Opracowanie algorytmów sortowania

Szymon Kaczmarek, nr indeksu 148056

1 Wprowadzenie

Niniejsze opracowanie ma na celu sprawdzenie efektywności czasowej algorytmów sortowania zaimplementowanych w C++ w zależności od rodzaju i ilości danych wejściowych. Opracowane algorytmy zaliczają się do trzech grup: naiwne, „dziel i zwyciężaj" oraz inne. Kody źródłowe oraz dane testowe znajdują się w repozytorium GitHub.

2 Algorytmy naiwne

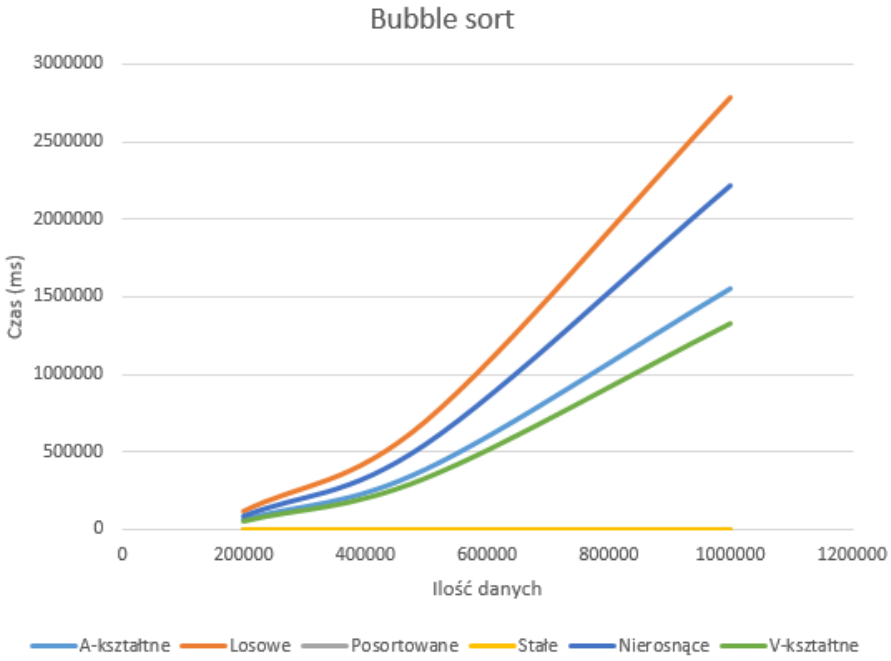
Algorytmy naiwne są prostymi w idei algorytmami, które zazwyczaj nie są bardzo szybkie. Ich zaletą jest prostota zrozumienia, a wadą czas potrzebny do przetworzenia dużej ilości danych.

2.1 Bubble sort

Bubble sort to algorytm polegający na porównywaniu sąsiadujących elementów w tablicy. Dopóki istnieją elementy sąsiadujące, które nie są uporządkowane, należy je zamienić miejscami i ponownie sprawdzić całą tablicę w poszukuiwaniu kolejnych takich par. Cechy tego algorytmu to:

- Duża złożoność obliczeniowa (średnio $O(n^2)$)
- Stabilny
- Działa *in situ*

Wykres zależności czasu sortowania od typu i ilości danych



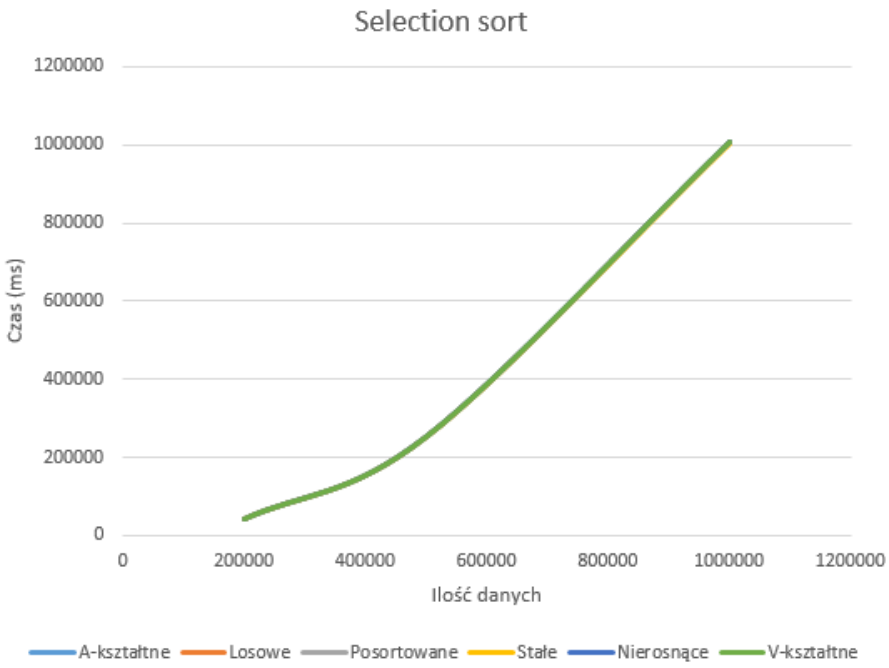
Obserwacje i wnioski
Jak widać czasy sortowania rosną bardzo szybko wraz ze wzrostem ilości danych. Już dla 500 000 elementów algorytm wykonuje się w okolicy 500 000 milisekund (około 8 minut). Jedynymi wyjątkami są dane już posortowane oraz stałe, dla których bubble sort przyjmuje złożoność liniową, nieporównywalnie szybszą od kwadratowej.

2.2 Selection sort

Selection sort również jest prostym w zrozumieniu algorytmen sortowania. Polega on na tym, że wyznaczamy najmniejszy element, po czym, wstawiamy go na odpowiednie miejsce. Następnie, szukamy następnego najmniejszego elementu, i wstawiamy go na kolejne miejsce. Powtażamy to, aż do posortowania tablicy. Do cech tego algorytmu należą:

- Duża złożoność obliczeniowa (średnio $O(n^2)$)
- Brak stabilności
- Działa *in situ*

Wykres zależności czasu sortowania od typu i ilości danych



Obserwacje i wnioski
Można zauważyć, że czasy sortowania nie różnią się od siebie ze względu na typ danych. Sybkość wykonywania, dla podanych danych, była pokroju kilku minut.

3 Algorytmy „dziel i zwyciężaj”

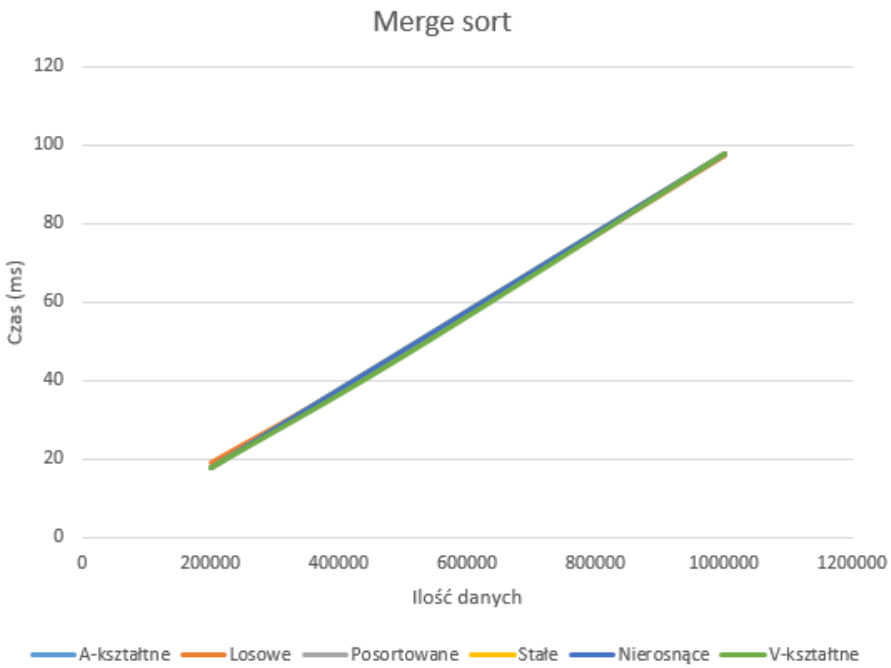
Nazwa „dziel i zwyciężaj” pochodzi od sposobu rozwiązywania problemu. Zamiast próbować rozwiązać cały problem od razu, dzielimy go na podproblemy, które są łatwiejsze do rozwiązania. Algorytmy z tej grupy są znacznie szybsze od poprzednich.

3.1 Merge sort

Merge sort jest stosunkowo prostym algorytmem do zrozumienia. Jego założenie to to, że jednoelementowy zbiór jest uporządkowany. Algorytm dzieli tablicę na dwie mniejsze części następnie robi to samo z otrzymanymi podtablicami, aż dojdzie do podtablic wielkości jeden. Następnie, algorytm scala ze sobą posortowane podtablice w większe, aż do uzyskania uporządkowanych danych. Jego cechy to:

- Niska złożoność obliczeniowa (średnio $O(nlog(n))$)
- Stabilność
- Funkcja scalająca potrzebuje dodatkowej tablicy

Wykres zależności czasu sortowania od typu i ilości danych



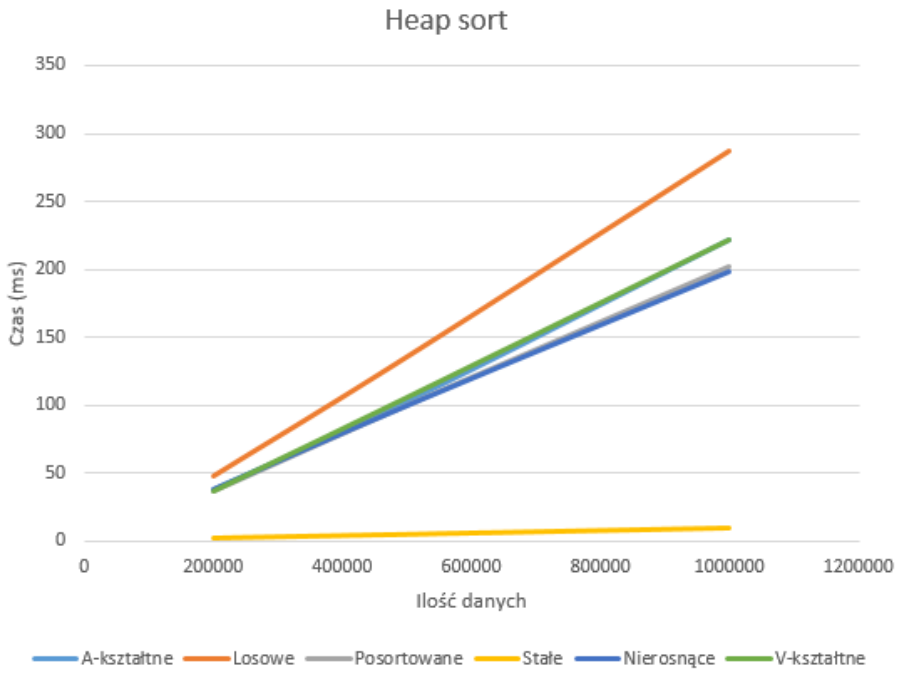
Obserwacje i wnioski
Można zauważyć ogromny spadek czasu sortowania względem poprzednich algorytmów, oraz brak wpływu typu danych na czas wykonywania.

3.2 Heap sort

Działanie heap sorta opiera się na strukturze kopca, czyli binarnego drzewa, w którym wartości potomków rodzica są mniejsze od niego. W takiej strukturze danych korzeń drzewa zawsze jest wartością największą w kopcu. Algorytm buduje kopiec z podanych danych, a następnie, usuwa jego wierzchołek zamieniając go z wartością z końca, po czym, odtwarza porządek kopcowy. Operacje te są powtarzane aż do posortowania danych. Cechy tego sortowania to:

- Niska złożoność obliczeniowa (średnio $O(nlog(n))$)
- Brak stabilności
- Działa *in situ*

Wykres zależności czasu sortowania od typu i ilości danych



Obserwacje i wnioski
Czas wykonywania sortowania jest o wiele mniejszszy od algorytmów naiwnych. Można zauważyć również znaczną rozbieżność czasów względem typu danych. Prawdopodobnie jest ona spowodowana algorytmem tworzenia kopca, który nieporównywalnie szybciej działa dla danych zawierających elementy o stałej wartości.

4 Inne algorytmy sortowania

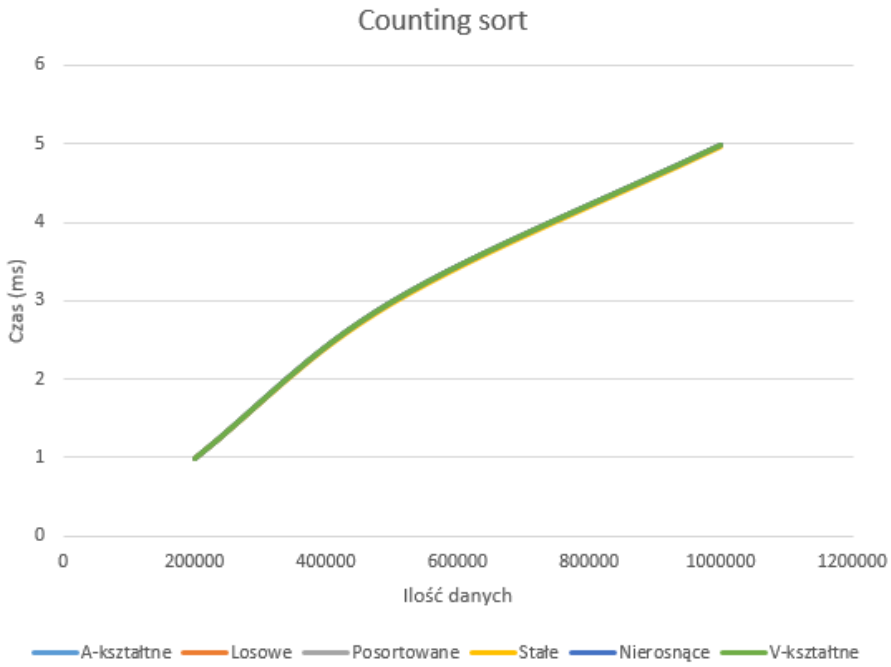
Do tej grupy algorytmów zaliczają się te, które do sortowania wykożystują pewne sprytne sztuczki.

4.1 Counting sort

Counting sort to algorytm, który nie sortuje elementów poprzez zamianę ich miejscami, lecz przez zliczenie ilości wystąpień danych wartości a następnie wypisanie ich we wcześniej ustalonej kolejności. Jego cechy to:

- Bardzo niska złożoność obliczeniowa ($O(n+k)$)
- Stabilność
- Musi znać zakres danych przed sortowaniem
- Potrzebuje dodatkowej tablicy o wielkości zakresu danych

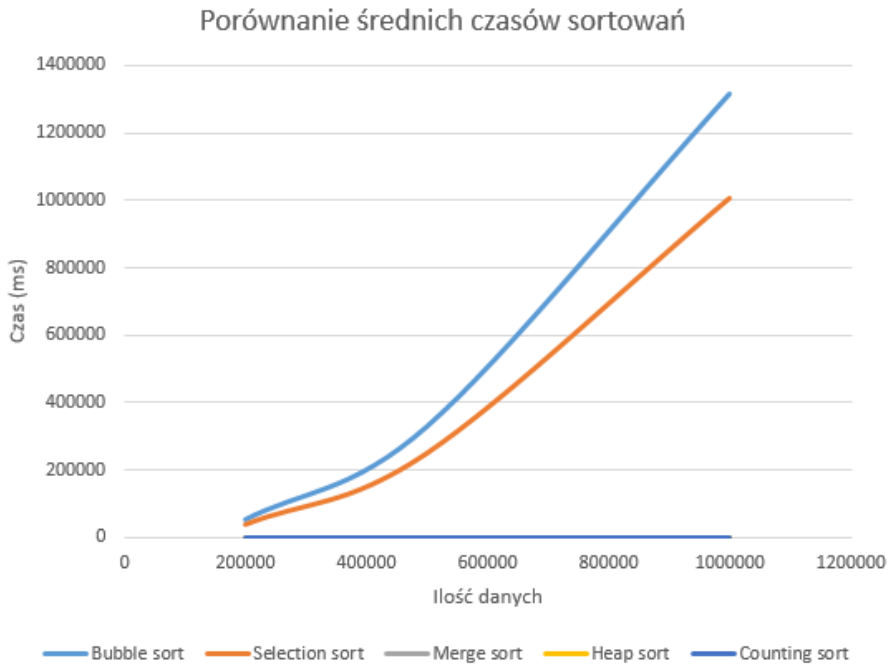
Wykres zależności czasu sortowania od typu i ilości danych



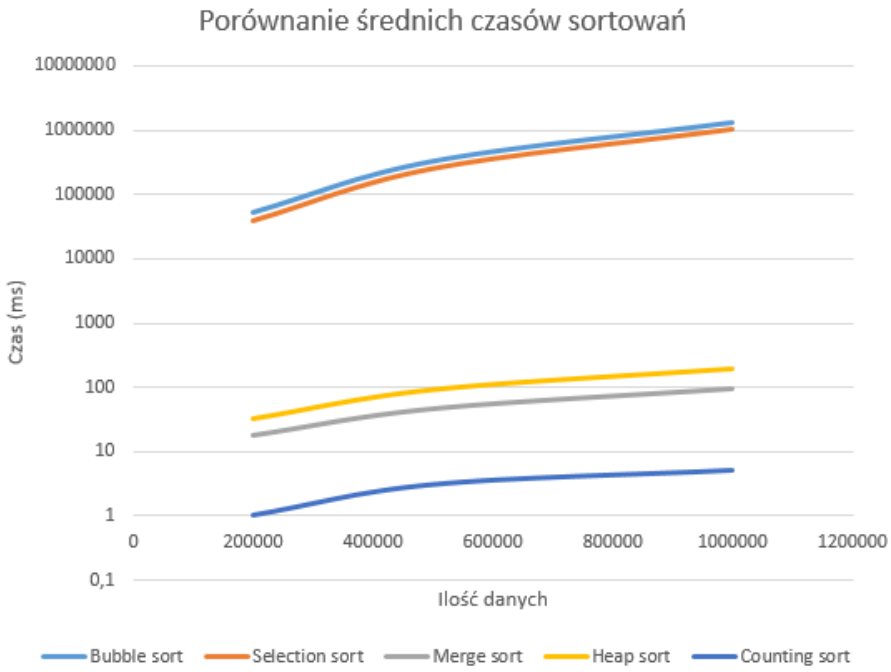
Obserwacje i wnioski
Jak widać złożoność liniowa skutkuje fenomenalnymi czasami sortowania nie zależnie od rodzaju danych wejściowych.

5 Porównanie algorytmów

Posiadając czasy wykonywania wyżej opisanych algorytmów, obliczyłem średni czas dla każdego sposobu sortowania. Następnie, zestawiłem je na wykresie poniżej.



Sporządziłem również wykres w skali logarytmicznej dla większej czytelności.



Obserwacje i wnioski
Na wykresie logarytmicznym widać podział na trzy wymienione grupy algorytmów. Co więcej, przepaść czasowa pomiędzy algorytmami naiwnymi a pozostałymi jest bardzo uwidoczniona. Ta różnica sprawia, że algorytmy naiwne nie są wykorzystywane na taką skalę jak reszta sortowań. Różnica między
counting sortem a algorytmami „dziel i zwyciężaj" jest również widoczna, jednakże, jest ona znacznie mniejsza, co pozwala na dyskusję na temat ich przydatności i zastosowań.

6 Zakończenie

W tym opracowaniu, zostało przedstawionych pięć algorytmów sortowania, ich cechy, oraz efektywność czasowa. Została również wyznaczona różnica czasowa pomiędzy grupami algorytmów.

Spis treści