Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

Projekt polega na ukazaniu różnic w działaniu różnych algorytmów sortujących, przy różnych tabelach o szczególnej specyfice. Celem analizy jest zrozumienie kiedy warto zastosować poszczególne algorytmy i zrozumieć ich silne oraz słabe strony.

Używane narzędzia:

- Microsoft Visual Studio 2019
- Google Sheets
- Microsoft Excel
- Google Docs
- Notatnik
- Procesor Intel Core i5-6400 2.70 GHz

1. Porównania prędkości sortowania specyficznych tablic przez różne algorytmy

Pierwszą częścią naszego badania (a drugą na poleceniu) jest porównanie w działaniu czterech algorytmów:

- Insertion Sort
- Selection Sort
- Heap Sort
- Cocktail Sort

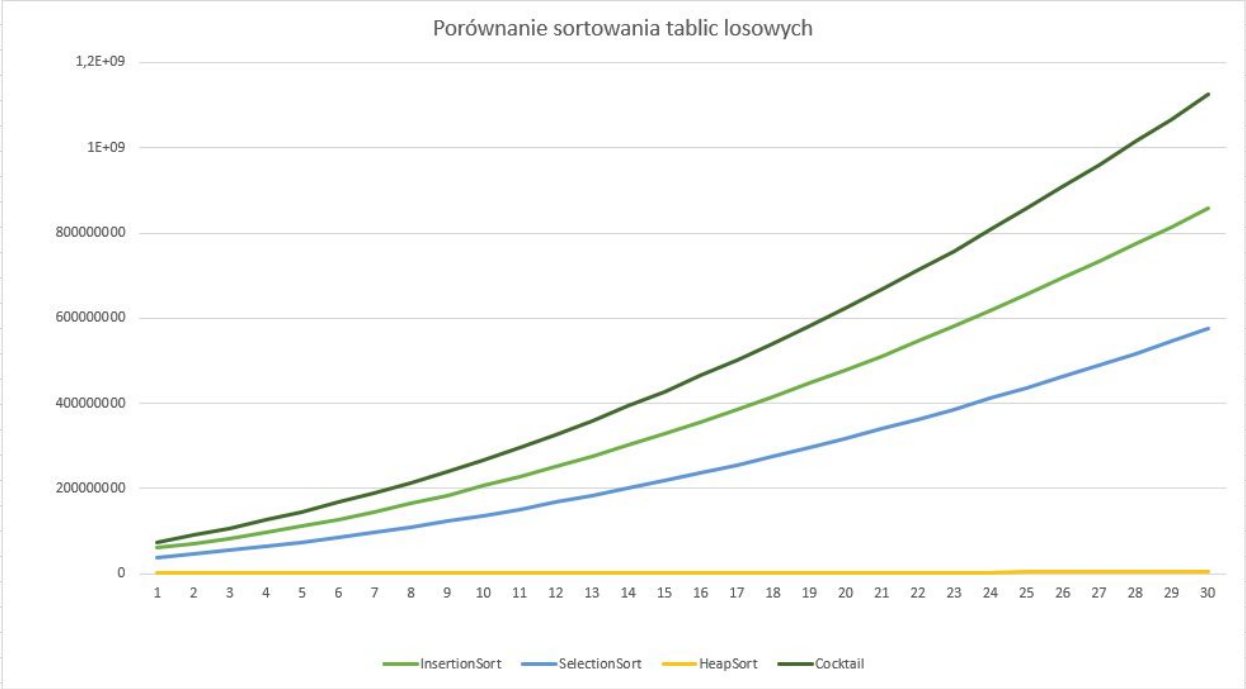
Badamy je pod względem czasu w tickach poprzez zestawienie ich wyników sortowania pięciu różnych typów tablic:

- Tablica losowa
- Tablica V-kształtna
- Tablica rosnąca
- Tablica malejąca
- Tablica stała

Każdy z algorytmów posiada swoją specyfikę działania, gdzie radzi sobie lepiej lub gorzej z określonym typem tablic. Poniżej zestawiamy wykresy dla poszczególnych tablic z wynikami, wraz z analizą.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

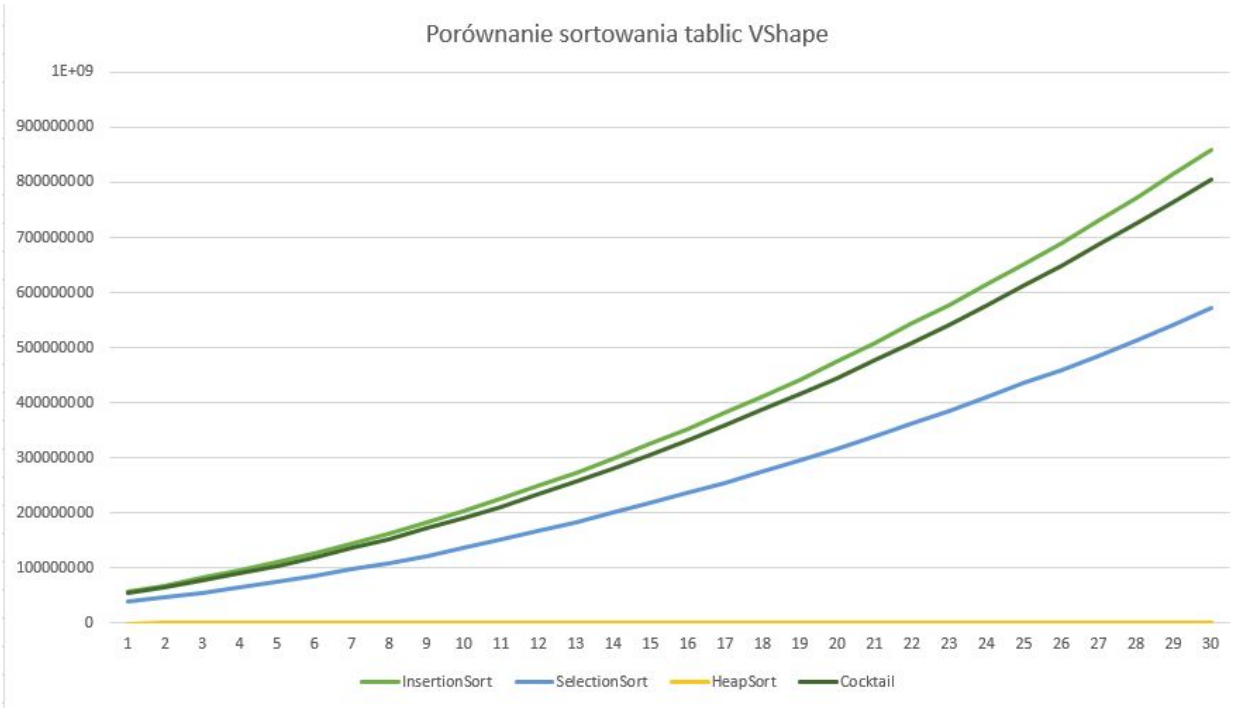


Wykres 1. Algorytmy sortujące tablicę liczb losowych, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

W powyższym przykładzie uszeregowania tablic losowych, najszybszym z testowanych algorytmów jest zdecydowanie Heap Sort. Brak większego zaskoczenia. Insertion Sort zajmuje konsekwentnie dużo czasu.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

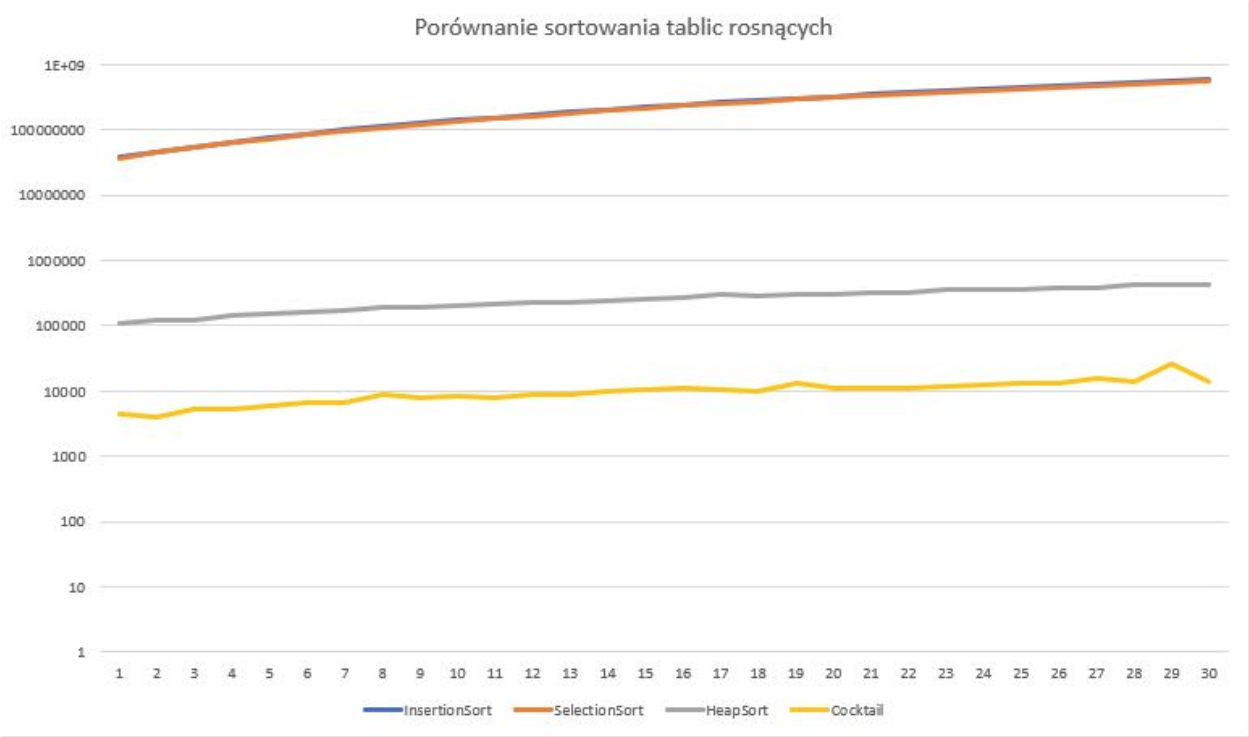


Wykres 2. Algorytmy sortujące tablicę V-kształtną, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Ponownie najlepiej sprawuje się Heap Sort. Insertion Sort i Selection Sort radzą sobie podobnie słabo.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

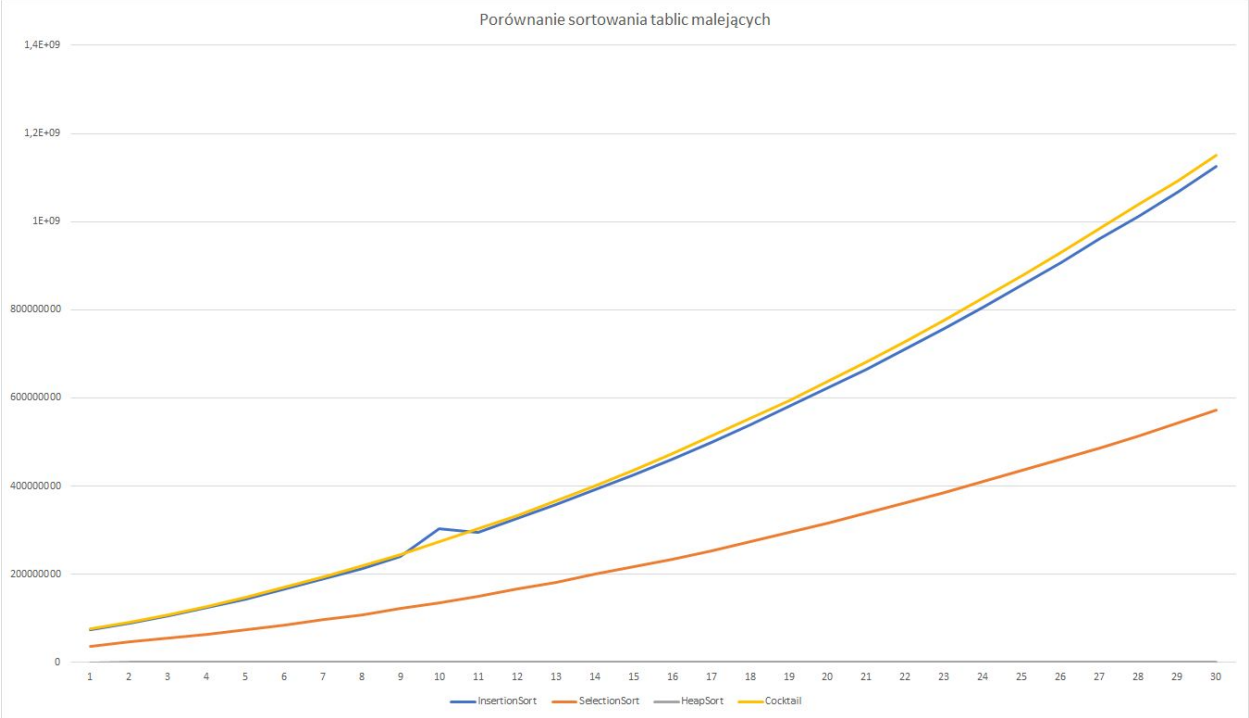


Wykres 3. Algorytmy sortujące tablicę liczb rosnących, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Tutaj zaczynają się pokrycia w czasach - Insertion sort oraz Selection Sort mają niemal jednakowe czasy działania. Tym razem to Cocktail Sort poradził sobie najlepiej.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

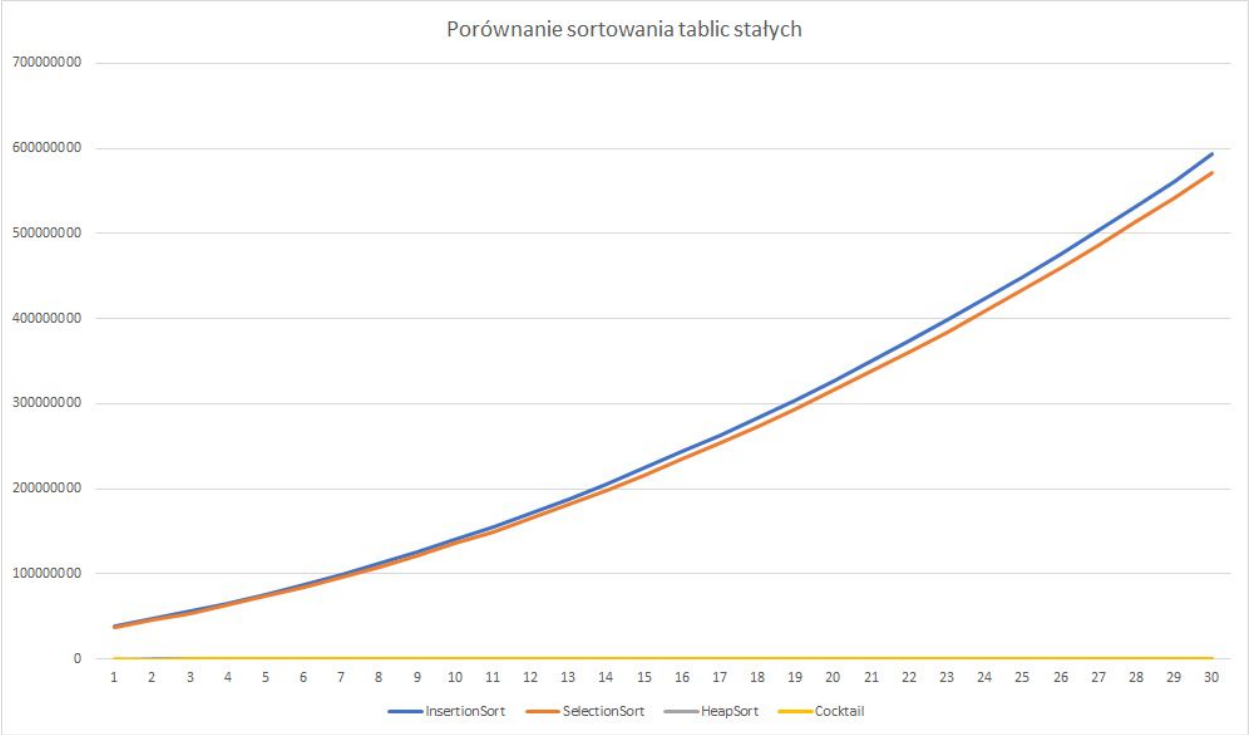


Wykres 4. Algorytmy sortujące tablicę liczb malejących, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Zwycięzcą w uporządkowaniu tablic malejących jest Heap Sort, niemal identycznie radzą sobie Insertion oraz Cocktail Sort, średnio Selection Sort.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak



Wykres 5. Algorytmy sortujące tablicę liczb stałych, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Ponownie mamy nałożenia - Insertion Sort oraz Selection Sort radzą sobie stosunkowo słabo, porównując z parą Heap Sort oraz Cocktail Sort - gdzie ten drugi ma około cztery razy lepsze wyniki od “rówieśnika”.

Wnioski: Do sortowania tablic malejących, losowych i V-kształtnych zdecydowanie należy używać Heap Sorta, który ma mniejszą złożoność obliczeniową i osiąga najlepsze wyniki.

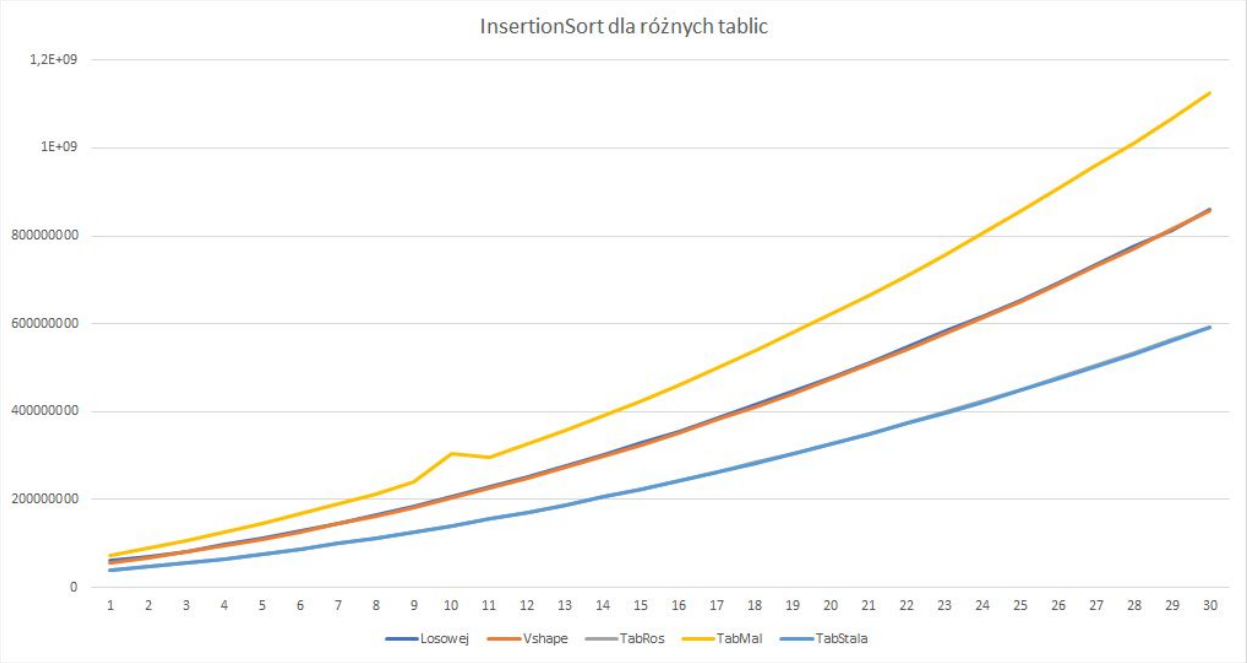
Dominik Karmoliński
Piotr Karolak

2. Porównanie działania czterech algorytmów dla każdego przypadku

W drugiej części naszego projektu (w poleceniu cz. 1) należy przedstawić działania czterech algorytmów:

- Insertion Sort
- Selection Sort
- Heap Sort
- Cocktail Sort

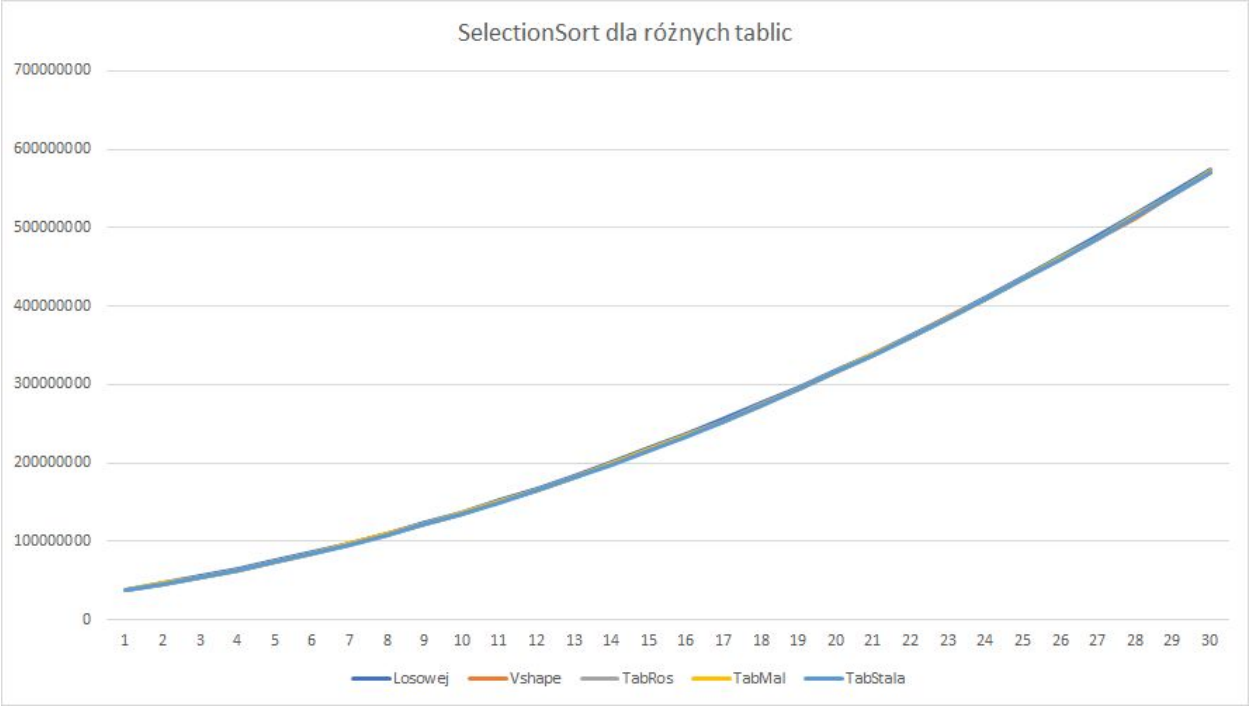
Każdy z nich radzi sobie lepiej lub gorzej przy sortowaniu poszczególnych tablic, co zademonstrujemy poniżej.



Wykres 6. Insertion Sort sortujący różne tablice, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Insertion Sort radzi sobie najlepiej z tablicami stałymi i rosnącymi, nieznacząco lepiej z tymi pierwszymi. Najgorzej radzi sobie z malejącymi tablicami. V-shape i losowa mają niemal identyczne czasy i plasują się pośrodku.

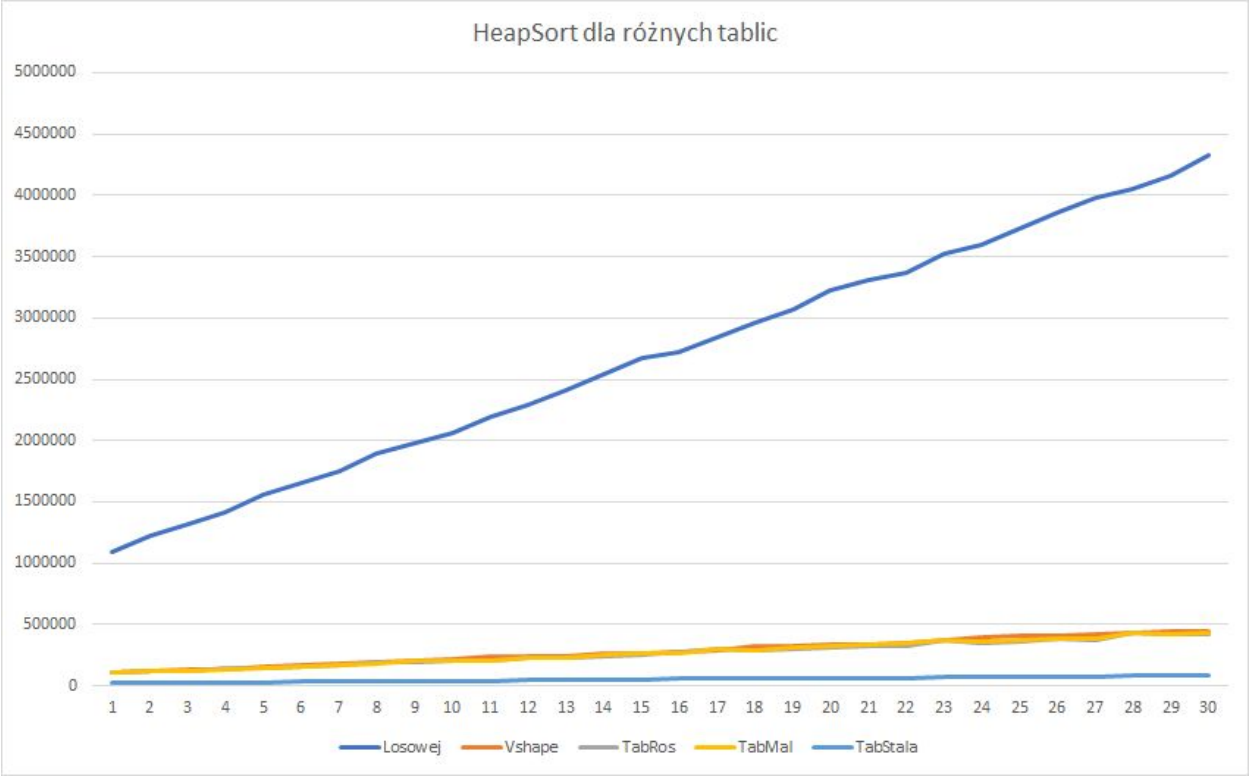
Dominik Karmoliński
Piotr Karolak



Wykres 7. Selection Sort sortujący różne tablice, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Selection sort w swojej specyfice działania nie wykazuje żadnych znaczących zmian przy porządkowaniu jakichkolwiek tablic. Różnice czasów wynoszą poniżej 1% w każdym przypadku, z praktycznie pomijalnym błędem odczytu.

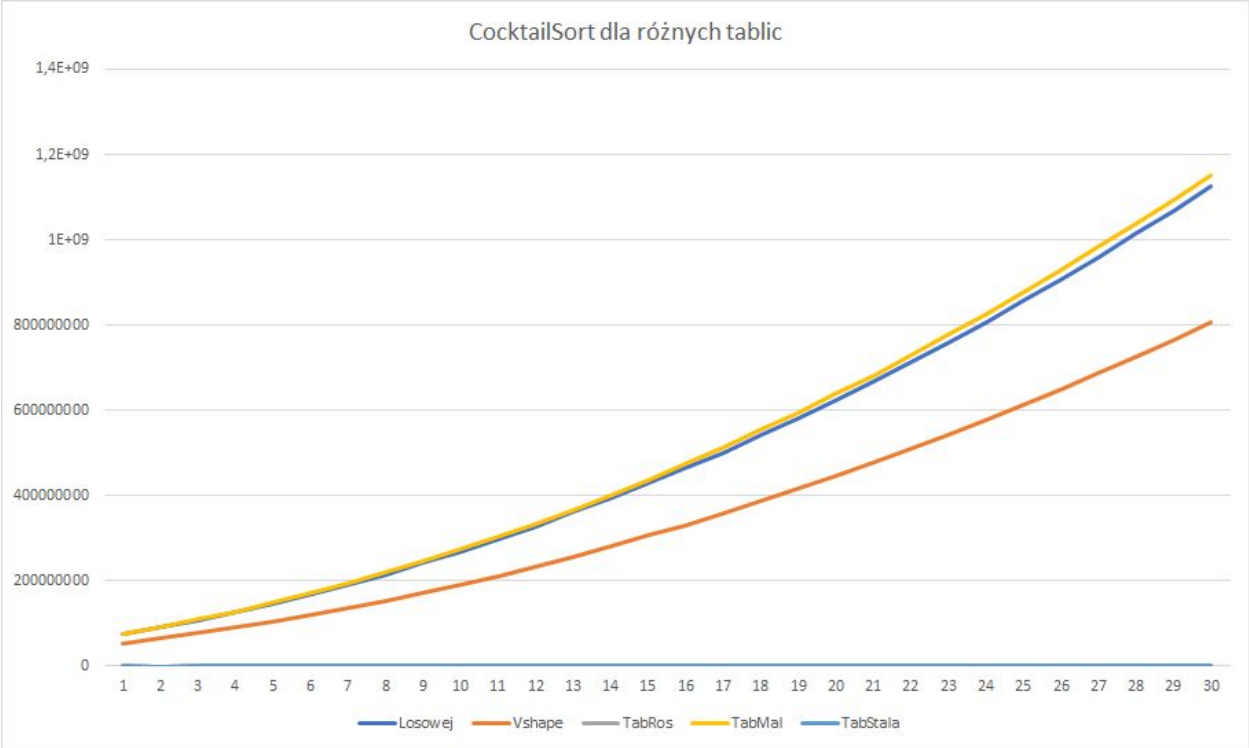
Dominik Karmoliński
Piotr Karolak



Wykres 8. Heap Sort sortujący różne tablice, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Heap sort radzi sobie bardzo dobrze w niemal każdym przypadku, a najlepiej przy tablicach stałych. Najgorzej jednak radzi sobie z prawdziwie losowymi tablicami, gdzie przebijają go liczne inne algorytmy. Wszelkie inne typy tablic (V, rosnąca, malejąca) mają nieznaczące odchyły.

Dominik Karmoliński
Piotr Karolak



Wykres 9. Cocktail Sort sortujący różne tablice, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

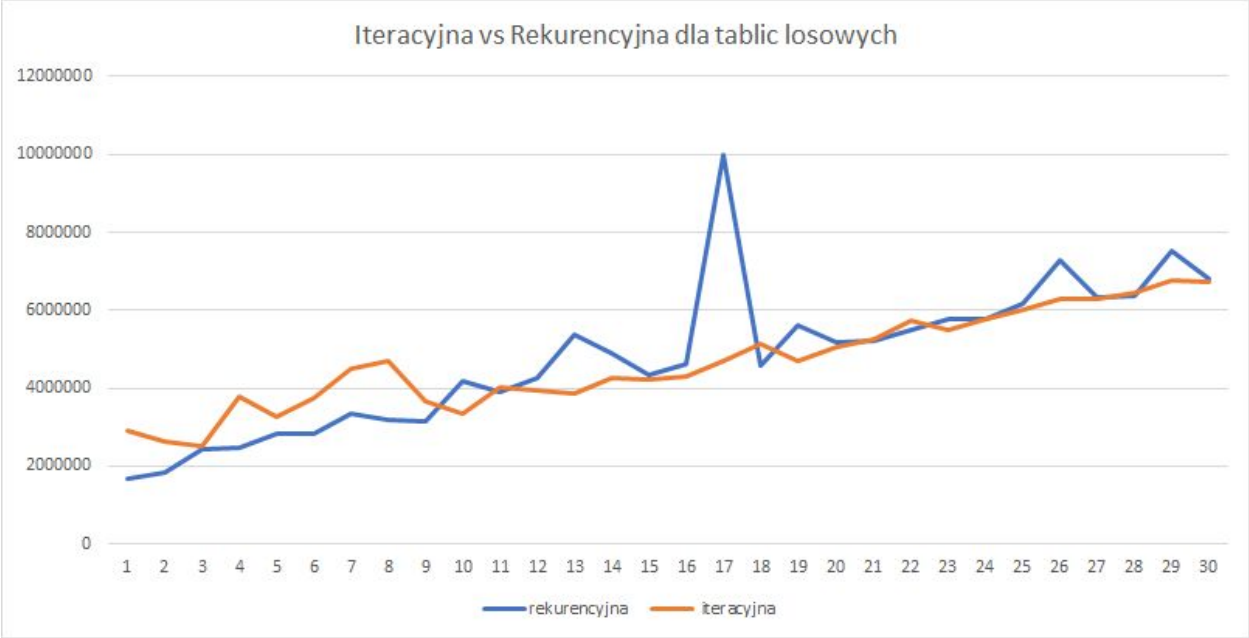
Cocktail sort radzi sobie bardzo dobrze z tablicami rosnącymi i stałymi. Stosunkowo najlepiej z reszty tablic wygląda uszeregowanie V-kształtnej tablicy, a dla reszty (losowa, malejąca) radzi sobie najgorzej - z czego wynik tablicy malejącej jest najsłabszy.

Dominik Karmoliński
Piotr Karolak

3. Analiza algorytmu Quicksort

Do trzeciej części zadania należy porównanie iteracyjnej oraz rekursywnej wersji QuickSorta dla tablic losowych oraz porównanie działania w zależności od wyboru klucza podziału przy tablicach A-kształtnych, czyli pesymistycznym przypadku dla algorytmu.

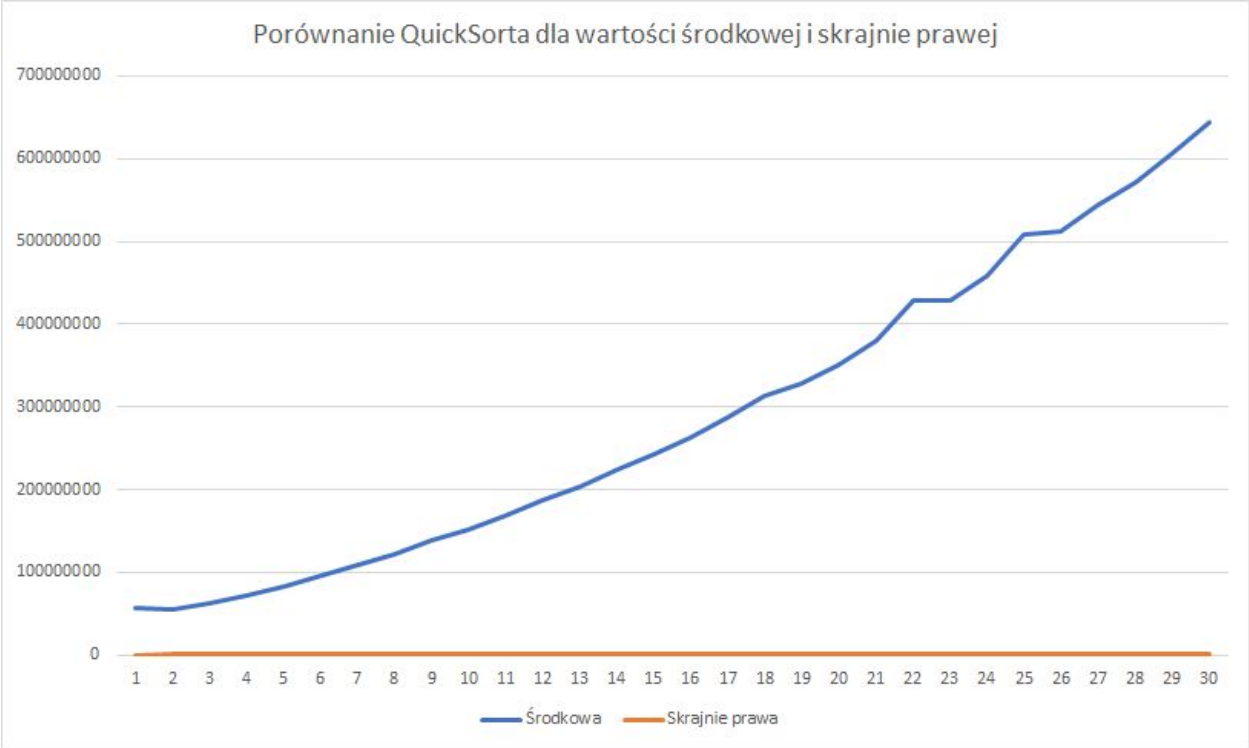
Wykresy oraz wnioski prezentujemy poniżej.



Wykres 10. QuickSort sortujący tablice losowe, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Z wykresu nie wynika, by którakolwiek wersja była znacząco lepsza niż inne, jednak spore wahania na naszym wykresie mogą zaburzać nasze wyniki. Czysto teoretycznie, tak przeciętny, jak i najgorszy przypadek dla obu wersji to odpowiednio O (n log n) oraz O (n^2), co potwierdzają powyższe wyniki.

Dominik Karmoliński
Piotr Karolak



Wykres 11. QuickSort sortujący wedle kluczy podziału, 50k-200k, skok 5k, 30 punktów pomiarowych. Czas w tickach procesora.

Na powyższym wykresie łatwo zauważyć, że dla środkowego klucza podziału wzrost czasu wykonania jest logarytmiczny. Skrajnie prawy klucz daje znacząco niższe wyniki, wskazujące na wzrost liniowy.

Różnica jest drastyczna, lecz są to spodziewane wyniki. Dobór złego klucza podziału może bardzo negatywnie wpłynąć na czas działania algorytmu.

Algorytmy - projekt trzeci

Dominik Karmoliński
Piotr Karolak

Podsumowanie

Nie ma algorytmów idealnych. Jest to czołowy i najbardziej kluczowy wniosek z całej analizy, gdyż interpretacja wyników jasno pokazuje, że każdy z omawianych algorytmów ma swoje “pięty achillesowe”, w których radzi sobie znacząco gorzej niż któryś z innych testowanych algorytmów.

Najgorszy możliwy przypadek A-kształtnej tablicy dla QuickSorta daje fatalne wyniki w porównaniu z innymi algorytmami. Dlatego też wybieranie zawsze jednego algorytmu “w ciemno” do sortowania tablic może być złym wyjściem dla każdego programisty liczącego się z ograniczonymi zasobami oraz czasem.

W realnym świecie, gdy skuteczna implementacja jest najistotniejsza dla działania naszego programu, wypracowano metodę analizowania tablicy do posortowania poprzez wybór paru punktów kontrolnych i porównanie ich, aby móc określić który z algorytmów z naszej listy należy użyć do konkretnie sprawdzanej tablicy. Gdy okaże się, że badany przypadek zdaje się być “przyjemny” dla naszego algorytmu sortującego pierwszej potrzeby, zostanie on zastosowany. Gdyby jednak okazało się, że próba wskazuje na pesymistyczny przypadek naszego algorytmu numer jeden, przechodzimy do “planu awaryjnego” i stosujemy któryś z innych algorytmów, które poradzą sobie znacząco lepiej w danym przypadku.

Gdyby nie stosowano takiego próbnego sprawdzania, niepotrzebnie wydłużanoby wykonywanie sortowania tablic do nieakceptowalnych z punktu widzenia wydajności czasów, gdzie zastosowanoby wykładniczo sortujące algorytmy zamiast dużo prostszych opcji oferowanych przez inny sorter.