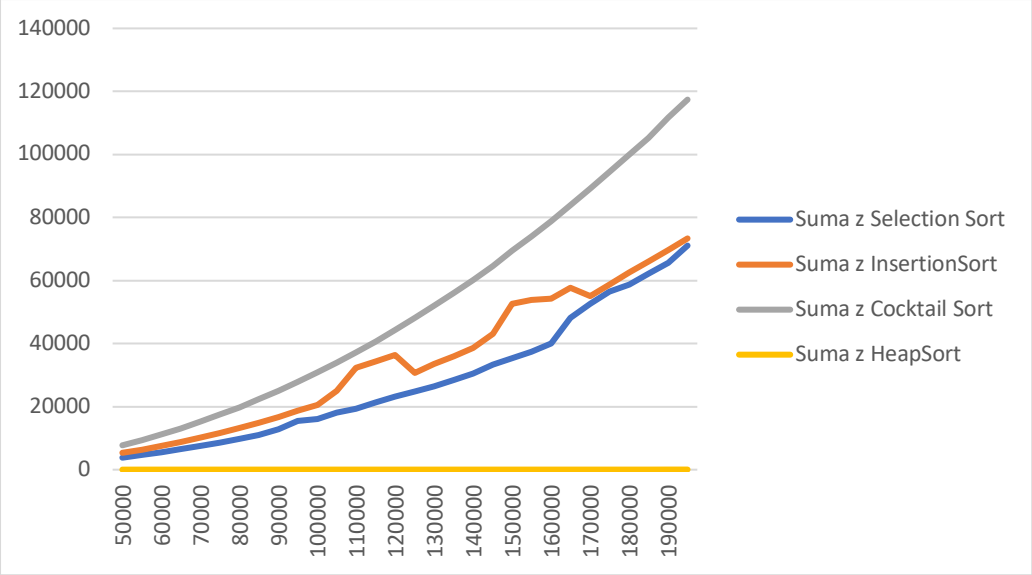


Złożoność kolejnego algorytmu wynosi 0(n log n). Dla tego typu sortowania nie ma większego znaczenia jaki typ danych wejściowych obsługuje. Algorytm HeapSort sprawia że właśnie on jest dotychczas najszybszy oraz bardzo uniwersalny.

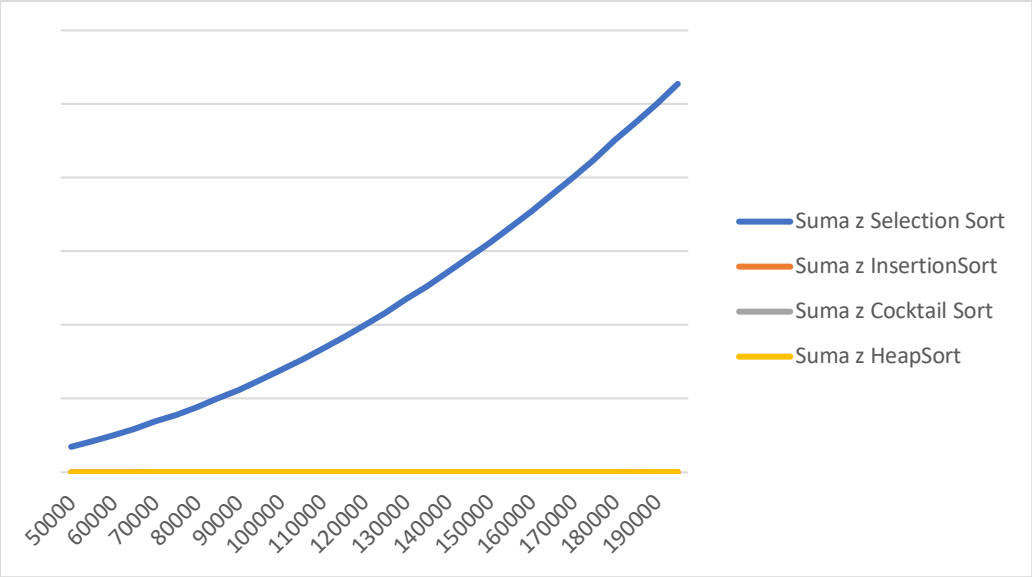
Część II – interakcja zbiorów liczbowych dla różnych typów algorytmów



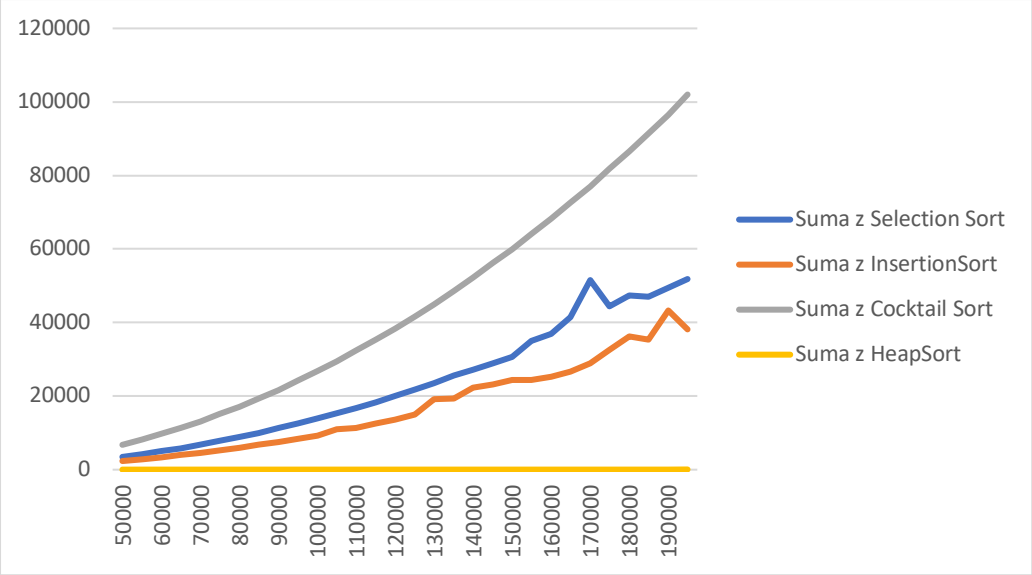
Tablica rosnąca to posortowana już tablica, która jest prawie zawsze przypadkiem optymistycznym.



Tablica malejąca jest najbardziej wymagająca. Na tym wykresie widać dużą rozbieżność w czasie.

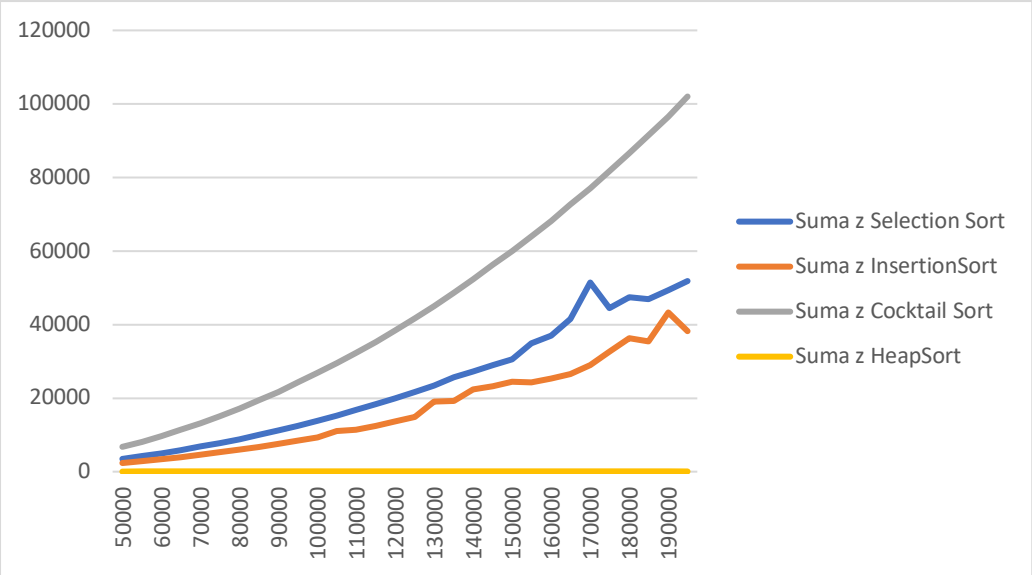


Tablice stałą można nazwać tak samo jak tablice rosnącą tablicą posortowaną



Tablica losowa czyli najbardziej nieprzewidywalny przypadek dla algorytmów sortujących.

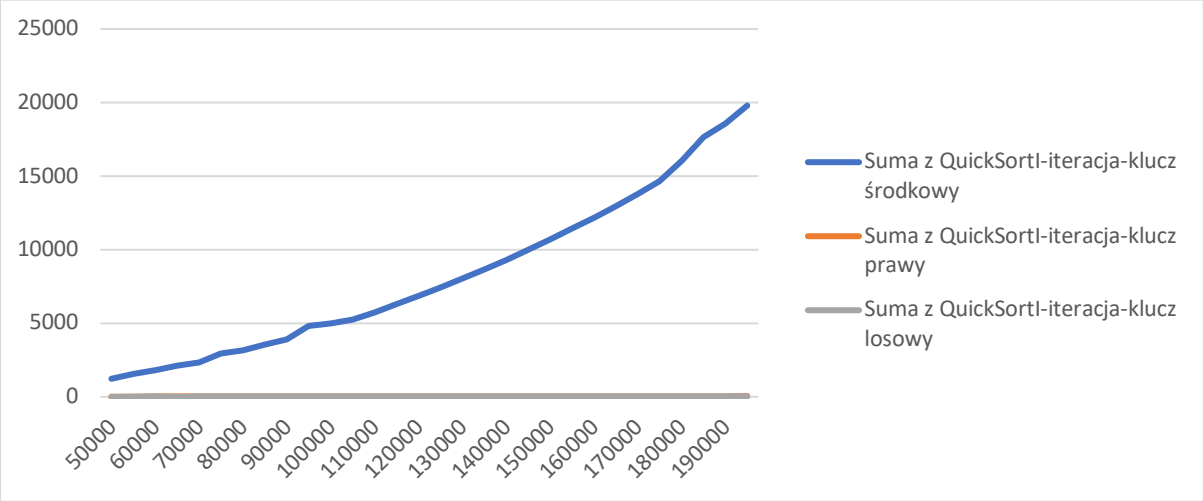
HeapSort radzi sobie z tym typem najlepiej



W tablicy V-kształtna podobnie jak w poprzednich przypadkach HeapSort wychodzi dobrze.

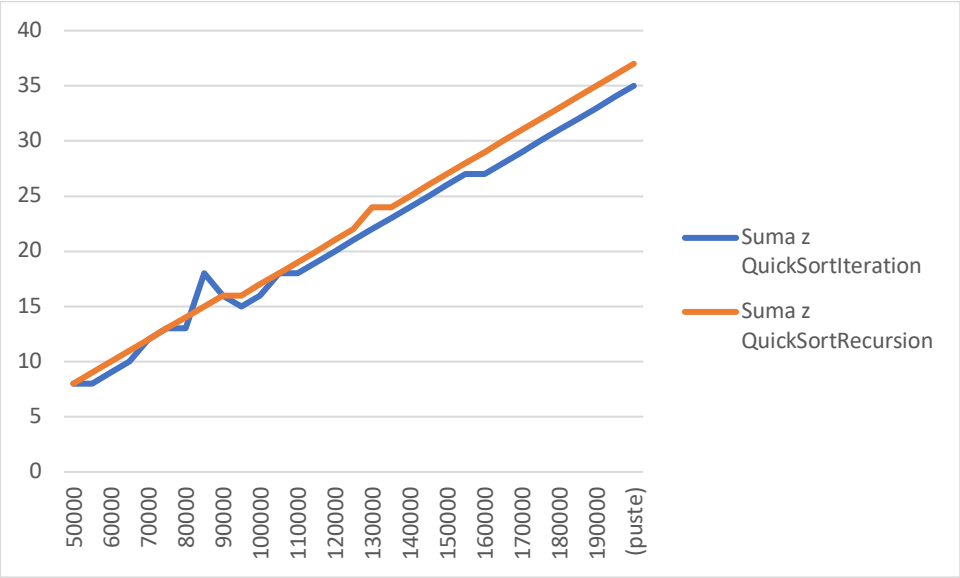
Część III – badanie QuickSort

Badanie nad QuickSort zaczęliśmy od porównania dwóch podstawowych wersji tego algorytmu czyli iteracyjnej i rekursywnej.



Jak widać czasy sortowania w obydwu wersjach są praktycznie takie same. Dzięki temu obie te metody są przyzwoite i wybór powinien paść na wygodniejszy sposób.

Następnie zostało porównanie różnych kluczy podziału. Wybraliśmy algorytm rekurencyjny QuickSort, wyposażony w trzy rodzaje kluczy: środkowy, prawy oraz lewy.



Zaskakującą rzeczą jest to że klucz generowany losowo został on najbardziej wydajny w tej konfiguracji.

Wnioski

Od początku programowania powstają to nowsze algorytmy sortujące. Dzięki temu możemy dopasować do swoich potrzeb oraz do programu różne algorytmy sortujące oraz dzięki temu projektowi mogliśmy się przekonać który algorytm jest najlepszy w danej sytuacji.