Sprawozdanie Algorytmy Sortowania

Autorzy:

Sebastian Nawrot 145177

Damian Trzybiński 145162

Środowisko: (Visual Studio Code)

Version: 1.43.0 (user setup)

Node.js: 12.8.1

V8: 7.8.279.23-electron.0

Python version: 3.7.4

Sprzęt:

Wersja: 10.0.18363 Kompilacja 18363

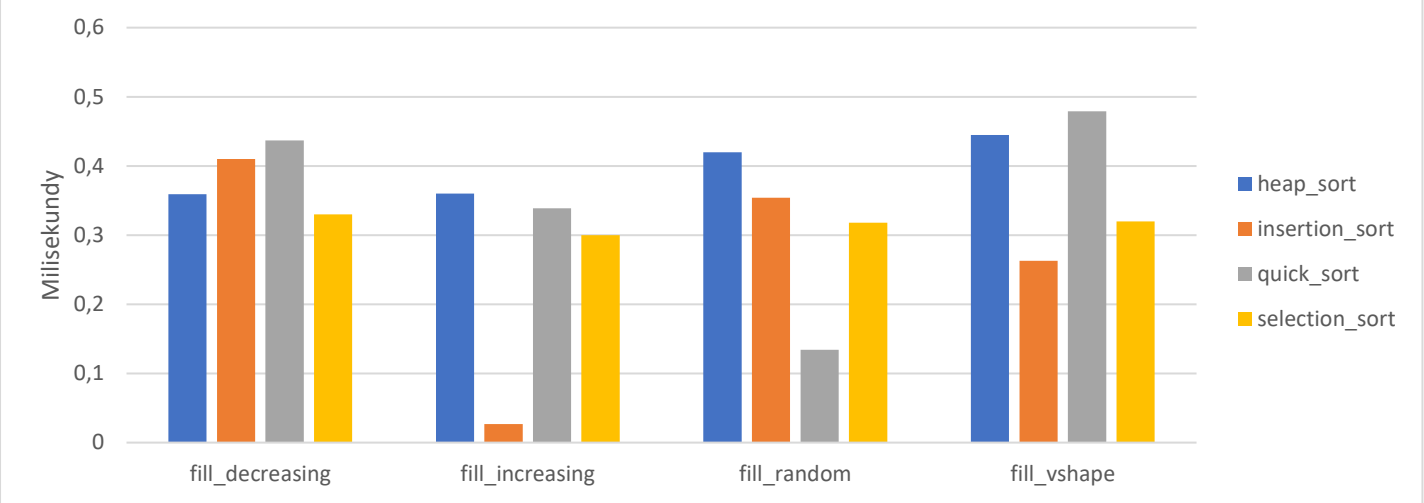
Typ systemu: x64-based PC

Procesor: Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 2601 MHz, Rdzenie: 2, Procesory logiczne: 4

Zainstalowana pamięć fizyczna (RAM): 16,0 GB

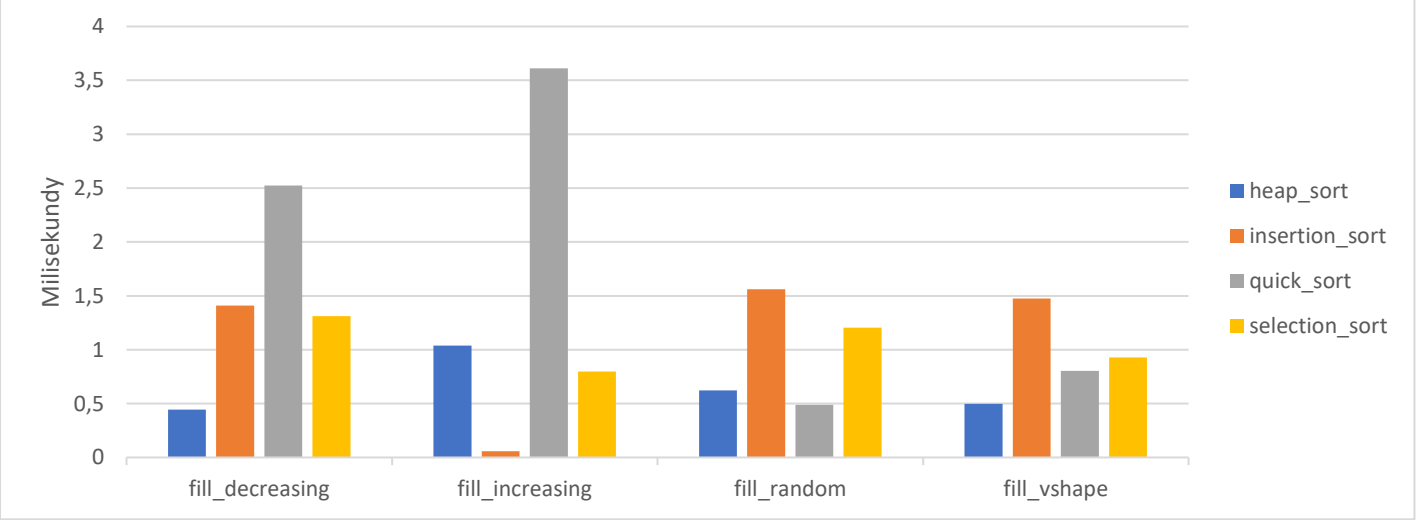
Tablica 50-elementowa:

Dla 50 elementów	heap_sort	insertion_sort	quick_sort	selection_sort
fill_decreasing	0,359 ms	0,41 ms	0,437 ms	0,33 ms
fill_increasing	0,36 ms	0,027 ms	0,339 ms	0,3 ms
fill_random	0,42 ms	0,354 ms	0,134 ms	0,318 ms
fill_vshape	0,445 ms	0,263 ms	0,479 ms	0,32 ms



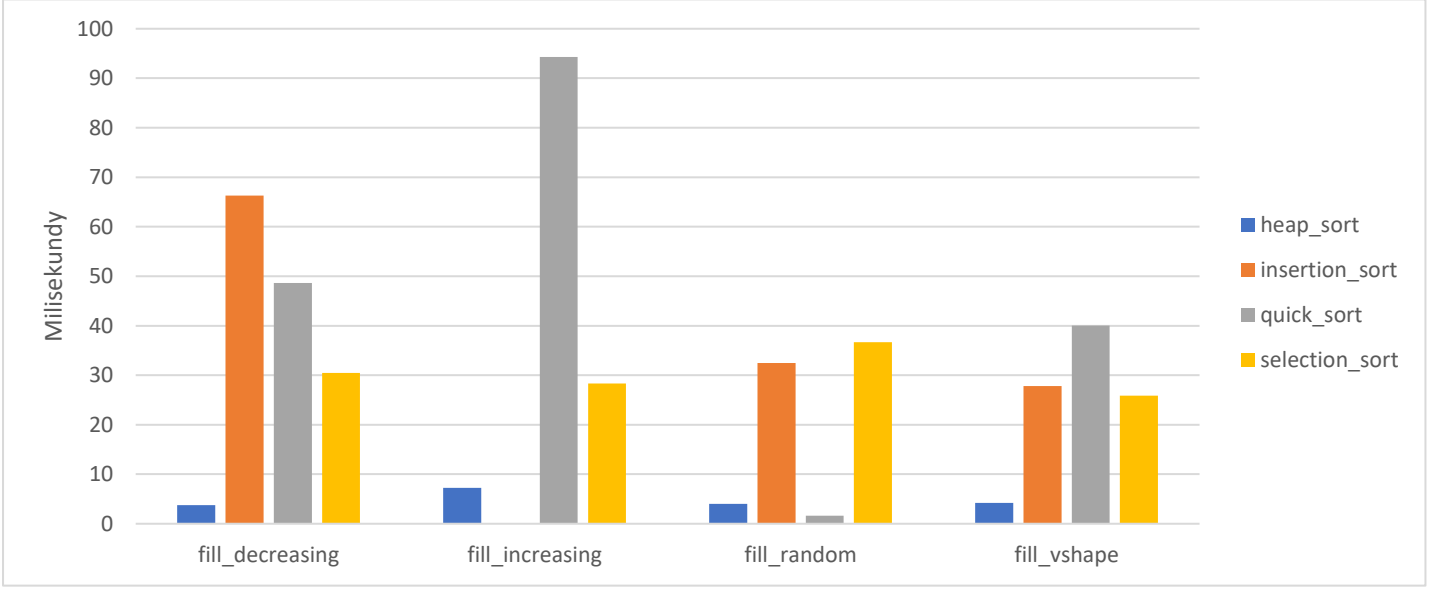
Tablica 100-elementowa:

Dla 100 elementów	heap_sort	insertion_sort	quick_sort	selection_sort
fill_decreasing	0,445 ms	1,409 ms	2,523 ms	1,313 ms
fill_increasing	1,038 ms	0,057 ms	3,61 ms	0,797 ms
fill_random	0,621 ms	1,561 ms	0,49 ms	1,205 ms
fill_vshape	0,497 ms	1,474 ms	0,804 ms	0,927 ms



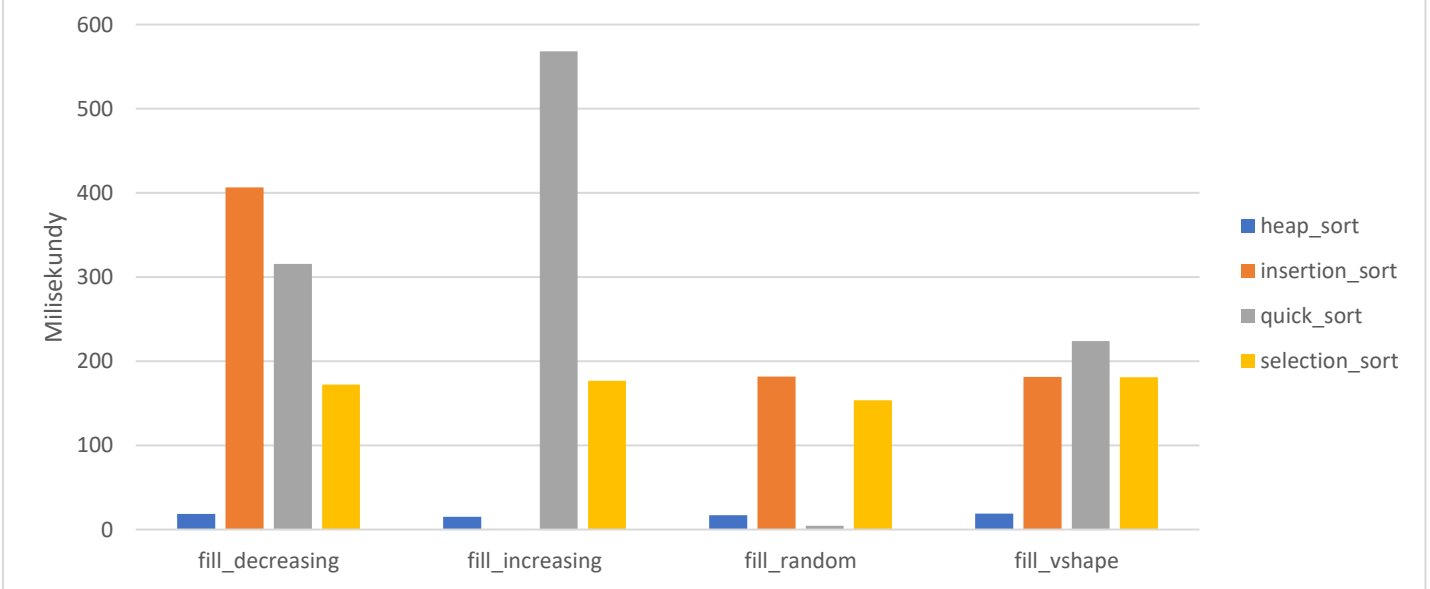
Tablica 800-elementowa:

Dla 800 elementów	heap_sort	insertion_sort	quick_sort	selection_sort
fill_decreasing	3,777 ms	66,27 ms	48,653 ms	30,484 ms
fill_increasing	7,278 ms	0,136 ms	94,301 ms	28,312 ms
fill_random	4,03 ms	32,476 ms	1,602 ms	36,655 ms
fill_vshape	4,211 ms	27,836 ms	40,06 ms	25,898 ms



Tablica 2000-elementowa:

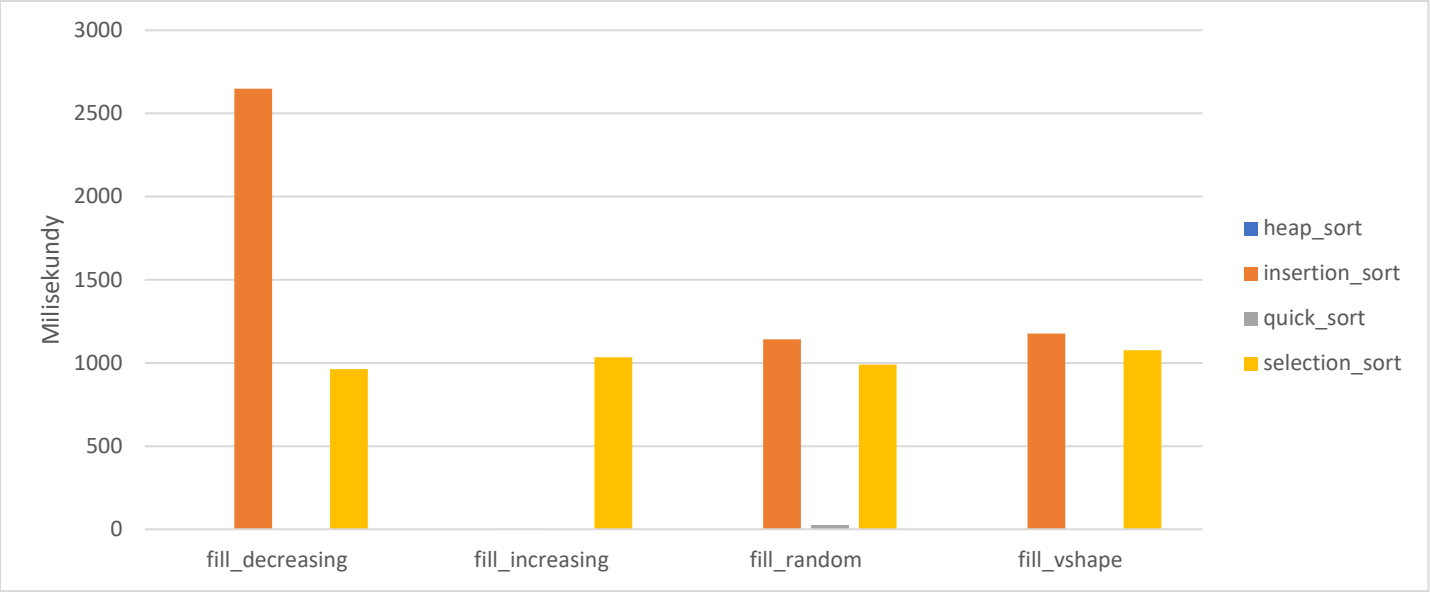
Dla 2000 elementów	heap_sort	insertion_sort	quick_sort	selection_sort
fill_decreasing	18,349 ms	406,56 ms	315,581 ms	172,139 ms
fill_increasing	14,948 ms	0,657 ms	568,282 ms	176,778 ms
fill_random	16,956 ms	181,861 ms	4,598 ms	153,681 ms
fill_vshape	19,104 ms	181,26 ms	223,834 ms	180,759 ms



Tablica 5000-elementowa:

Dla 5000 elementów	heap_sort	insertion_sort	quick_sort	selection_sort
fill_decreasing	N/A	2649,587 ms	N/A	962,298 ms
fill_increasing	N/A	0,902 ms	N/A	1034,296 ms
fill_random	N/A	1142,536 ms	22,66 ms	990,476 ms
fill_vshape	N/A	1176,815 ms	N/A	1075,88 ms

N/A: >10000 (Maximum recursion depth)



Wnioski:

Najlepszym algorytmem okazał się być heap sort. Wykazuje on jednakową tendencję przy wzroście elementów do posortowania. Algorytm quick sort wydaje się najgorszym spośród całej reszty. Tendencja wraz ze wzrostem rośnie niekorzystnie czasowo, jednak przy generowaniu losowym radzi sobie najlepiej. Krótkie zobrazowanie działania algorytmów:

Rodzaj	Krótko	Długo
heap_sort	malejąco	rosnąco
insertion_sort	rosnąco	malejąco
quick_sort	losowo	rosnąco
selection_sort	losowo	malejąco