AISDI zadanie 1

-

Algorytmy sortowania

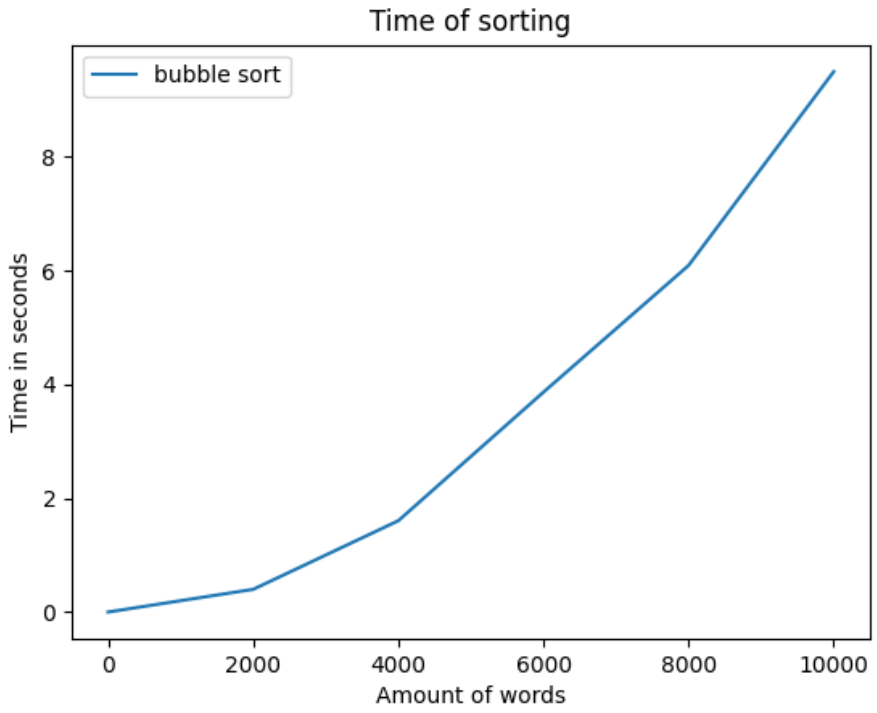
Mikołaj Rożek, Aleksandra Szymańska

1. Opisy algorytmów sortujących

Funkcja change element positions() odpowiada za zamianę miejsc dwóch elementów w liście.

1.1. Bubble sort

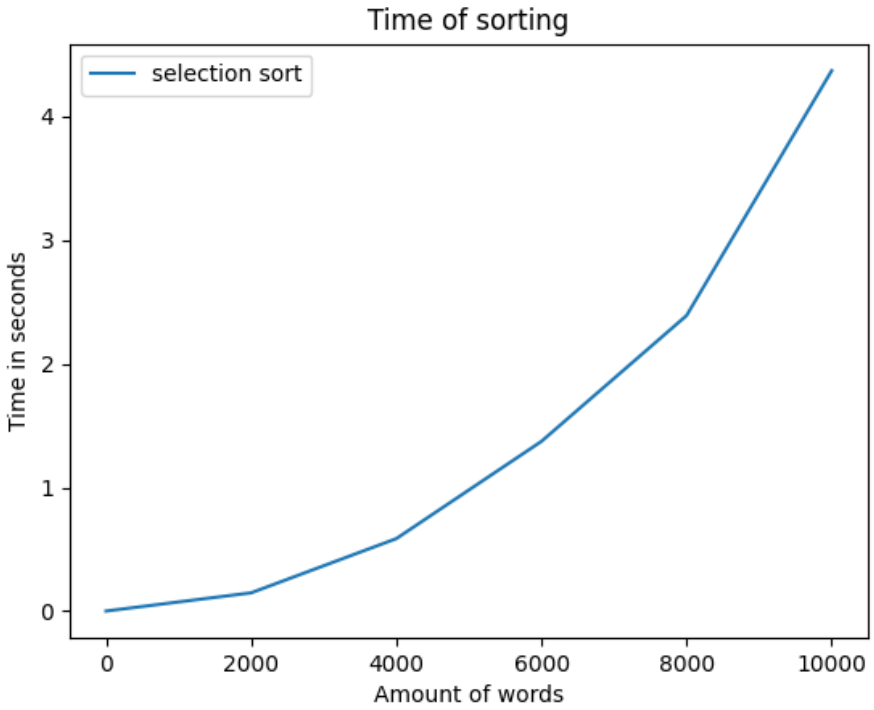
Algorytm sortowania BubbleSort polega na sortowaniu poprzez porównywanie elementów o sąsiednich indeksach oraz zamianie ich kolejności jeżeli jeden jest większy od drugiego. Ten proces porównywania i zamiany powtarza się wielokrotnie aż do momentu, gdy wszystkie elementy zostaną uporządkowane według porządku sortowania. W kodzie, początkowo ustalamy długość listy i iterujemy po niej za pomocą pętli for. Wewnętrzna pętla for iteruje po kolejnych parach sąsiednich elementów i porównuje ich wartości. Jeśli wartość elementu j jest większa od elementu j+1, to wywołujemy funkcję **"change element positions()"** w celu zamiany ich miejscami. Po zakończeniu każdej iteracji zewnętrznej pętli for, sprawdzamy, czy dokonaliśmy jakiejkolwiek zamiany. Jeśli nie, oznacza to, że lista jest już posortowana i kończymy działanie algorytmu.



Rysunek 1. Wykres czasu od ilości posortowanych słow algorytmu Bubble Sort

1.2. Selection sort

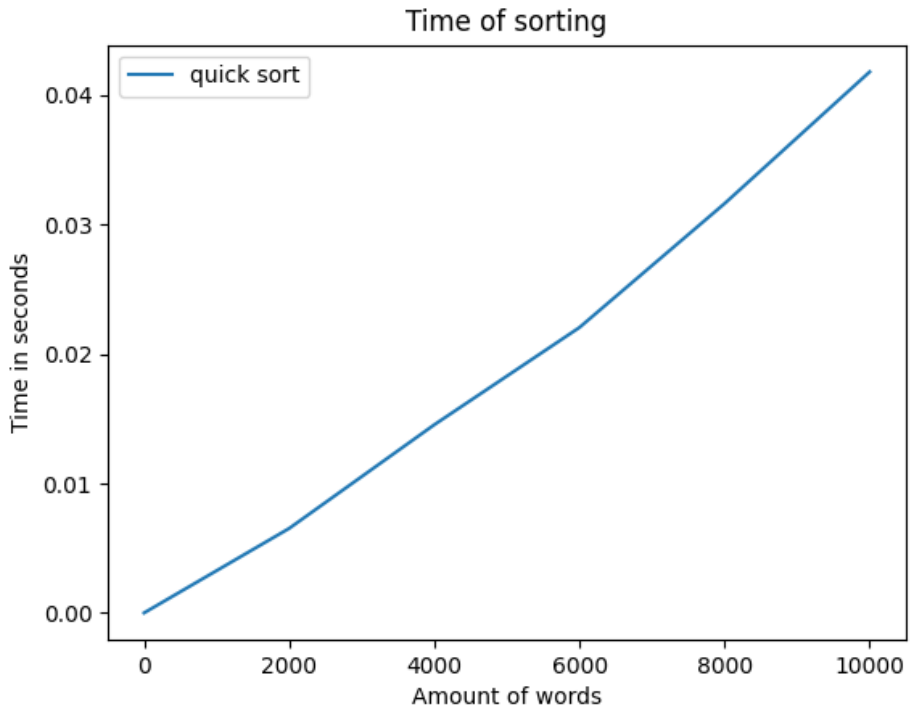
Selection Sort rozpoczyna się od iteracji po wszystkich elementach wejściowej listy. W każdej iteracji pętli zapisywany jest indeks najwcześniejszego (najmniejszego) elementu w nieposortowanej części listy. Następnie wewnętrzna pętla for iteruje po pozostałych elementach nieposortowanej części listy i szuka kolejnego elementu o najwcześniejszym indeksie. Jeśli jakiś element ma indeks wcześniejszy niż obecnie najwcześniejszy, to jego indeks zostaje zapisany jako najwcześniejszy. W końcu, zamiana miejscami elementów, których indeksy odpowiadają indeksom najwcześniejszego i i-tego elementu, umieszcza najwcześniejszy element na jego właściwej pozycji w posortowanej liście.



Rysunek 2. Wykres czasu od ilości posortowanych słow algoryutmu Selection Sort

1.3. Quick sort

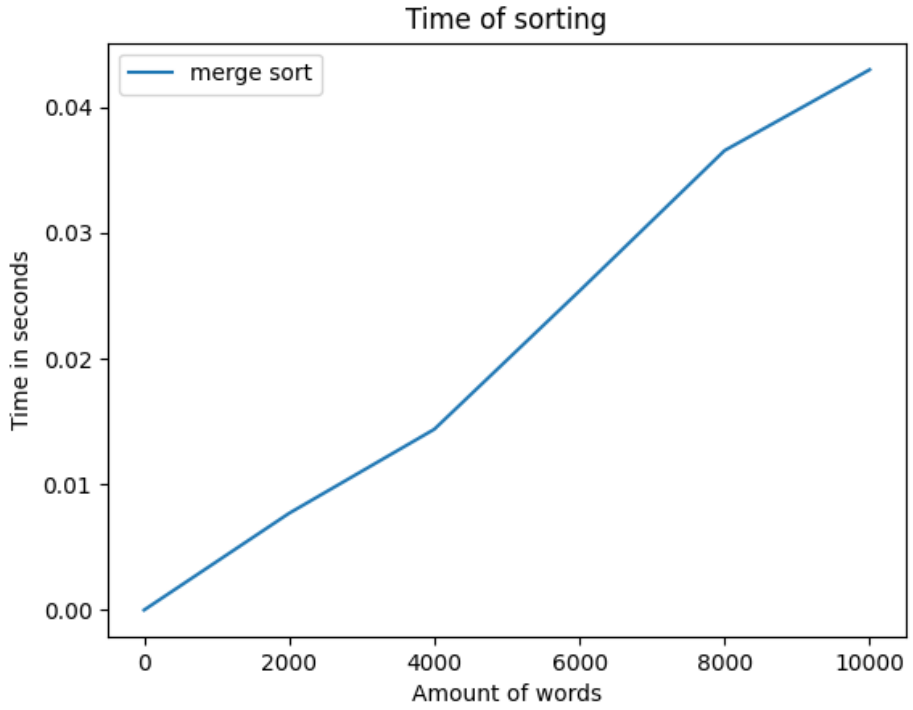
Algorytm Quick Sort jest algorytmem sortowania dziel i zwyciężaj. Polega on na podziale listy na mniejsze części, a następnie sortowaniu tych części. W przypadku Lomuto partition scheme, które jest wykorzystane w naszym kodzie, element o losowym indeksie zostaje wybrany jako **'pick'**. Jest on wybierany za pomocą funkcji randint(), która zwraca losową wartość między startem a końcem listy. Następnie wartość **'pick'** jest zamieniana z ostatnim elementem listy, który jest używany jako **'end'**. W kolejnym kroku pętla for iteruje po elementach listy od startu do końca, aż do elementu poprzedzającego **'pick'**. Jeśli dany element jest mniejszy od wartości **'pick'**, to zostaje zamieniony z elementem, który znajduje się na bieżącej pozycji, a ta pozycja jest przesuwana w prawo. Po przejściu przez całą listę, **'pick'** jest zamieniany z ostatnim elementem, który nie był mniejszy od niego. Następnie algorytm Quick Sort zostaje wywołany rekurencyjnie dla dwóch części listy: od startu do pozycji bieżącej minus jeden i od pozycji bieżącej plus jeden do końca. W ten sposób wszystkie części listy zostają posortowane.



Rysunek 3. Wykres czasu od ilości posortowanych słow algorytmu Quick Sort

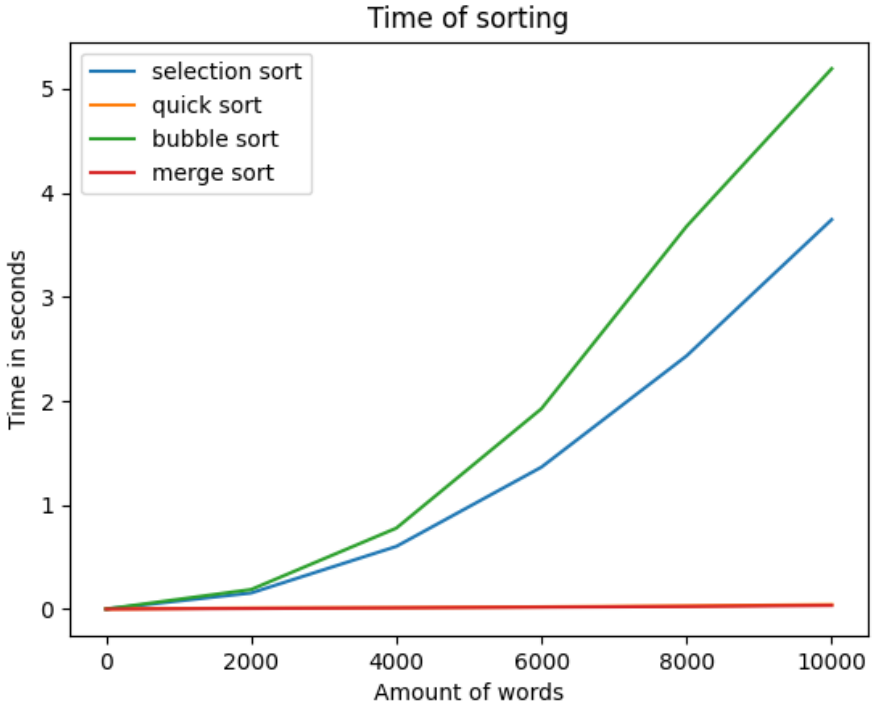
1.4. Merge sort

Algorytm Merge Sort polega na dzieleniu listy na dwie równe części, a następnie sortowaniu każdej z nich rekurencyjnie przez scalanie posortowanych podlist. W naszym kodzie implementacja rozpoczyna od podziału listy na połowę, a następnie wywołuje się na każdej z tych podlist rekurencyjnie. Gdy lista będzie miała tylko jeden element lub będzie pusta, funkcja zwraca ją jako posortowaną. Następnie wywołuje się funkcję scalającą dwie posortowane listy, co daje ostatecznie posortowaną listę wynikową. Warto zauważyć, że funkcja scalająca nie wymaga dodatkowej pamięci, ponieważ elementy są przepisywane do listy tymczasowej.



Rysunek 4. Wykres czasu od ilości posortowanych słow algortymu Merge Sort

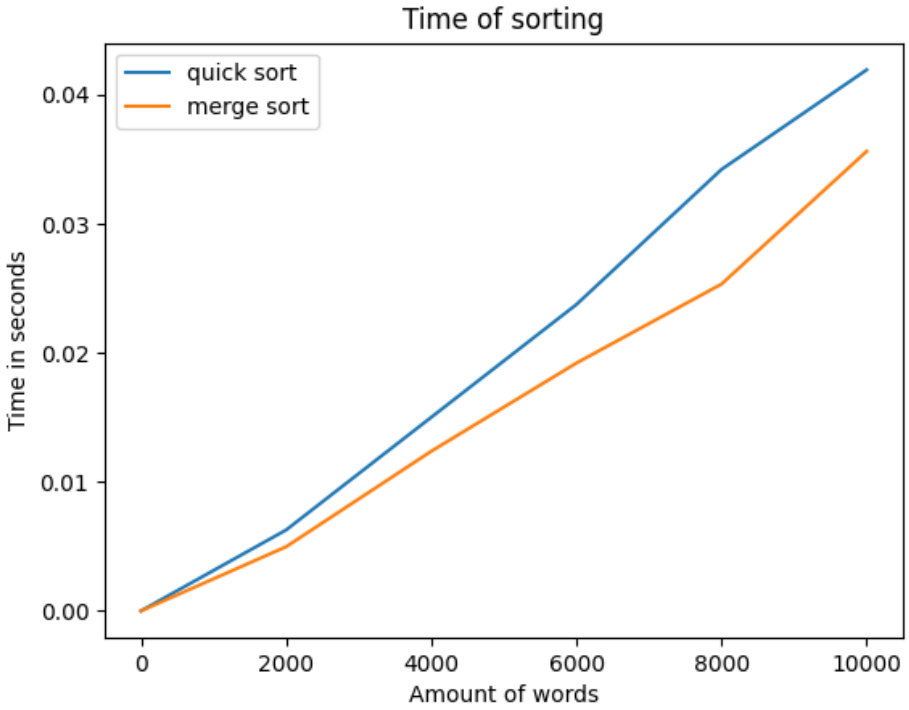
2. Porównanie wszystkich algorytmów



Rysunek 5. Wykresy wszystkich algorytmów

3. Wnioski

Najbardziej wydajnym algorytmem okazał się Merge Sort, a najwolniejszym Bubble Sort. Quick Sort był szybszy od Selection Sort, ale wolniejszy od Merge Sort. Merge Sort zapewnił szybkość dzięki swojej naturze dzielenia listy na połowy i rekurencyjnemu łączeniu ich w posortowanej kolejności. W przypadku sortowania słów z "Pana Tadeusza"Bubble Sort okazał się najwolniejszy ze względu na dużą liczbę elementów, które trzeba było przesuwać na końcu listy. Dlatego Bubble Sort jest mniej wydajnym algorytmem w porównaniu do innych algorytmów sortujących, zwłaszcza w przypadku długich list. Merge Sort oraz Quick Sort są najbardziej wydajnymi algorytmami i to je opłaca się stosować do sortowania bardzo długich list.



Rysunek 6. Porównanie Merge Sorta oraz Quick Sorta