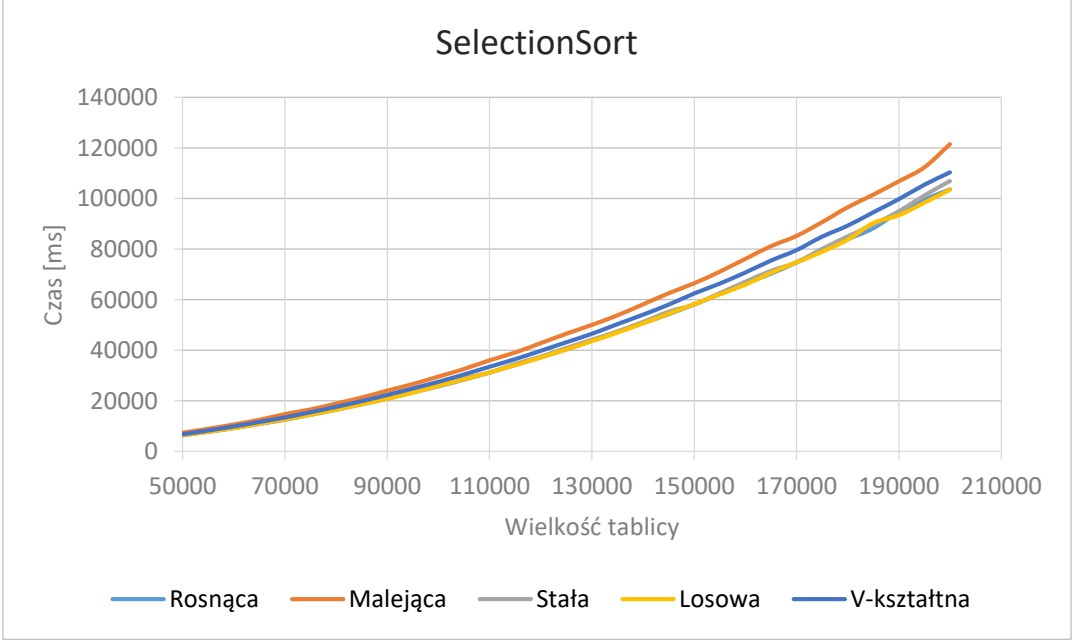
Projekt 3 – Sortowanie

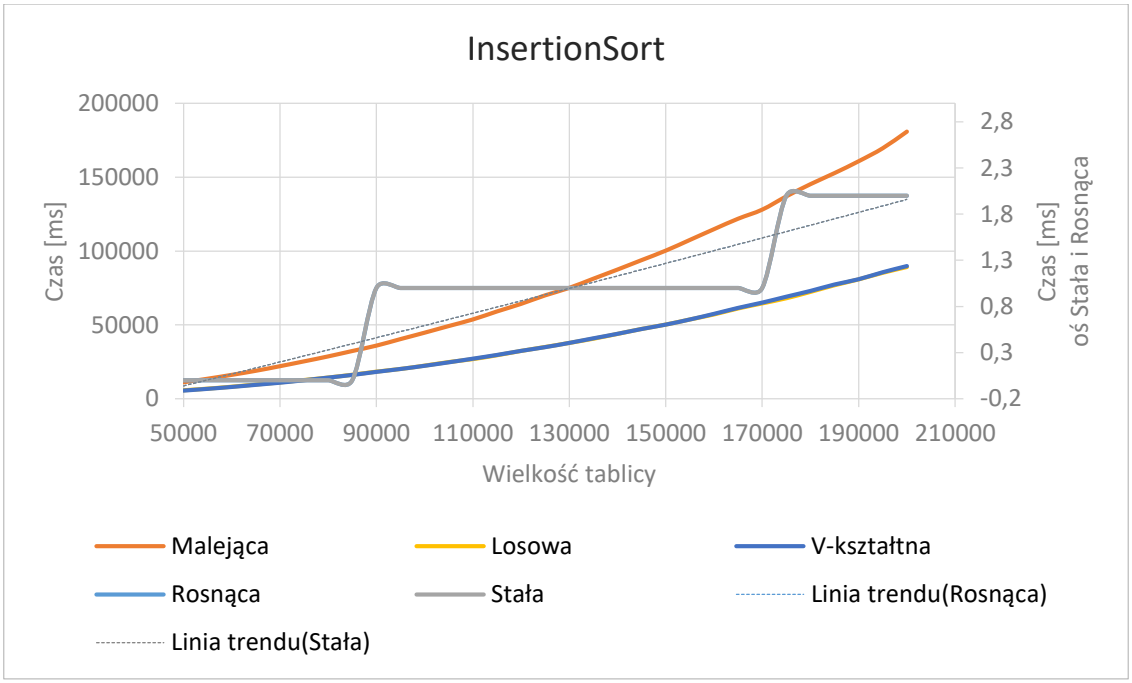
Hubert Nowak & Wiktor Szczeszek

Projekt składa się z trzech części, a każda z nich poświęcona jest analizie algorytmów sortujących tablice liczbowe. Za bazę do badań należało stworzyć tablice: rosnącą, malejącą, stałą, losową, V-kształtną, A-kształtną o wielkości od 50 000 do 200 000 liczb, które następnie podlegały sortowaniu przez algorytmy: SelectionSort, InsertialSort, CoctailSort, HeapSort oraz QuickSort. W ramach projektu mierzony był czas wykonywanych sortowań, a każde z nich było testowane w 31 punktach pomiarowych (tablice większe co 5 000), dodatkowo wszystkie wyniki zostały uśrednione w dziesięciu próbach. Do tego przedsięwzięcia wykorzystaliśmy IDE Microsoft Visual Studio Community 2019 w wersji 16.6.0, .NET framework 4.8.03761, język programistyczny C# oraz serwis hostingowy GitHub. Sprzęt dokonujący pomiarów to (AMD Athlon II X4 651, NVIDIA GeForce GTS 450, 8 GB RAM).

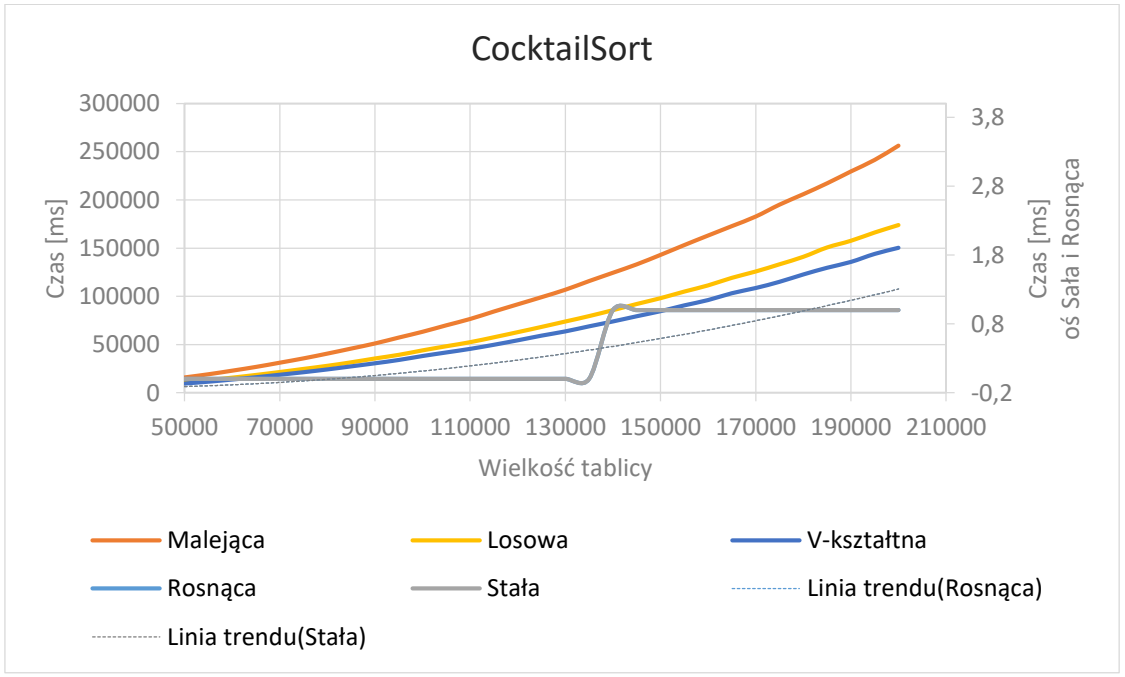
Część I - działanie algorytmów dla różnych typów zbiorów liczbowych



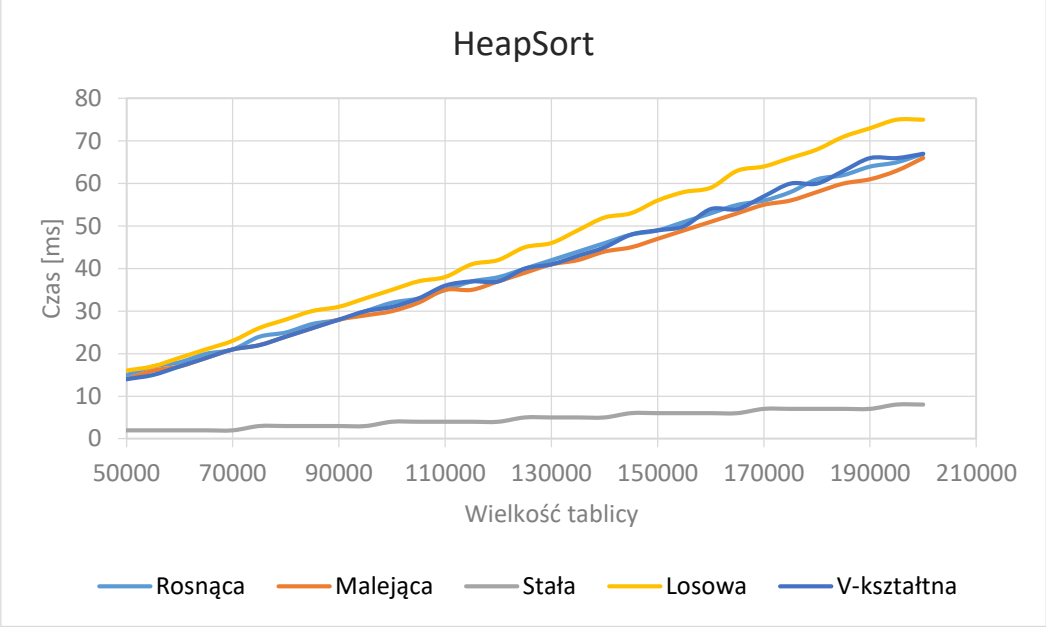
Algorytm SelectionSort jest jednym z najwolniejszych w zestawieniu tego projektu, a wynika to ze stałej złożoności $O(n^2)$. Dla każdego przypadku czas działania jest bardzo zbliżony, przez co niektóre krzywe na wykresie nakładają się na siebie.



Algorytm InsertionSort nie jest już tak jednolity jak poprzednik i potrafi bardzo szybko zakończyć proces sortowania dla tablic już posortowanych, czyli w tym przypadku dla tablicy rosnącej oraz stałej. W najgorszym przypadku, którym jest tablica malejąca czas wykonania sortowania wzrasta n^2, gdzie n to długość sortowanego zbioru liczbowego. Natomiast dla tablicy losowej i V-kształtnej czas ten jest bardzo podobny do siebie, jednak wciąż zbliżony do notacji $O(n^2)$.

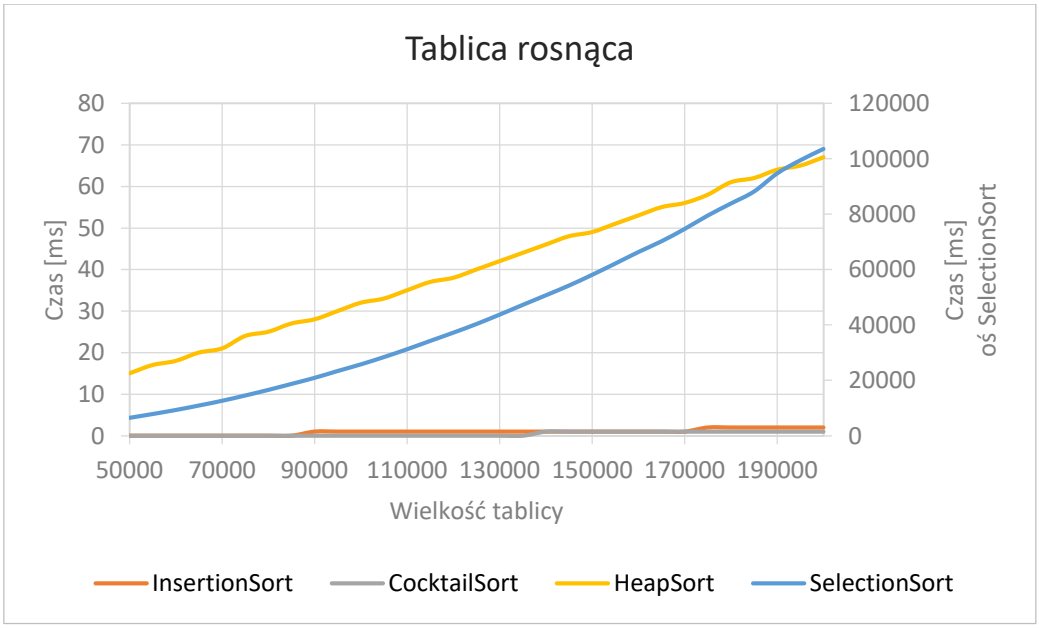


Algorytm CocktailSort ma podobną złożoność do algorytmu InsertionSort, natomiast jego czasy sortowania są stosunkowo dłuższe. W tym przypadku również tablica rosnąca i stała są najlepszym przypadkiem, w przeciwieństwie do tablicy malejącej.

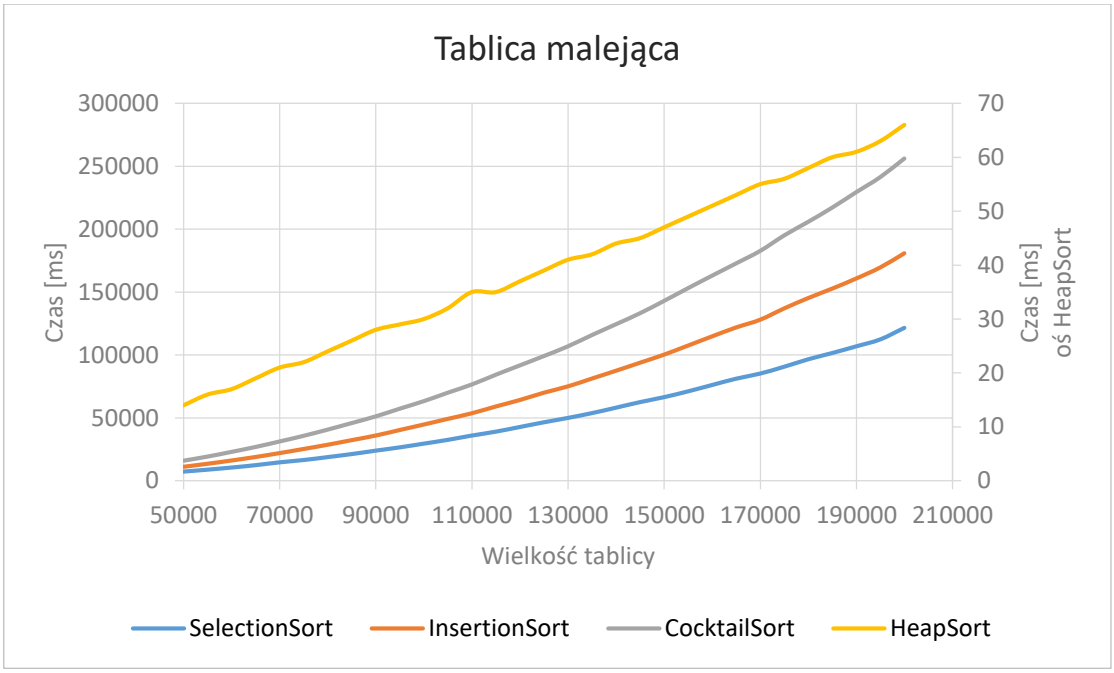


Algorytm HeapSort jest równie specyficzny co SelectionSort, natomiast w przeciwieństwie do niego jego złożoność algorytmiczna wynosi O(n log n). Dla tego typu sortowania nie ma większego znaczenia jaki typ danych wejściowych obsługuje. Jedynym wyróżniającym się przypadkiem jest przypadek optymistyczny występujący tylko dla tablicy stałej. Sprawia to, że algorytm ten jest dotychczas uśredniając najszybszy oraz bardzo uniwersalny.

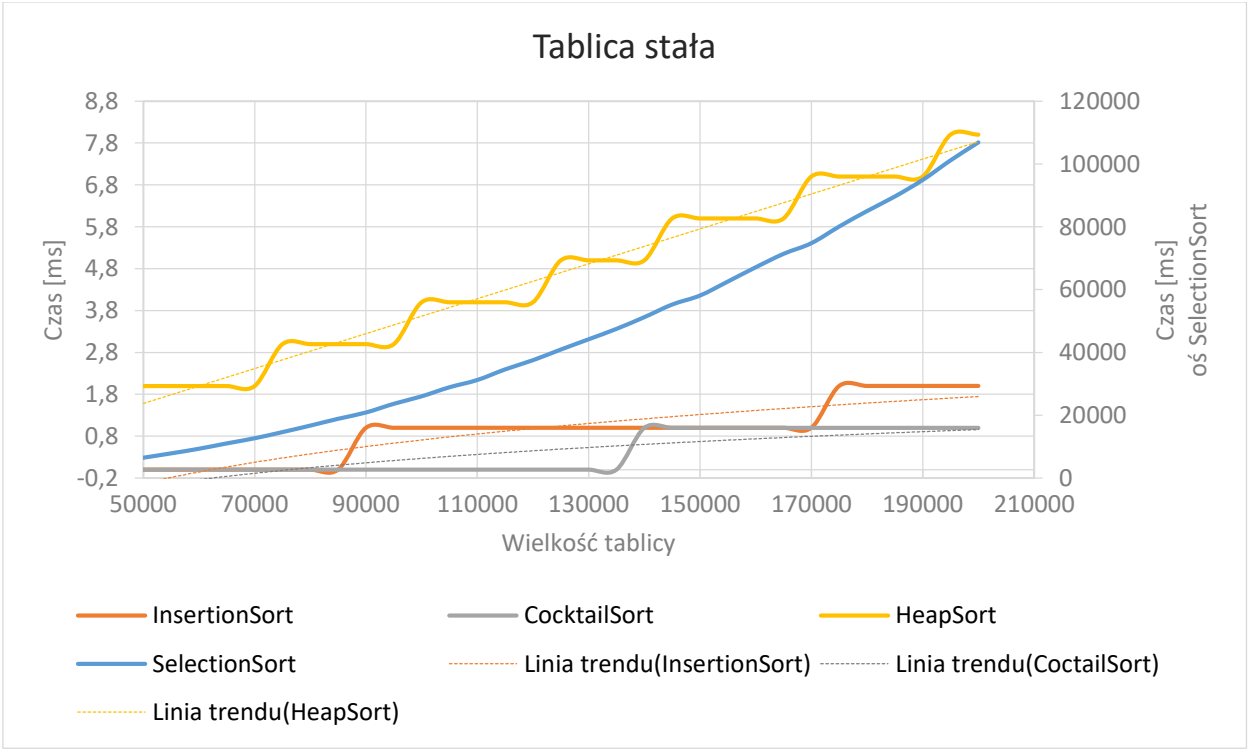
Część II – interakcja zbiorów liczbowych dla różnych typów algorytmów



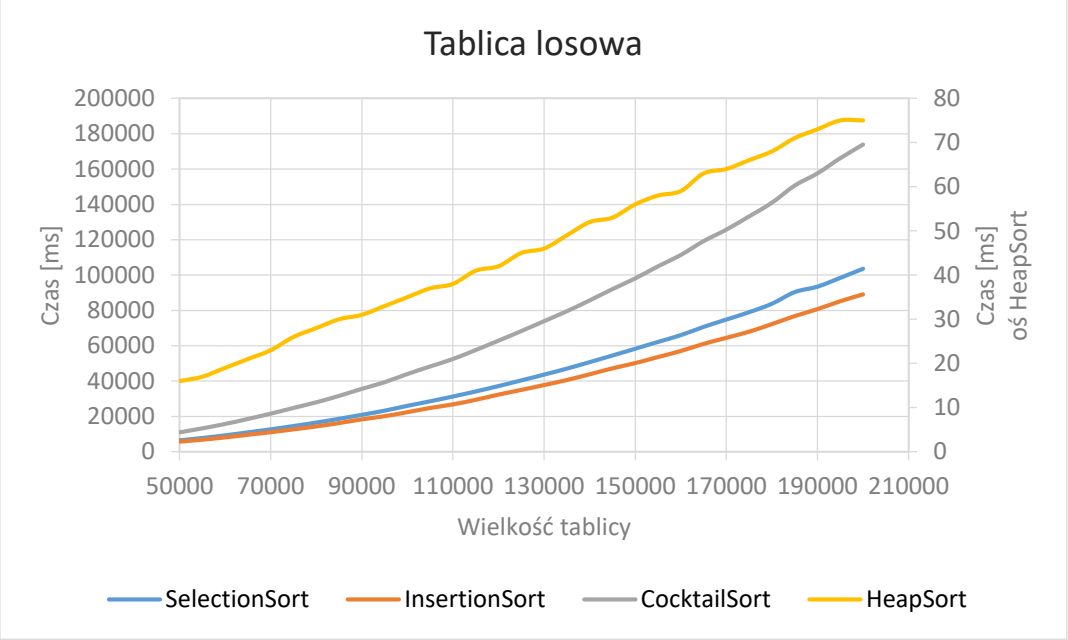
Tablica rosnąca to posortowana już tablica, która jest prawie zawsze przypadkiem optymistycznym dla algorytmów sortujących. Jak widać na wykresie najsprawniejszy jest InsertionSort oraz CocktailSort, a sortowanie w stylu HeapSort odbiega od reszty, jednak można uznać że to dobry wynik, tym bardziej biorąc pod uwagę jak duże są to zbiory liczbowe. Najgorzej wypada tutaj SelectionSort.



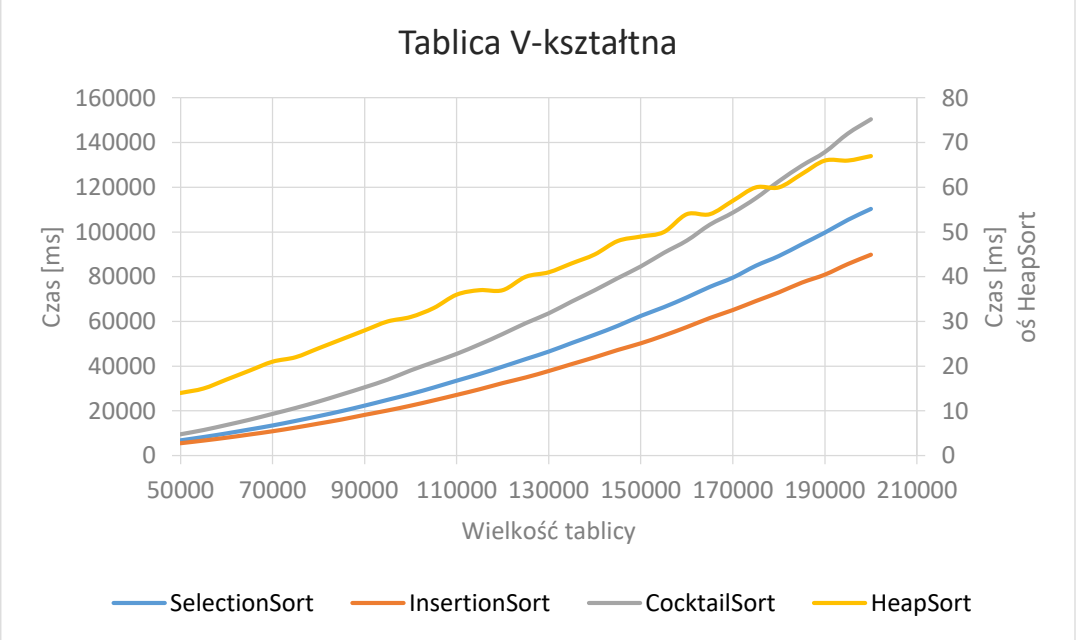
Tablica malejąca jest najbardziej wymagająca, ponieważ liczby w zbiorze są posortowane odwrotnie. Rozbieżność czasów jest dość spora, ale z pewnością można wskazać najszybszy algorytm HeapSort, natomiast najgorzej sobie poradził CocktailSort.



Tablica stała to podobieństwo tablicy rosnącej, gdyż można ją nazwać posortowaną. W tym przypadku CocktailSort oraz InsertionSort radzą sobie najlepiej, w przeciwieństwie do SelectionSort. HeapSort natomiast plasuje się trochę wyżej na wykresie, jednak wciąż czas sortowania jest bardzo krótki przedstawiony liniową krzywą trendu.



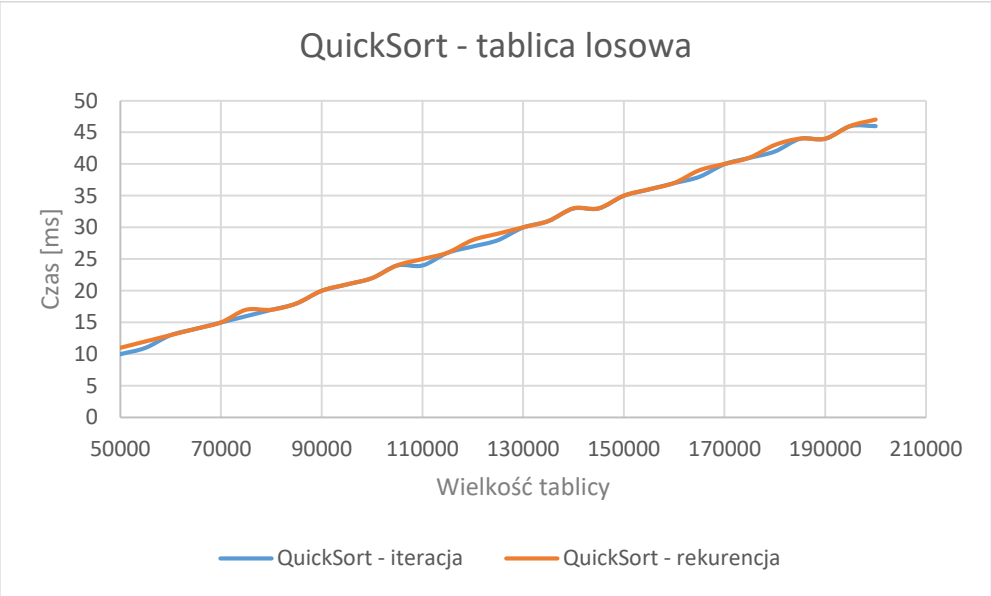
Tablica losowa czyli najbardziej naturalny i nieprzewidywalny przypadek dla algorytmów sortujących. HeapSort wskazany osią pomocniczą radzi sobie z tym typem danych wejściowych najlepiej i deklasuje resztę metod sortujących. Kolejnym zaskoczeniem jest CocktailSort, który radzi sobie gorzej nawet od SelectionSort i zostaje nominowany przypadkiem pesymistycznym w tym zestawieniu.



Tablica V-kształtna to specyficzne rozstawienie wartości liczbowych, które początkowo maleją, a końcowo wzrastają. Podobnie jak dla poprzedniej tablicy algorytm HeapSort pokazał się z dobrej strony. Na wykresie wyraźnie można zauważyć wyróżniającą się krzywą tego algorytmu, która jest w notacji O(n log n) (dla każdego przypadku, nawet pesymistycznego tego algorytmu). CocktailSort po raz kolejny plasuje się na ostatnim miejscu.

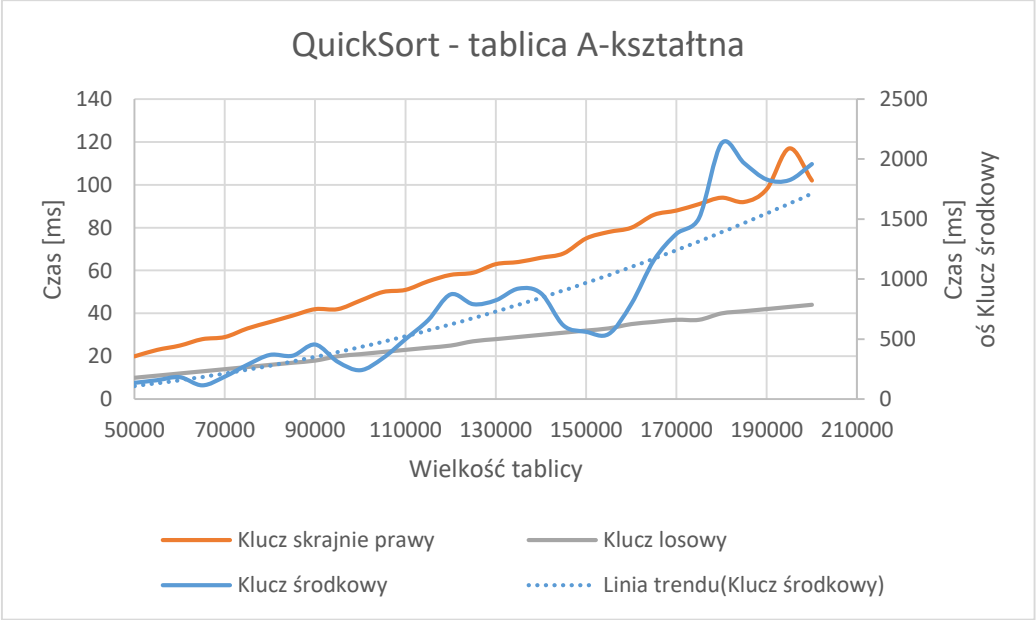
Część III – badanie QuickSort

Badanie nad QuickSortem należało zacząć od porównania dwóch podstawowych wersji zapisu tego algorytmu, czyli wersji iteracyjnej oraz rekursywnej. Obie te implementacje sortowały tablice liczb losowych ich klucz podziału (pivot) wyznaczał środek zbioru liczbowego.



Jak widać czasy sortowania w obydwu wersjach są praktycznie takie same. Z teoretycznego punktu widzenia wersja rekursywna powinna być wolniejsza ze względu na samą rekurencje, która jest zapisywana i przywracana ze środowiska. Natomiast obie są sprawne i wybór powinien paść na wygodniejszy sposób implementacji.

Kolejnym eksperymentem jest porównanie różnych kluczy podziału (pivot'ów). Za podstawę badania wybrany został skonfigurowany algorytm rekurencyjny QuickSort, wyposażony w trzy rodzaje kluczy: klucz środkowy, klucz skrajnie prawy, klucz losowy. Tablica A-kształtna, czyli przeciwieństwo tablicy V-kształtnej, stanowiła bazę sortowania zbiorów liczbowych.



Dane na wykresie są bardzo spontaniczne w szczególności dla wersji klucza środkowego. Zaskakującym przeciwieństwem jest klucz generowany losowo i zarazem został on najbardziej wydajny w tej konfiguracji. Klucze podziału dla algorytmu QuickSort są bardzo ważnym elementem, ponieważ złe wyznaczenie klucza dla odpowiednich zbiorów liczbowych mogłoby nawet przepełnić stos kompilatora, w szczególności dla wersji rekurencyjnej. Dlatego najbardziej elastyczny wydaje się klucz losowy oczywiście w podstawowej wersji konfiguracyjnej tego algorytmu.

Podsumowanie

Przez wiele lat powstawały różnorodne algorytmy sortujące i wciąż powstają. Wybór sposobu sortowania to ważny element niejednego programu, wpływający na jego wydajność. Dzięki I oraz II części tego projektu można wyznaczyć z łatwością nieskomplikowane algorytmy do odpowiednich zadań. Jednak QuickSort jest w tym zestawieniu najszybszym oraz jednocześnie najbardziej rozwijanym przez społeczność algorytmem sortującym zbiory liczbowe i warto zaznajomić się z jego działaniem.