Sprawozdanie z projektu nr 3

W ramach projektu zadaniem było zbadanie efektywności algorytmów sortowania. Wybrano pięć takowych algorytmów: *Insertion Sort, Selection Sort, Heap Sort* oraz *Cocktail Sort*.

Podczas analizy wzięto pod uwagę pomiar szybkości sortowania tablic.

Kody źródłowe wykorzystane w eksperymencie można odnaleźć w repozytorium https://github.com/daniel-sobczak-wsb/sort_algorithmsw katalogu „sort_codes”, zaś w katalogu „results” zawarto arkusze z wynikami pomiarów. Dodatkowo w katalogu „charts” można znaleźć znajdujące się poniżej wykresy w postaci wektorowej SVG.

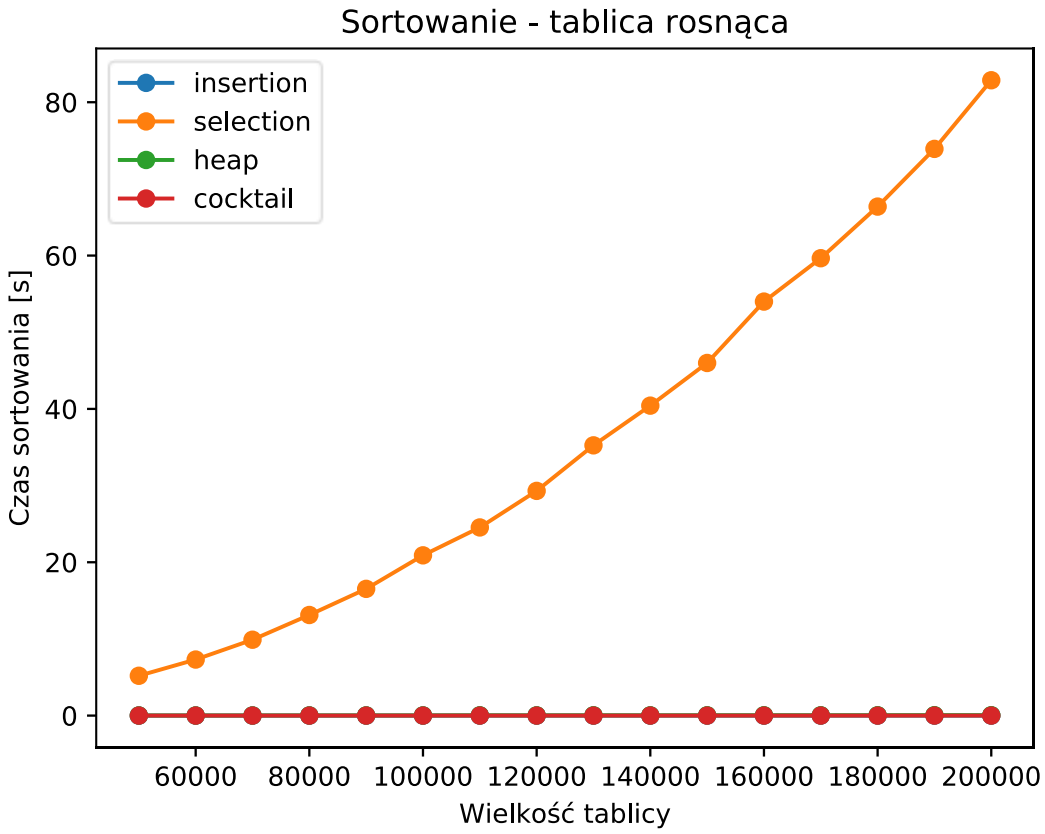
Poniższe pomiary zostały wykonane na laptopie Lenovo ThinkPad T440 (procesor Intel Core i5-4300U) z wykorzystaniem edytora Visual Studio Code dostosowanym do użytku z platformą .NET Core. Wykresy wykonano przy użyciu biblioteki do Pythona Matplotlib.

Badania przeprowadzono do poniższego zbioru punktów pomiarowych:

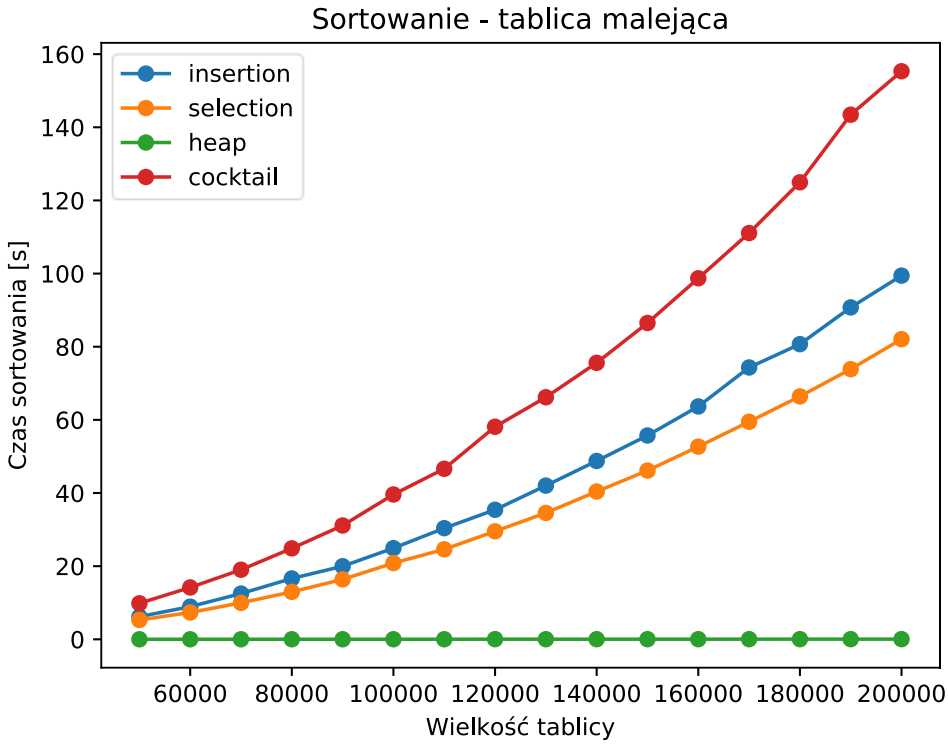
{ 50000, 60000, 70000, 80000, 90000, 100000, 110000, 120000, 130000, 140000, 150000, 160000, 170000, 180000, 190000, 20000 }

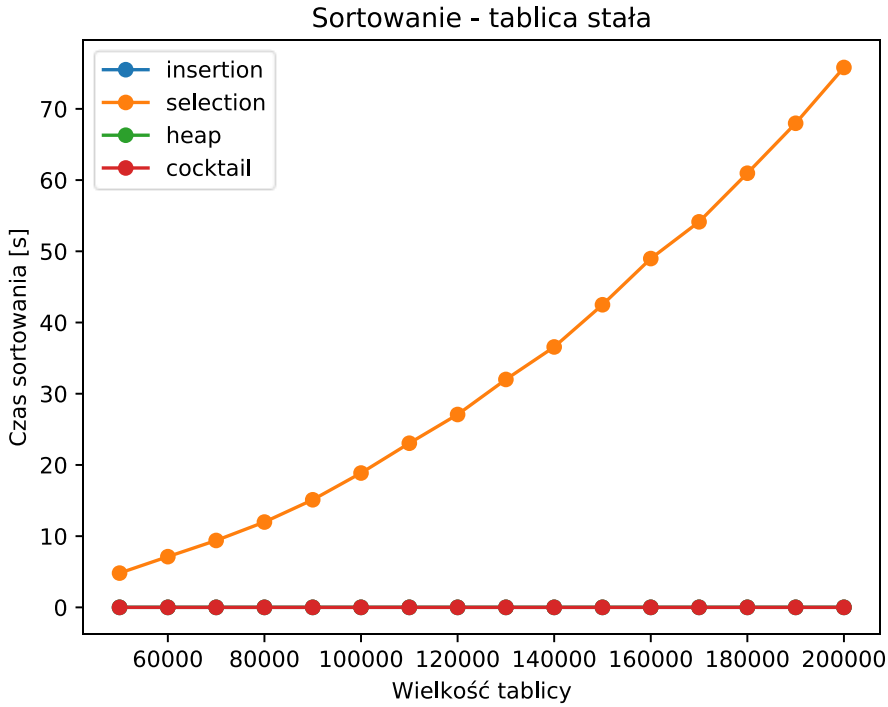
Część 1

W części pierwszej porównano szybkość działania pierwszych czterech algorytmów sortowania w zależności od wstępnego uporządkowania tablicy. Dla testów przygotowano tablice: rosnącą, malejącą, losową, V-kształtną oraz stałą.

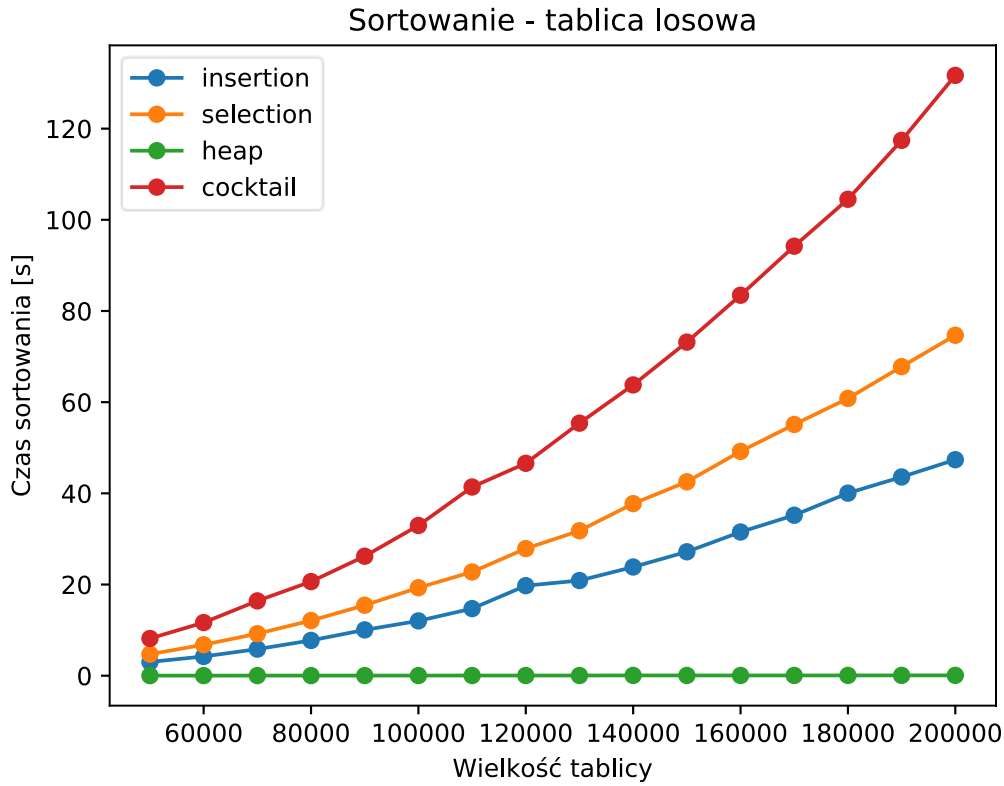


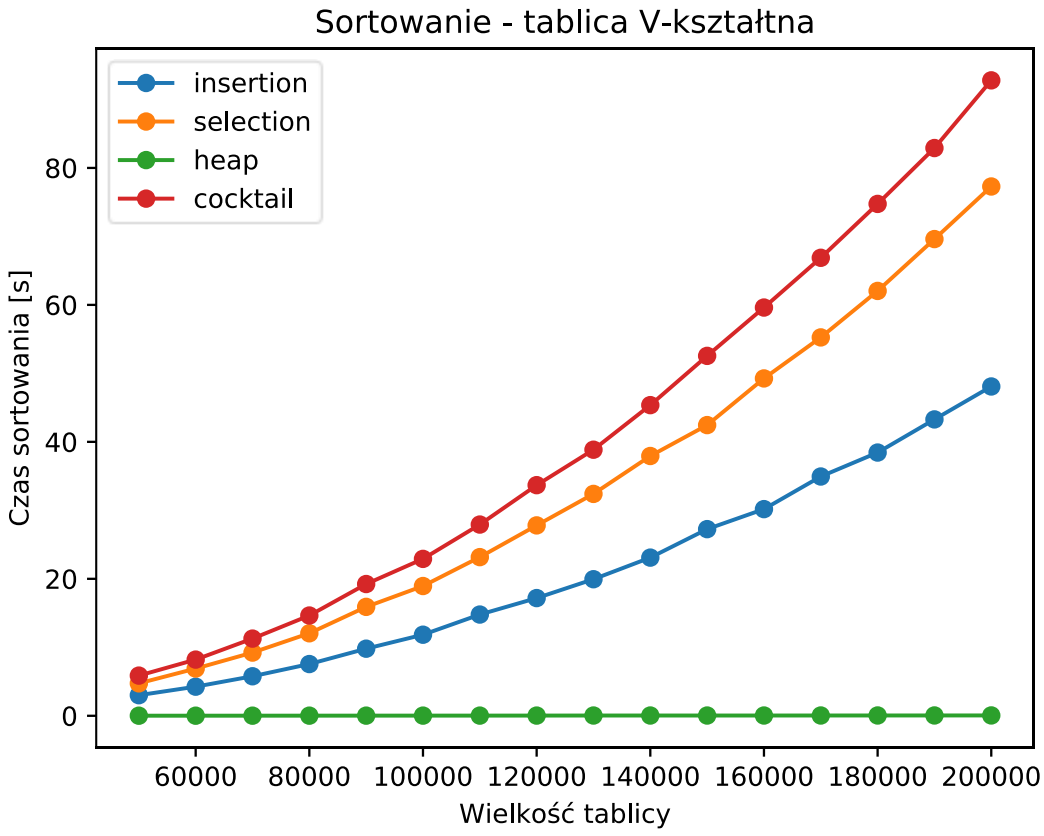
Cocktail Sort, Heap Sort i Insertion Sort dały na tyle zbliżone wyniki, że ich linie na wykresie się nakładają





Cocktail Sort, Heap Sort i Insertion Sort dały na tyle zbliżone wyniki, że ich linie na wykresie się nakładają


Sortowanie - tablica losowa
insertion
selection
heap
cocktail
Czas sortowania [s]
0
20
40
60
80
100
120
60000
80000
100000
120000
140000
160000
180000
200000
Wielkość tablicy


Sortowanie - tablica V-kształtna
insertion
selection
heap
cocktail
Czas sortowania [s]
0
20
40
60
80
60000
80000
100000
120000
140000
160000
180000
200000
Wielkość tablicy

Wnioski z części 1

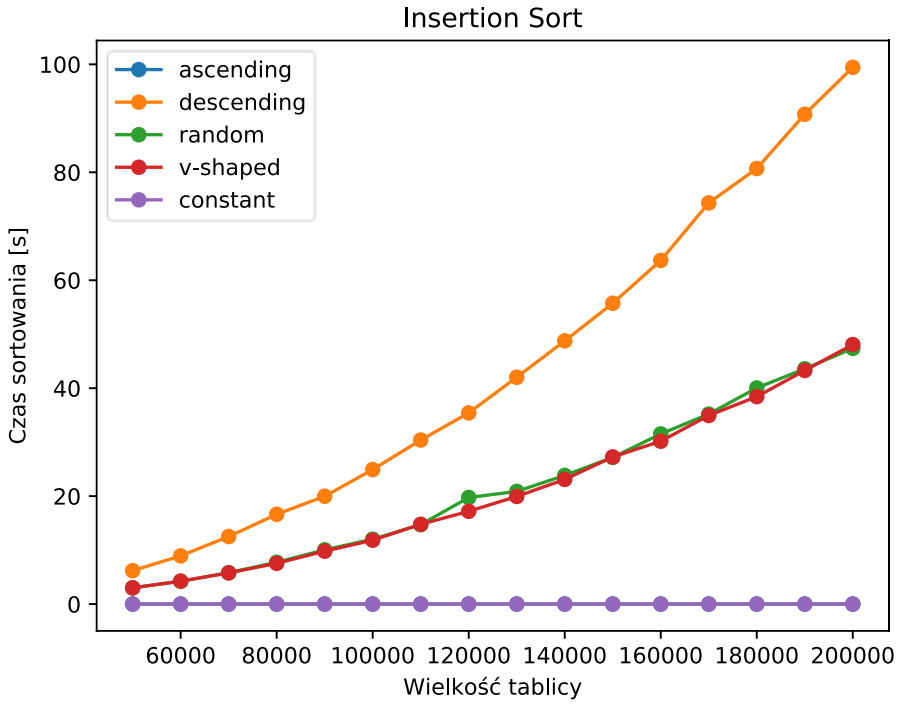
Selection Sort, Insertion Sort oraz Cocktail Sort mają typową, optymistyczną i pesymistyczną złożoność obliczeniową $O(n^2)$ – wyjątkiem jest optymistyczny przypadek Insertion Sort i Cocktail Sort (tablica jest już wstępnie uporządkowana rosnąco bądź jest ze stałymi wartościami), gdzie złożoność jest liniowa O(n). Heap Sort natomiast ma złożoność obliczeniową O(n log n), niezależnie od rodzaju przypadku.

Wszystkie cztery algorytmy mają tą samą złożoność pamięciową O(1).

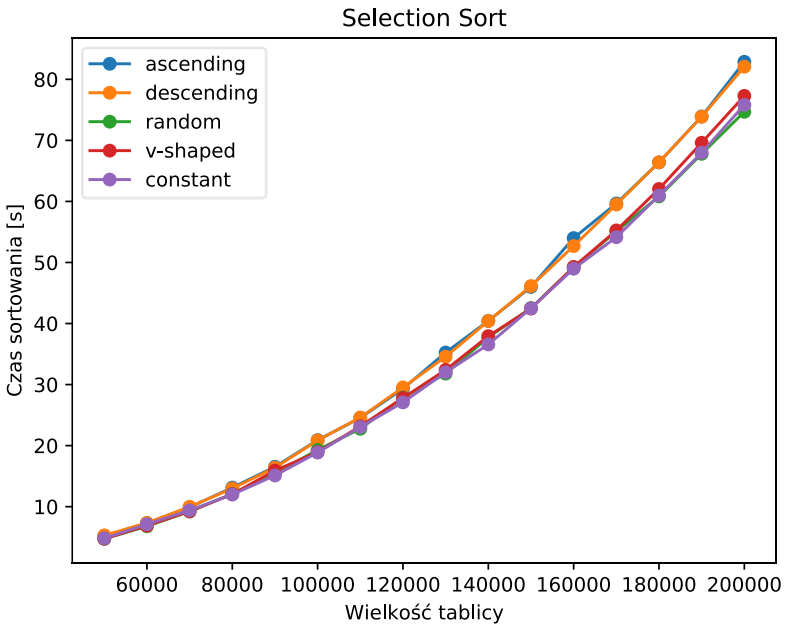
Największym przegranym tych pomiarów stał się Insertion Sort, który w żadnym z przypadków nie wykazał przewagi nad innymi algorytmami. Selection Sort wykazał się nie gorzej od Heap i Cocktail Sort przy już uporządkowanych tablicach. Cocktail Sort nie sprawdził się przy bardziej pesymistycznych przypadkach (tablice V-kształne, losowe i uporządkowane malejąco). Najbardziej uniwersalnym algorytmem okazał się Heap Sort – we wszystkich przypadkach efektywność była bardzo zadowalająca.

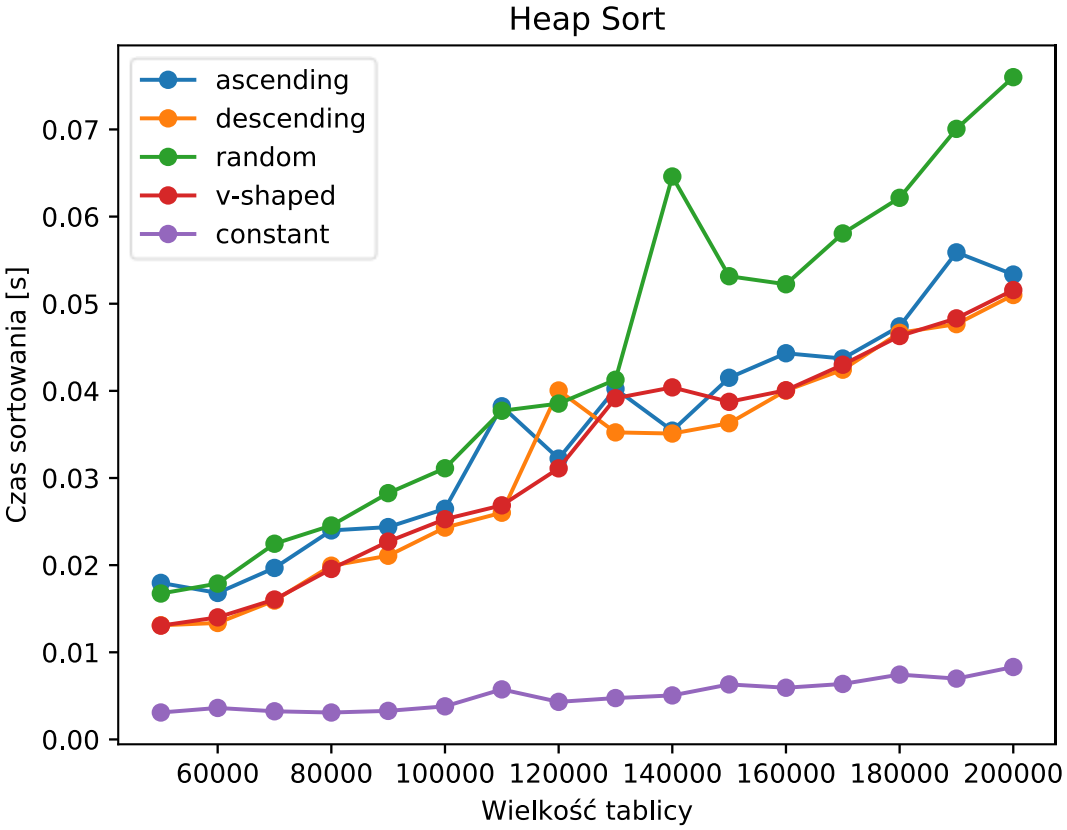
Część 2

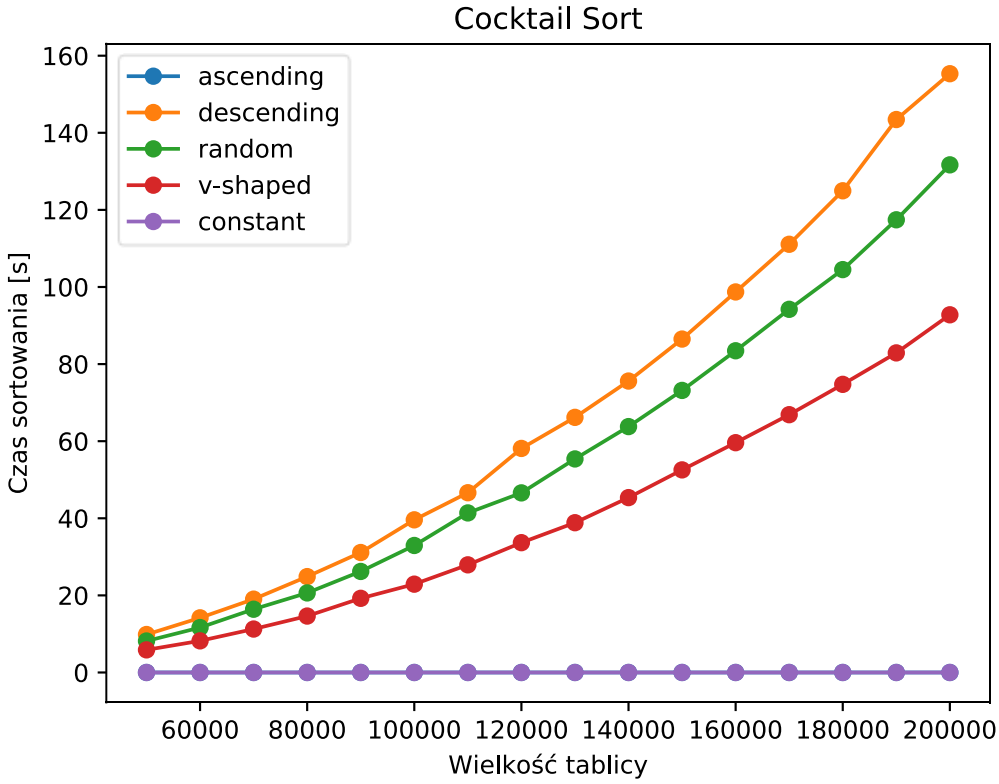
W części drugiej wykorzystano dane uzyskane przy części pierwszej, celem porównania algorytmów sortowania na tle wybranych typów wstępnego uporządkowania tablicy. Dla testów przygotowano tablice: rosnącą, malejącą, losową, V-kształtną oraz stałą.



Uporządkowanie stałe i rosnąco dało na tyle zbliżone wyniki, że ich linie na wykresie się nakładają







Uporządkowanie stałe i rosnąco dało na tyle zbliżone wyniki, że ich linie na wykresie się nakładają

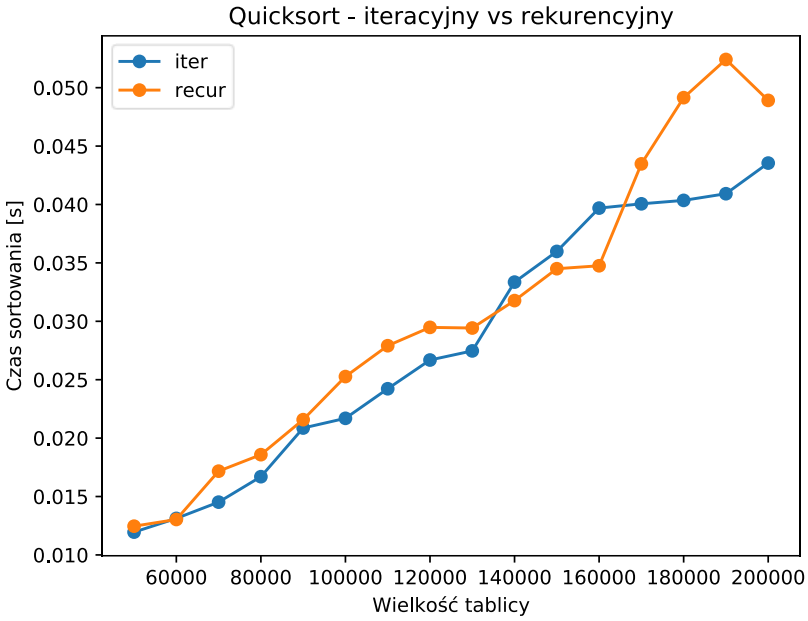
Wnioski z części 2

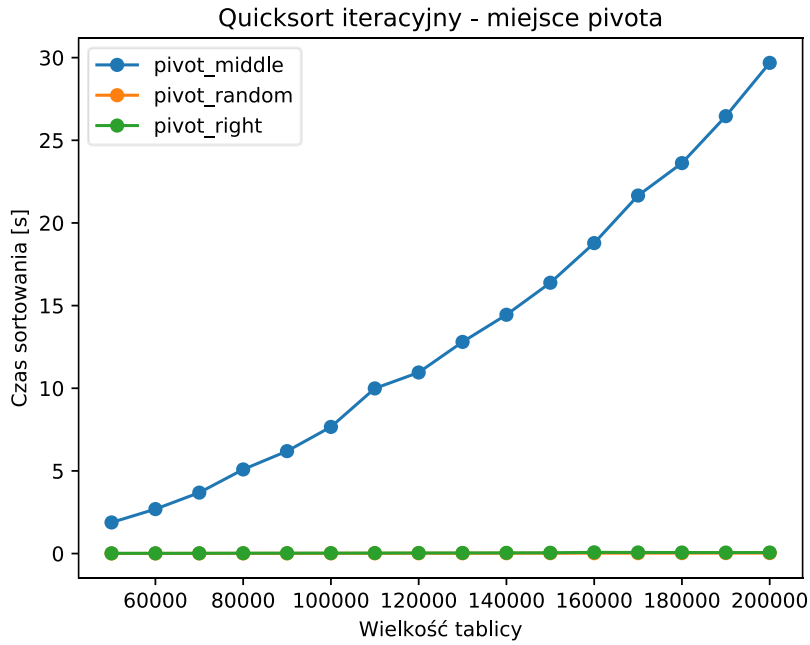
Najbardziej sprzyjającymi sortowaniu sposobami wstępnego uporządkowania tablicy były oczywiście uporządkowanie rosnąco lub stale (wyjątkiem było uporządkowanie rosnący przy Heap Sort), do przewidzenia było również tego, że wielkim przegranym będzie najbardziej pesymistyczny przypadek uporządkowani tablicy malejąco.

Ciekawym spostrzeżeniem jest to, że Selection Sort był wydajny w zbliżony sposób niezależnie od sposobu wstępnego posortowania tablicy. Porządek losowy lub v-kształtny w każdej sytuacji uplasował się pośrodku stawki.

Część 3

Ostatnią, osobną część projektu poświęcono popularnemu algorytmowi Quick Sort. Jej celem było jego zbadanie pod kątem wpływu sposobu implementacji (iteracyjnie bądź rekursywnie) oraz sposobu wyboru klucza do porównania (dalej nazwanego z angielskiego „pivotem”). Dla testów, jak poprzednio, przygotowano tablice: rosnącą, malejącą, losową, V-kształtną oraz stałą.





Losowy oraz skrajny na prawo element tablicy jako pivot dały na tyle zbliżone wyniki, że ich linie na wykresie się nakładają.

Wnioski z części 3

Sposób implementacji quicksorta nie wykazuje szczególnej przewagi do progu 170k elementów w tablicy. gdzie dalej na prowadzenie wysuwa się rekurencyjna wersja algorytmu.

Jego złożoność obliczeniowa w typowym oraz pesymistycznym przypadku O(n log n) jest jego zaletą, lecz przy co bardziej pesymistycznych przypadkach, może się zredukować nawet do bardzo niepożądanego $O(n^2)$.

Wybór klucza ma znaczący wpływ na efektywność tego algorytmu. Udowodniło to wykazanie miażdżącej przewagi losowego lub skrajnego prawo elementu tablicy jako pivota nad klasycznym przykładem wybrania środkowego elementu tablicy.