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

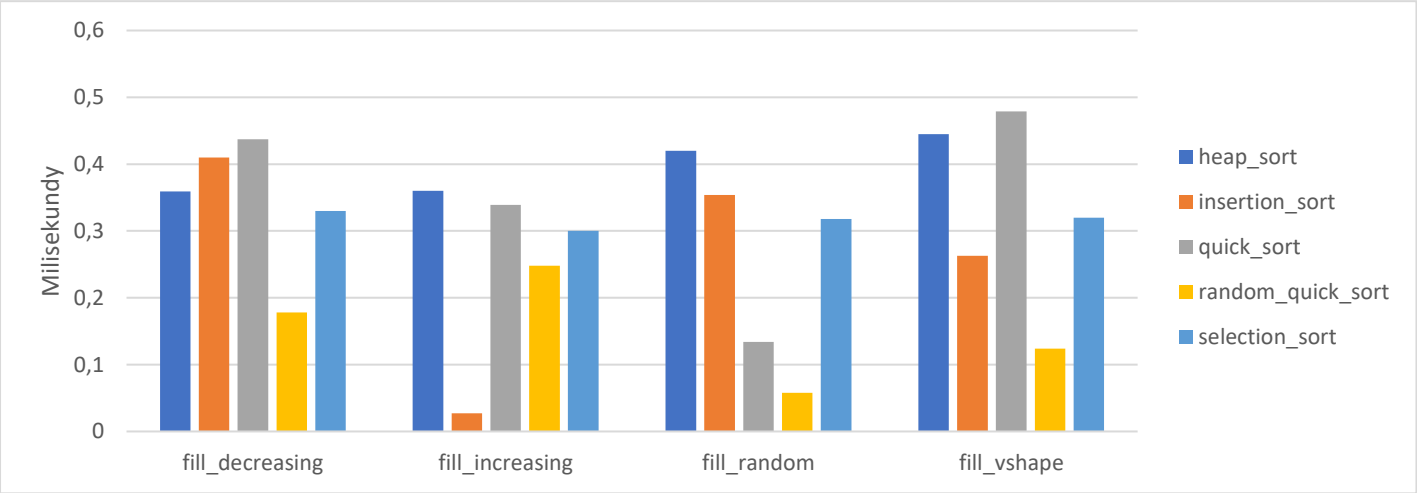
Typ systemu: x64-based PC

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

Zainstalowana pamięć fizyczna (RAM): 16,0 GB

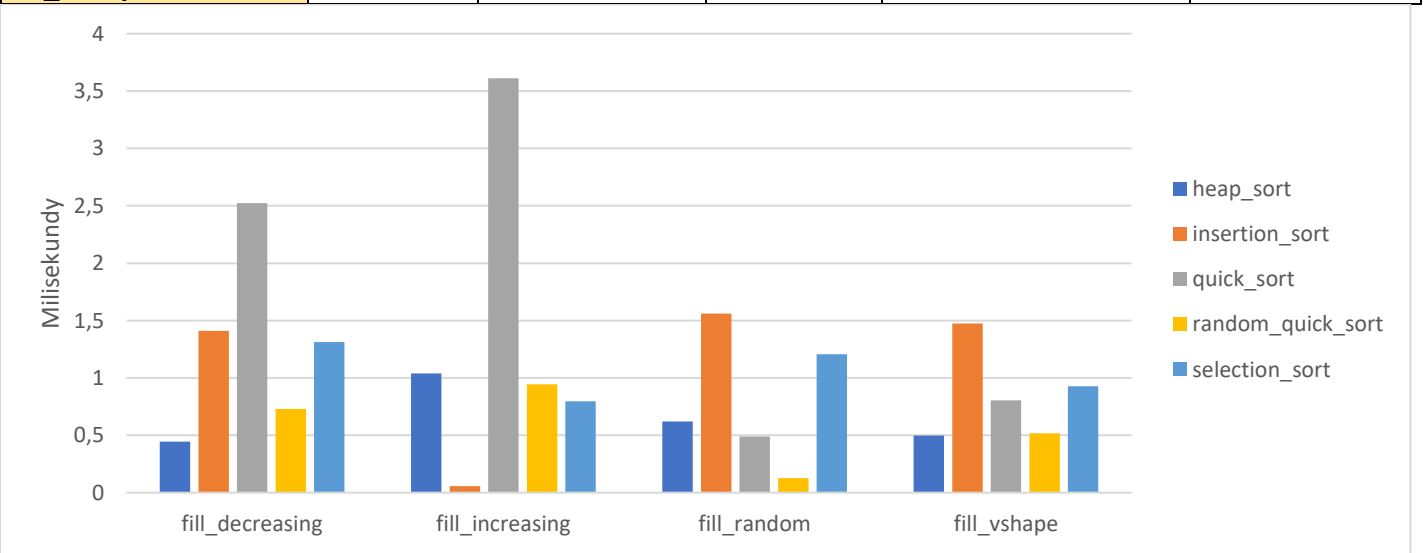
Tablica 50-elementowa: (w milisekundach)

Dla 50 elementów	heap_sort	insertion_sort	quick_sort	random_quick_sort	selection_sort
fill_decreasing	0,359	0,41	0,437	0,178	0,33
fill_increasing	0,36	0,027	0,339	0,248	0,3
fill_random	0,42	0,354	0,134	0,058	0,318
fill_vshape	0,445	0,263	0,479	0,124	0,32



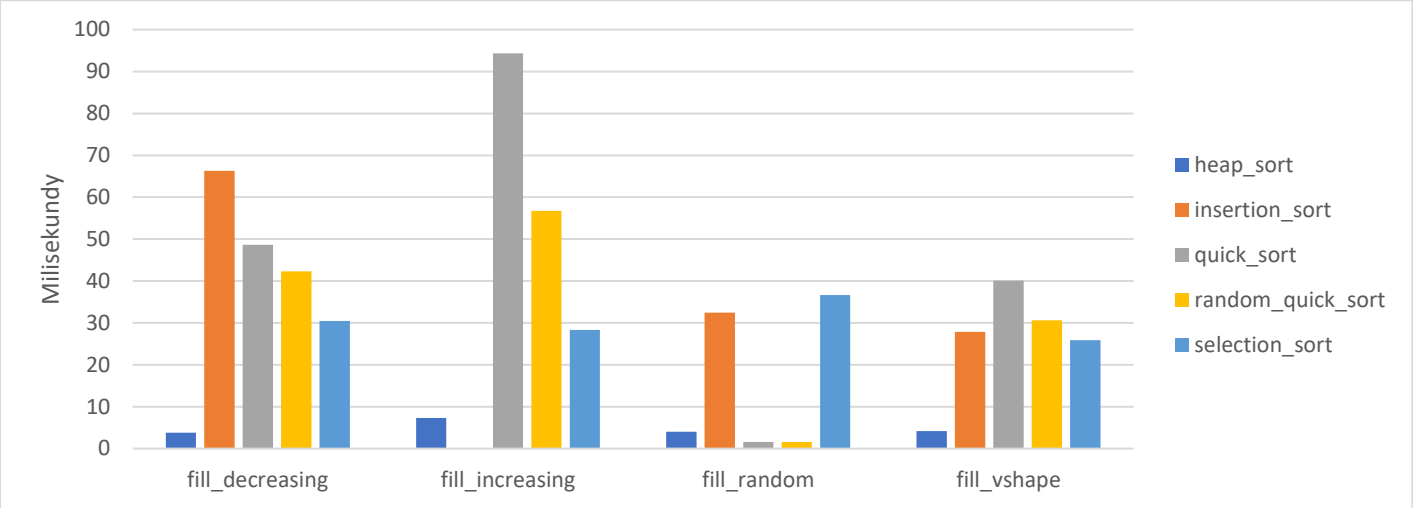
Tablica 100-elementowa: (w milisekundach)

Dla 100 elementów	heap_sort	insertion_sort	quick_sort	random_quick_sort	selection_sort
fill_decreasing	0,445	1,409	2,523	0,729	1,313
fill_increasing	1,038	0,057	3,61	0,943	0,797
fill_random	0,621	1,561	0,49	0,128	1,205
fill_vshape	0,497	1,474	0,804	0,516	0,927



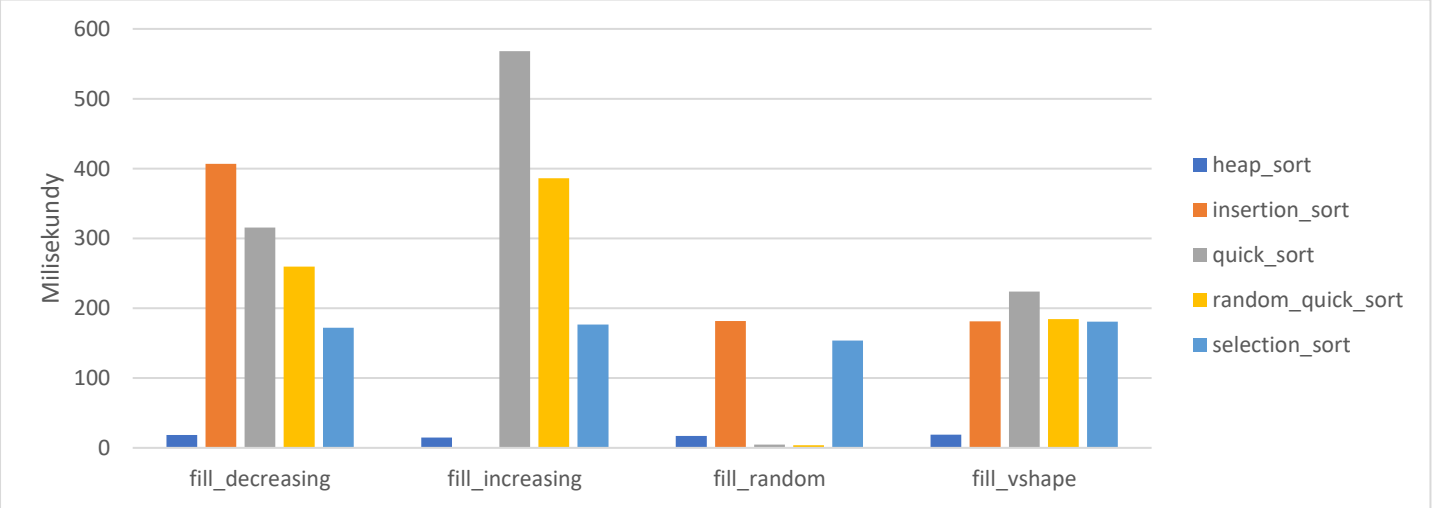
Tablica 800-elementowa: (w milisekundach)

Dla 800 elementów	heap_sort	insertion_sort	quick_sort	random_quick_sort	selection_sort
fill_decreasing	3,777	66,27	48,653	42,292	30,484
fill_increasing	7,278	0,136	94,301	56,773	28,312
fill_random	4,03	32,476	1,602	1,544	36,655
fill_vshape	4,211	27,836	40,06	30,623	25,898



Tablica 2000-elementowa: (w milisekundach)

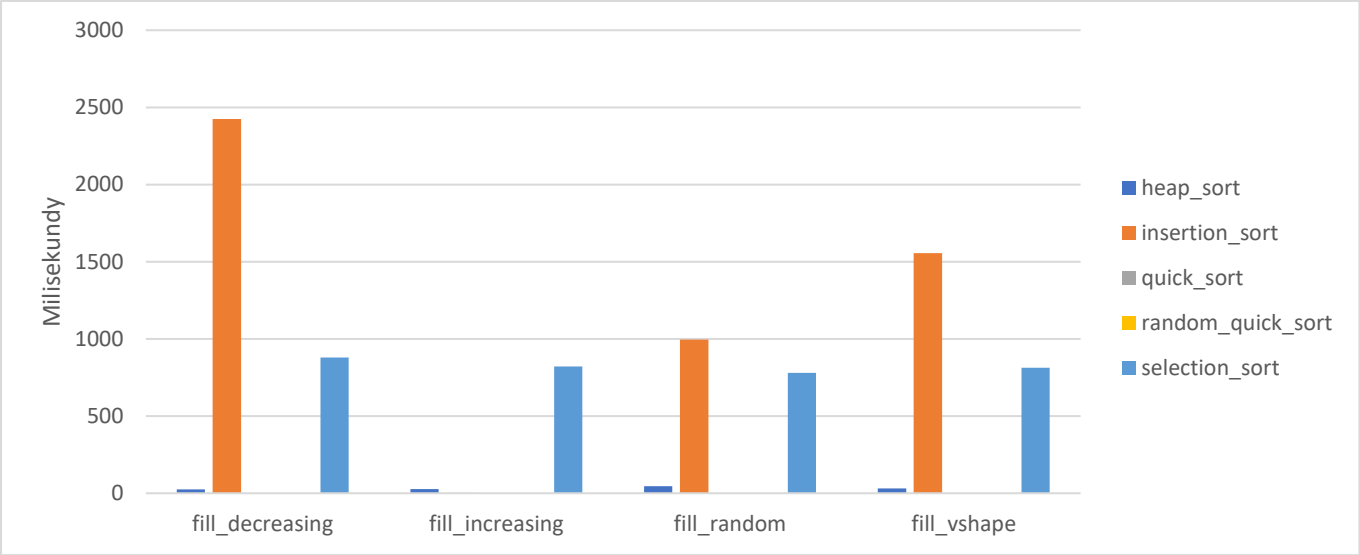
Dla 2000 elementów	heap_sort	insertion_sort	quick_sort	random_quick_sort	selection_sort
fill_decreasing	18,349	406,56	315,581	259,535	172,139
fill_increasing	14,948	0,657	568,282	386,225	176,778
fill_random	16,956	181,861	4,598	3,881	153,681
fill_vshape	19,104	181,26	223,834	184,213	180,759



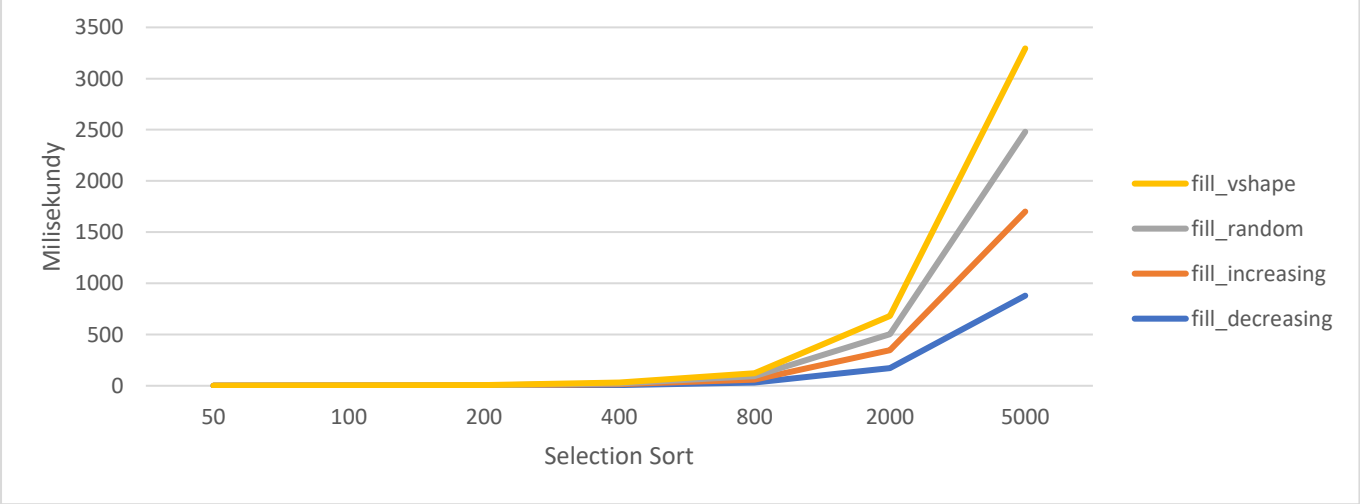
Tablica 5000-elementowa: (w milisekundach)

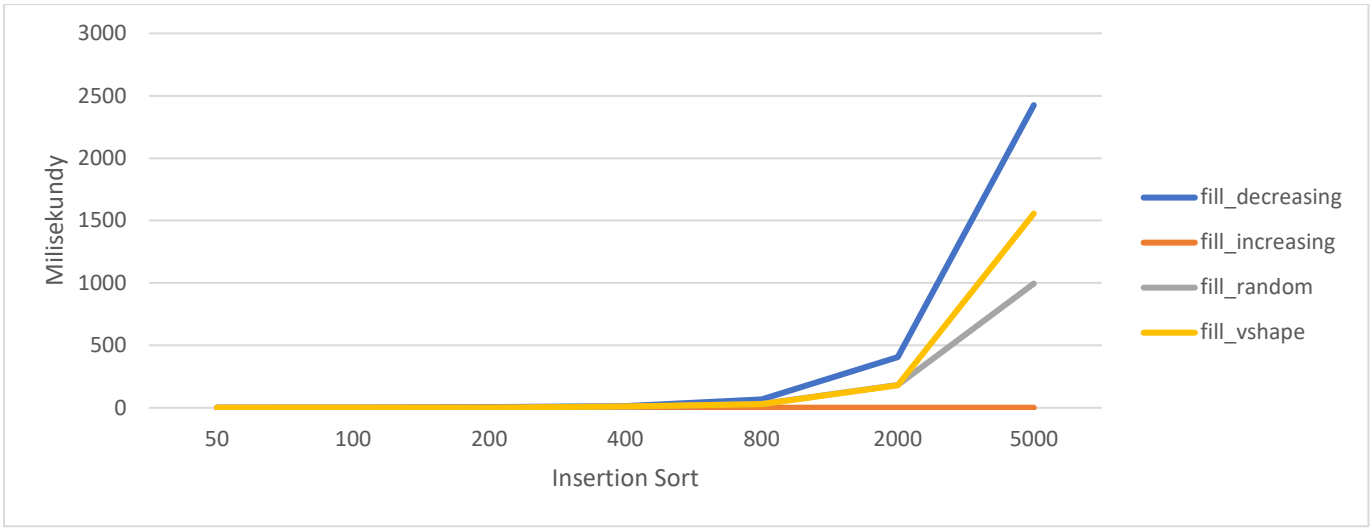
Dla 5000 elementów	heap_sort	insertion_sort	quick_sort	random_quick_sort	selection_sort
fill_decreasing	24,8214	2425,2806	N/A	N/A	879,8823
fill_increasing	27,6941	0,6663	N/A	N/A	821,4843
fill_random	44,7195	995,4923	N/A	N/A	779,1157
fill_vshape	30,5605	1556,55	N/A	N/A	813,9452

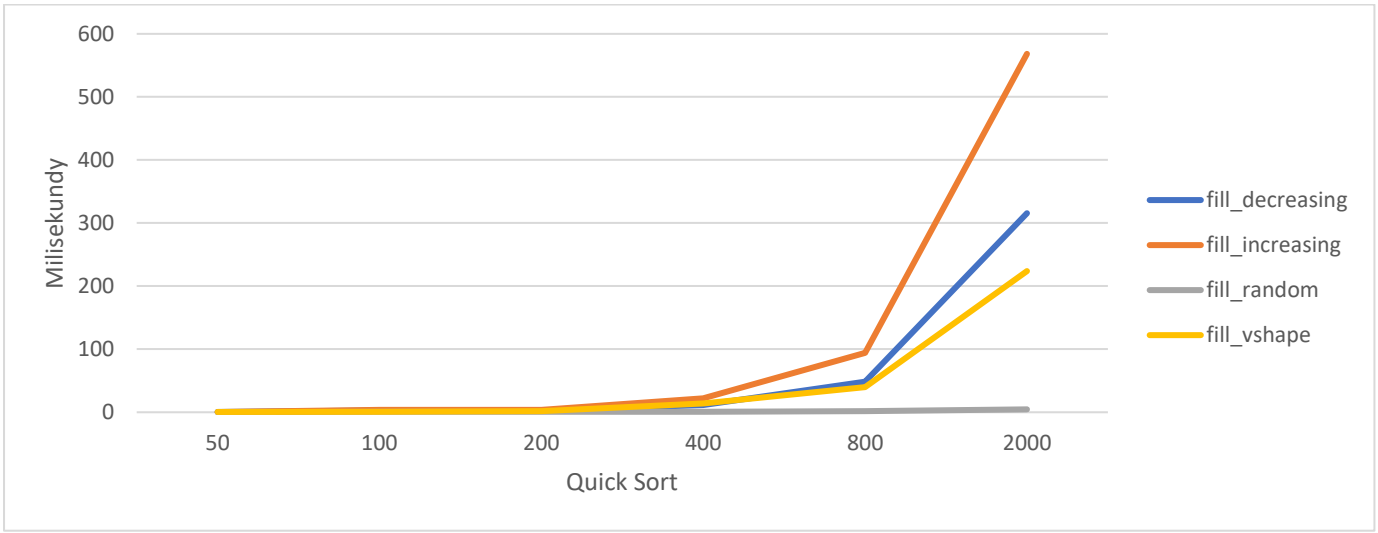
N/A: >10000 (Maximum recursion depth)

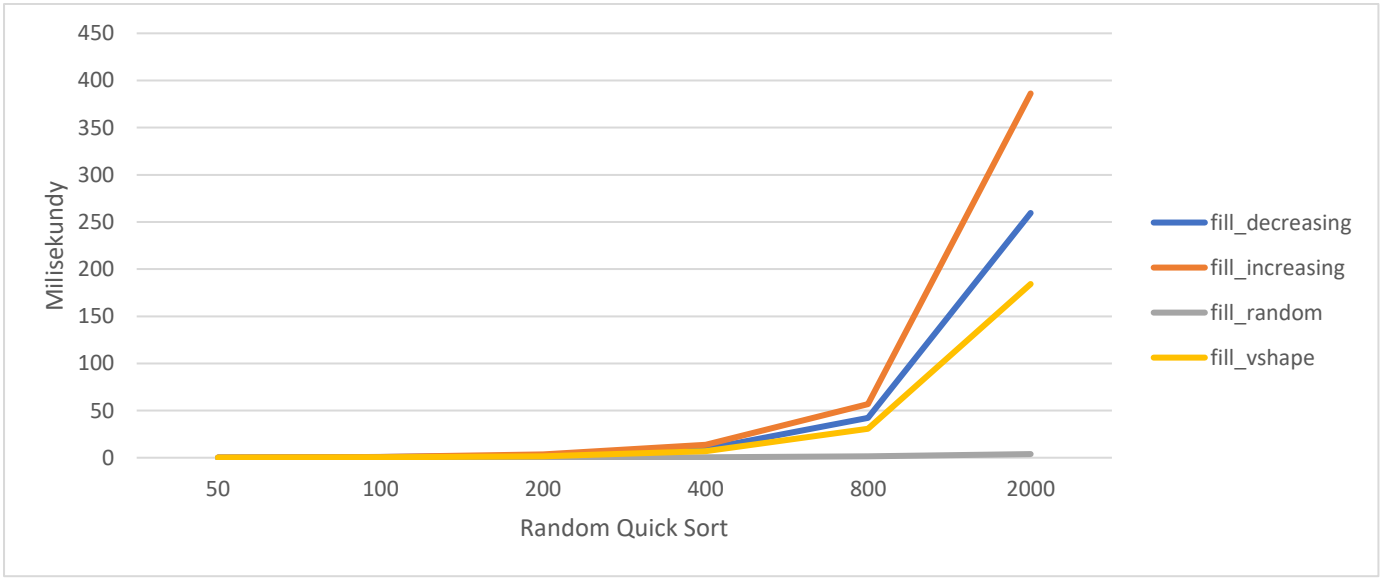


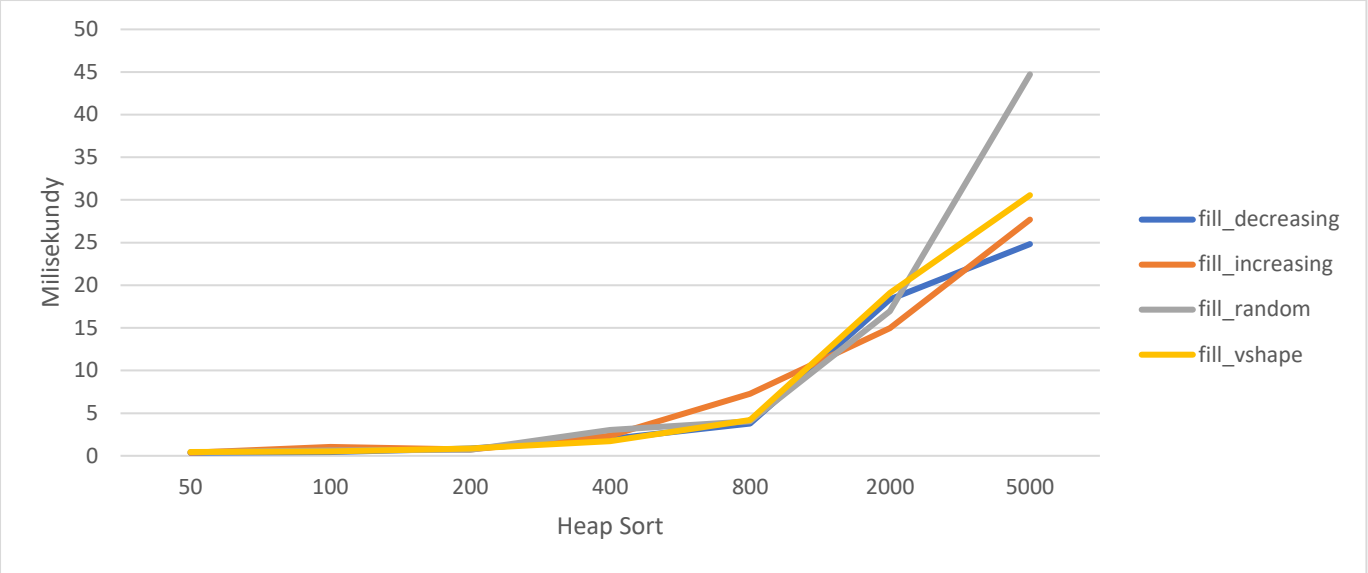
Wydajność ilościowa sortowań:




3000
2500
2000
1500
1000
500
0
Milisekundy
50
100
200
400
800
2000
5000
Insertion Sort
fill_decreasing
fill_increasing
fill_random
fill_vshape


600
500
400
300
200
100
0
Milisekundy
50
100
200
400
800
2000
Quick Sort
fill_decreasing
fill_increasing
fill_random
fill_vshape


450
400
350
300
250
200
150
100
50
0
Milisekundy
50
100
200
400
800
2000
Random Quick Sort
fill_decreasing
fill_increasing
fill_random
fill_vshape



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 sortowania	Krótko	Długo
Heap Sort	malejąco	rosnąco
Insertion Sort	rosnąco	malejąco
Quick Sort	losowo	rosnąco
Selection Sort	losowo	malejąco

Złożoność obliczeniowa dla najgorszych przypadków:

Rodzaj sortowania	Złożoność czasowa
Heap Sort	O (n log (n))
Insertion Sort	O (n^2)
Quick Sort	O (n^2)
Selection Sort	O (n^2)