LABORATORIUM 1, ALGORYTMY SORTOWANIA

Zespół: Zuzanna Filipkowska, Aleksandra Sypuła

Środowisko: Visual Studio Code

Sposób uruchomienia:

Program należy wywoływać z konsoli poleceniem: `python3 plotting.py plik.txt`

Testy dla poszczególnych plików można uruchomić np. komendą:

```
pytest test_nazwa_algorytmu.py
```

Podział pracy:

Zuzanna Filipkowska:

- Algorytmy sortowania: bąbelkowe, przez scalanie
- Testy jednostkowe dla funkcji (a za dostosowanie testów do odpowiednich algorytmów odpowiedzialna była osoba od danej funkcji sortującej)
- Możliwość wywołania głównej funkcji z linii poleceń
- Generowanie wykresów dla porównywania algorytmów

Aleksandra Sypuła:

- Algorytmy sortowania: szybkie, przez wstawianie
- Generowanie wykresów dla porównywania algorytmów
- Przekształcanie pliku .txt na tablicę słów
- Obliczanie czasu sortowania dla algorytmów (z biblioteką time)

Do każdego z algorytmów zostały napisane testy przy użyciu framework'u pytest.

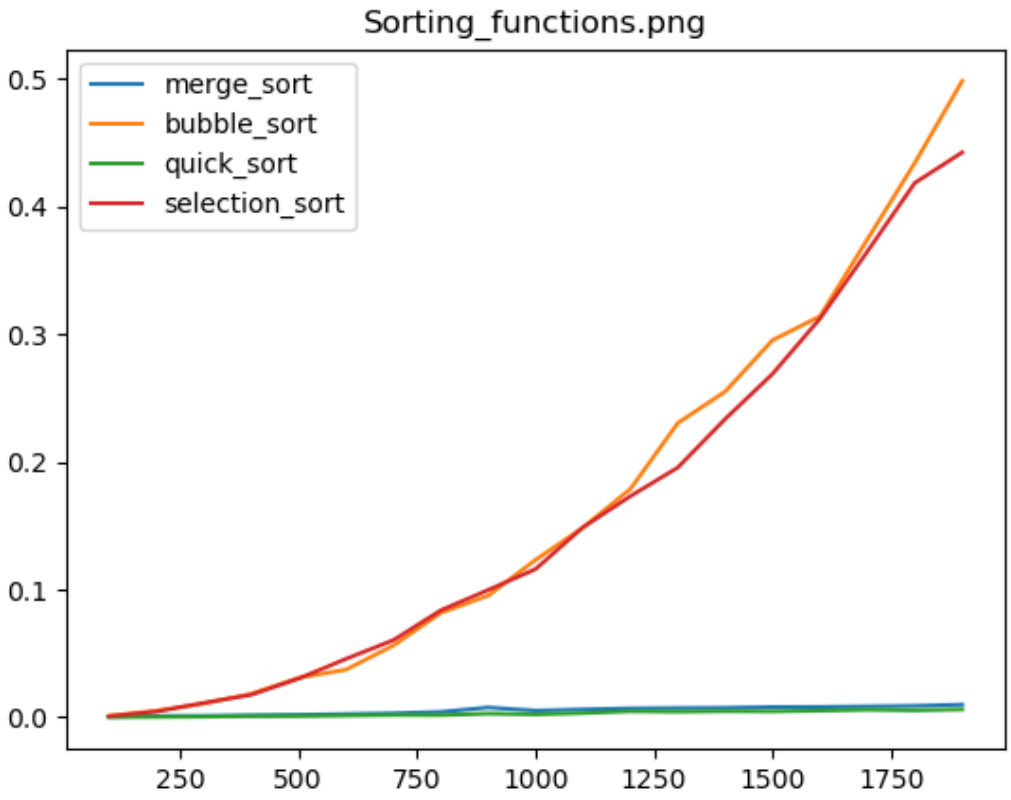
Algorytmy pomyślnie przechodzą testy.

WNIOSKI:

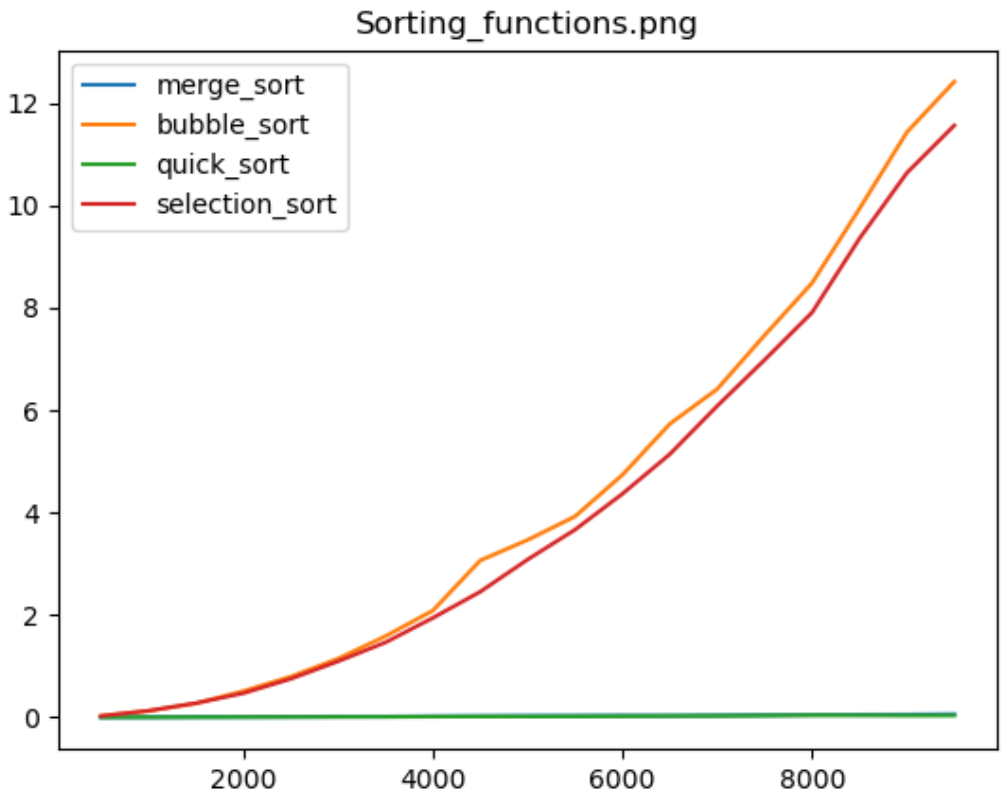
Po utworzeniu wykresów możemy zaobserwować, że najbardziej efektywnymi i naszybszymi algorytmami z czterech testowanych są quick_sort oraz merge_sort. Ich czasy działania są bardzo zbliżone do siebie, chociaż wraz ze zwiększaniem się liczby elementów do sortowania możemy zauważyć, że czas działania algorytmu quick_sort zaczyna być odrobinę bardziej efektywny (niewielkie różnice widoczne są na wykresie 1, na wykresie 2 przy większej ilości elementów różnice te nie są widoczne, co może być spowodowane mniejszą dokładnością czasów na osi y, co spowodowane jest znacznie dłuższymi czasami sortowania bąbelkowego oraz przez wstawianie). Dla pozostałych dwóch algorytmów sortowania: selection_sort oraz bubble_sort czasy sortowania rosną bardzo szybko (po nawet nieznacznym zwiększeniu słów do posortowania). Na wykresie drugim – dla większej liczby elementów do posortowania widoczne jest, że najwolniej działającym z algorytmów jest bubble_sort, który jeszcze do 4 000 elementów do sortowania działa podobnie do selection_sort, jednak powyżej tej liczby staje się najmniej efektywny. Dla przetestowanych algorytmów, możemy zauważyć że wykresy ich

czasów działania nie są dość „regularnymi” liniamii, dla niektórych wartości elementów sortowania widoczne są pewne skoki czasu, co może być spowodowane innymi procesami równolegle działającymi na naszych komputerach oraz niedokładnościami w liczeniu czasu z wykorzystaniem w Pythonie biblioteki time. Z tych powodów również dość wiarygodnym do porównywania algorytmów sortowania jest wykres drugi z maksymalną liczbą elementów równą 10 000, chociaż czasy działania tych funkcji widoczne na wykresie możemy traktować jedynie poglądowo, z uwagi na inne procesy komputera działające w tym samym czasie co testowane przez nas funkcje.

Bazując na samym zapisie i funkcjonalności napisanych przez nas algorytmów sortowania najszybszymi algorytmami powinny być quick sort oraz merge sort o średniej złożoności czasowej O(nlogn) i to one dla dużej liczby słów powinny dawać najlepsze czasowe rezultaty. Pozostałe dwa algorytmy: bubble sort oraz selection sort o złożoności średniej $O(n^2)$ dla długiej listy argumentów powinny dawać gorsze rezultaty. Wspólną cechą wszystkich algorytmów sortowania jest również rosnący czas działania programu dla zwiększającej się liczby elementów do posortowania. Powyższe właściwości są widoczne na wykresach czasowych działania naszych algorytmów sortowania – najszybszymi algorytmami są sortowanie szybkie oraz przez scalanie a najwolniejszymi przez wstawianie oraz bąbelkowe. Na wykresach widoczne są również jak ogromne różnice w czasie działania dają algorytmy o złożoności czasowej O(nlogn) a $O(n^2)$.



Wykres dla algorytmów sortowania ze zwiększaniem ilości słów o 100 elementów, aż do 2000 elementów.



Wykres dla algorytmów sortowania ze zwiększaniem ilości słów o 500 elementów, aż do 10 000 elementów.