Program porównuje wydajności dwóch algorytmów sortowania: mojego zaimplementowanego algorytmu sortowania Shella i wbudowanej funkcji sortującej z biblioteki Pythona. Analiza polegała na zmierzeniu czasu i wygenerowaniu losowych danych w różnych stopniach żeby porównanie było wiarygodne, na koniec przedstawię również wykresy.

Funkcja shell_sort(l), tak jak już pisałem w poprzedniej dokumentacji implementuje algorytm sortowania Shella. Oblicza rozmiar listy, inicjalizuje zmienne pomocnicze. Pętla, która stopniowo zmniejsza krok porównywania elementów listy, definiowany jako delta. Dzięki temu najpierw porównywane są elementy oddalone od siebie o większe odległości, a w kolejnych iteracjach krok maleje, co prowadzi do dokładniejszego porządkowania. Jeśli dwa elementy są w niewłaściwej kolejności, zamieniają się miejscami, a algorytm kontynuuje porządkowanie aż do momentu, gdy krok osiągnie wartość 1, a lista zostanie w pełni posortowana.

Wbudowana funkcja sortująca Pythona (sort()) została użyta jako punkt odniesienia. Jest to zoptymalizowany algorytm, który działa szybciej niż nasza implementacja Shella, dzięki bardziej zaawansowanej logice i optymalizacji pamięci. Porównanie obu algorytmów pozwala ocenić różnice w wydajności w różnych sytuacjach.

W celu przeprowadzenia analizy dane testowe zostały wygenerowane w pięciu różnych wariantach: całkowicie losowe, częściowo posortowane w 25%, częściowo posortowane w 50%, w pełni posortowane oraz posortowane w odwrotnej kolejności. Dla każdego wariantu zostały wygenerowane listy o trzech rozmiarach: 1000, 2000 i 5000 elementów. Obie funkcje sortujące testowano na tych samych danych, a czas ich wykonania został zmierzony z wykorzystaniem funkcji time.perf_counter.

Na podstawie wyników wygenerowane zostały wykresy, które pokazują czas sortowania w zależności od rozmiaru danych oraz stopnia ich posortowania. Na wykresach widać, że algorytm wbudowany w Pythonie działa znacząco szybciej we wszystkich przypadkach. Algorytm sortowania Shella, choć poprawnie realizuje swoje zadanie, wymaga więcej czasu, szczególnie przy większych listach lub bardziej losowych danych. Różnice te wynikają z większej złożoności obliczeniowej naszej implementacji oraz optymalizacji wbudowanej funkcji sort().

Wnioskiem z przeprowadzonej analizy jest to, że własnoręczne implementacje algorytmów są dobrą opcją do nauki, ale w praktyce raczej korzysta się z wbudowanych, gotowych funkcji dostępnych w bibliotekach. Wybrany shell, ze względów na to że jest ciekawy, niestety raczej nigdy nie dorówna zoptymalizowanej funkcji sortującej wbudowanej w Pythona

Wyniki programu:

Stopień: random, Rozmiar: 1000, Shell Sort: 0.003902, Wbudowane: 0.000065

Stopień: random, Rozmiar: 2000, Shell Sort: 0.010398, Wbudowane: 0.000134

Stopień: random, Rozmiar: 5000, Shell Sort: 0.032474, Wbudowane: 0.000370

Stopień: 25_sorted, Rozmiar: 1000, Shell Sort: 0.003687, Wbudowane: 0.000048

Stopień: 25_sorted, Rozmiar: 2000, Shell Sort: 0.009382, Wbudowane: 0.000104

Stopień: 25_sorted, Rozmiar: 5000, Shell Sort: 0.031945, Wbudowane: 0.000297

Stopień: 50_sorted, Rozmiar: 1000, Shell Sort: 0.003339, Wbudowane: 0.000038

Stopień: 50_sorted, Rozmiar: 2000, Shell Sort: 0.008038, Wbudowane: 0.000086

Stopień: 50_sorted, Rozmiar: 5000, Shell Sort: 0.029631, Wbudowane: 0.000219

Stopień: 100_sorted, Rozmiar: 1000, Shell Sort: 0.000490, Wbudowane: 0.000002

Stopień: 100_sorted, Rozmiar: 2000, Shell Sort: 0.001112, Wbudowane: 0.000004

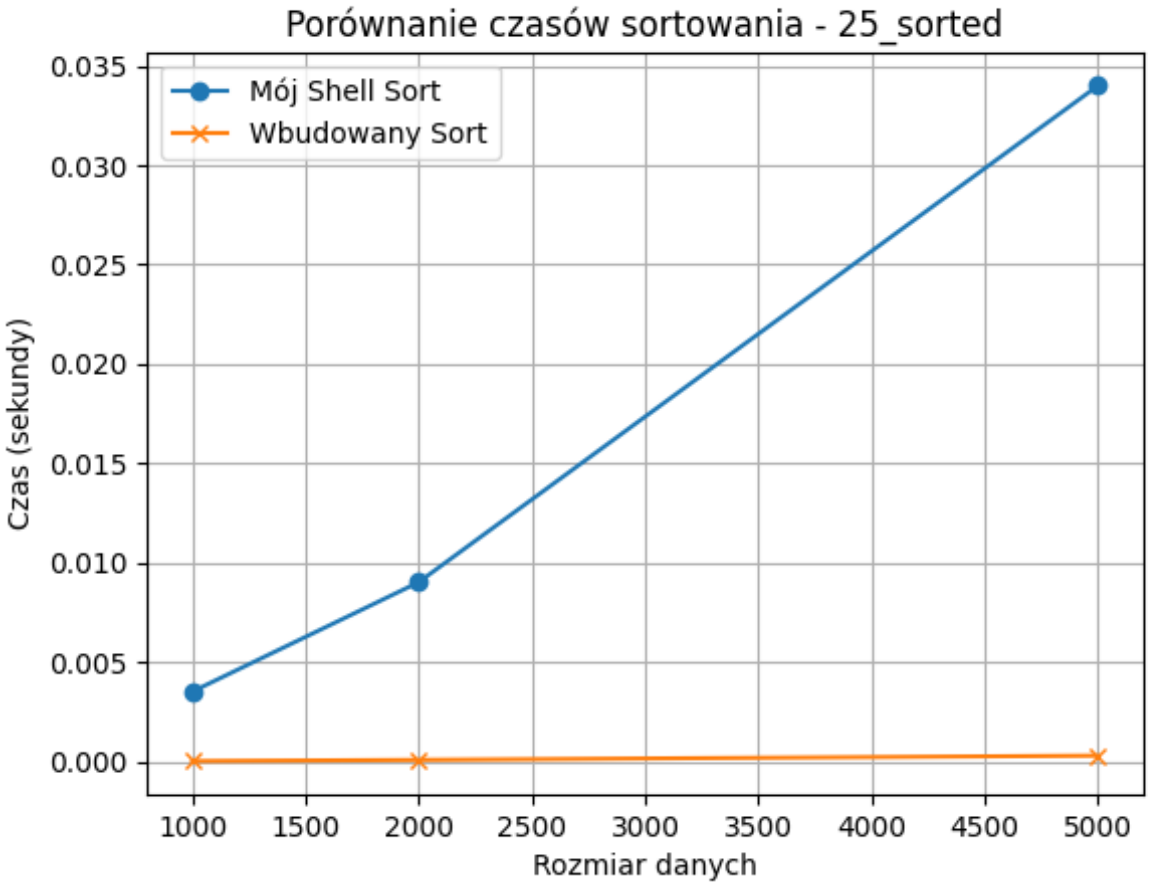
Stopień: 100_sorted, Rozmiar: 5000, Shell Sort: 0.003426, Wbudowane: 0.000009

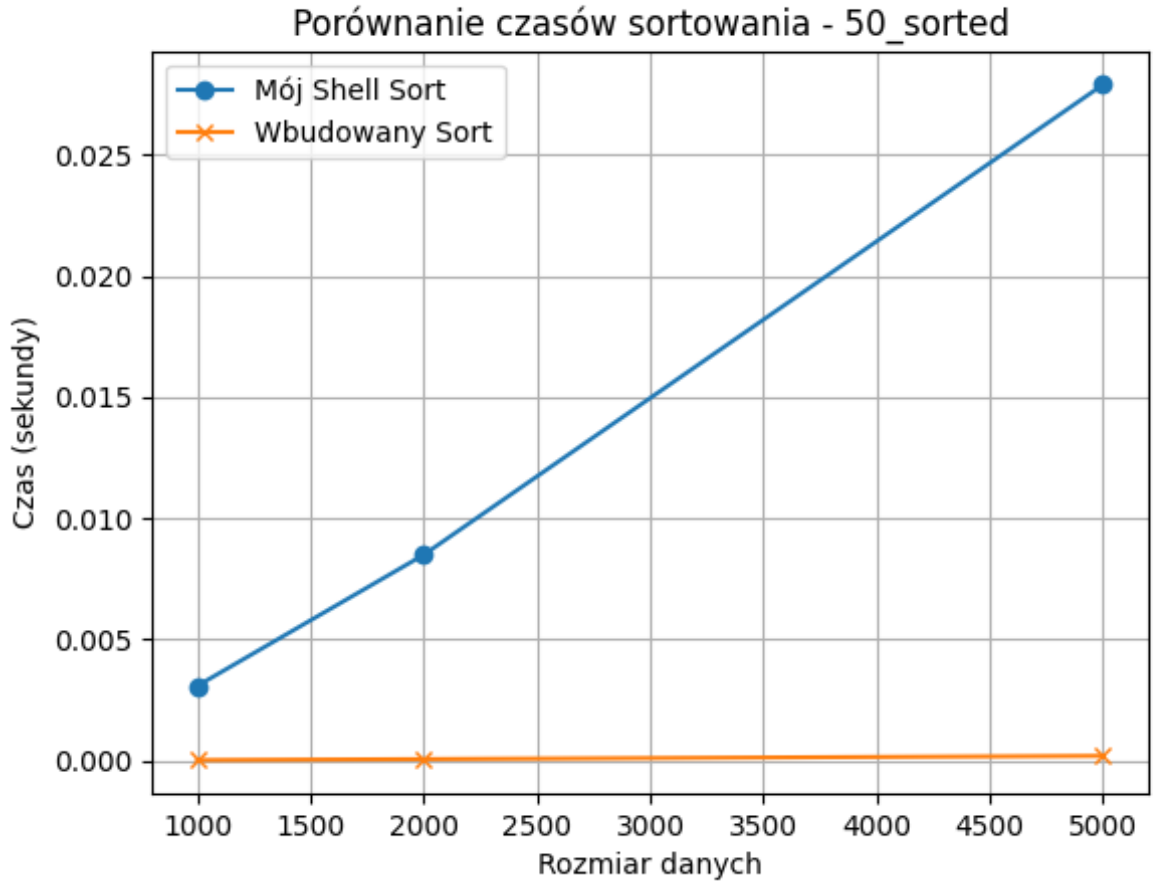
Stopień: reverse_sorted, Rozmiar: 1000, Shell Sort: 0.001309, Wbudowane: 0.000046

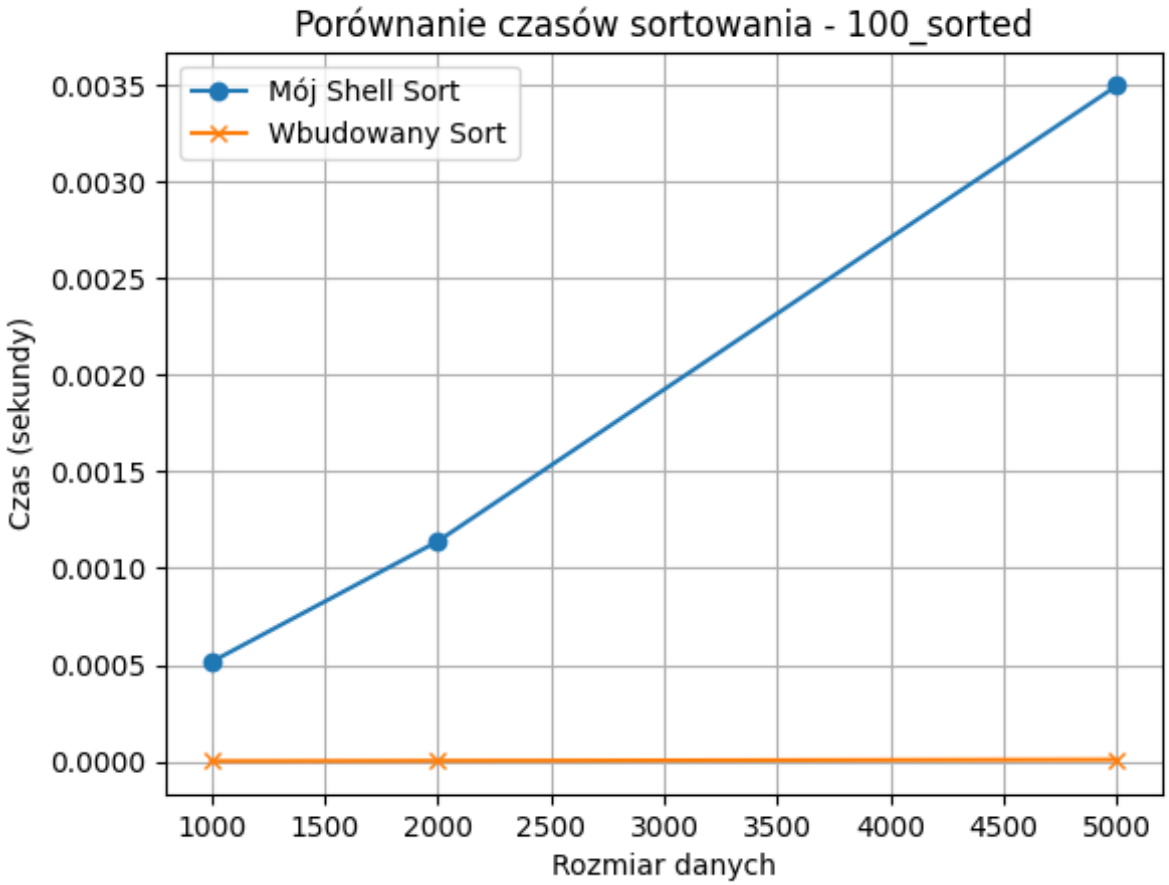
Stopień: reverse_sorted, Rozmiar: 2000, Shell Sort: 0.002852, Wbudowane: 0.000081

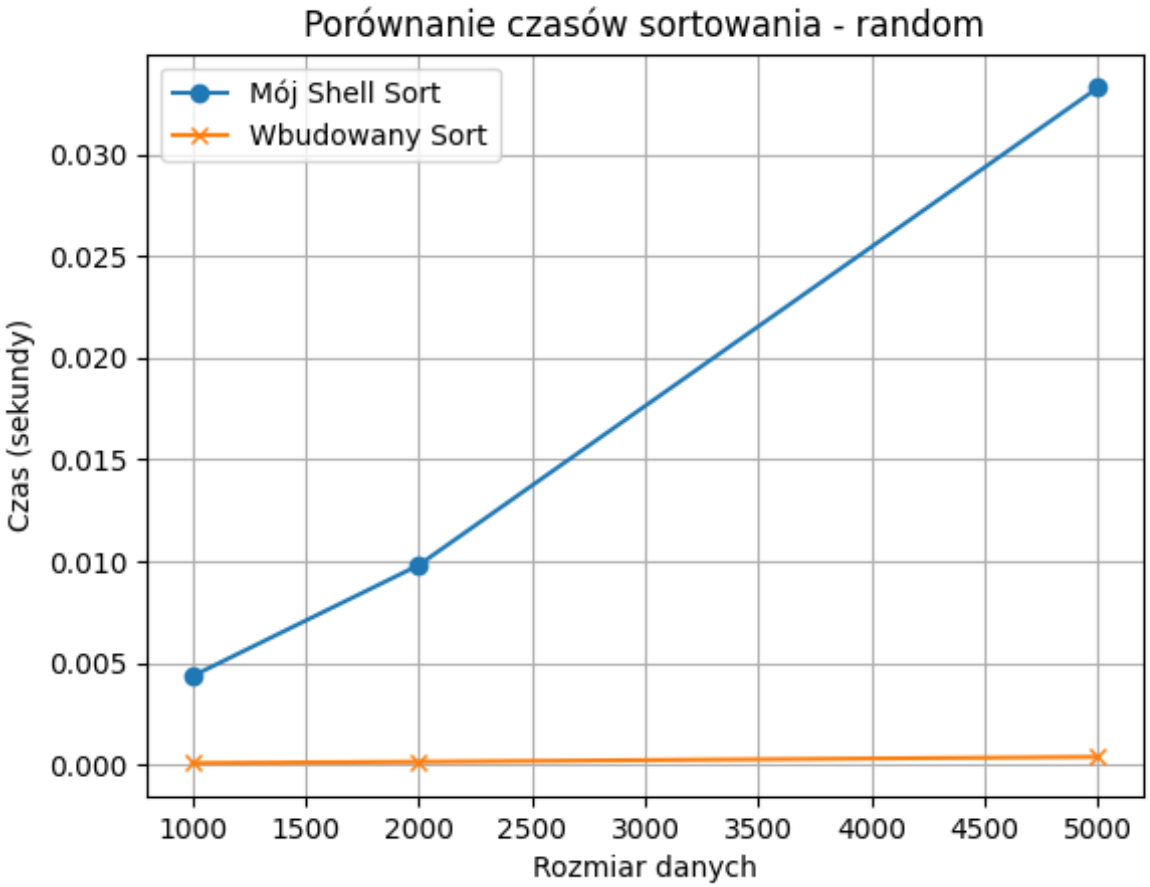
Stopień: reverse_sorted, Rozmiar: 5000, Shell Sort: 0.008249, Wbudowane: 0.000188

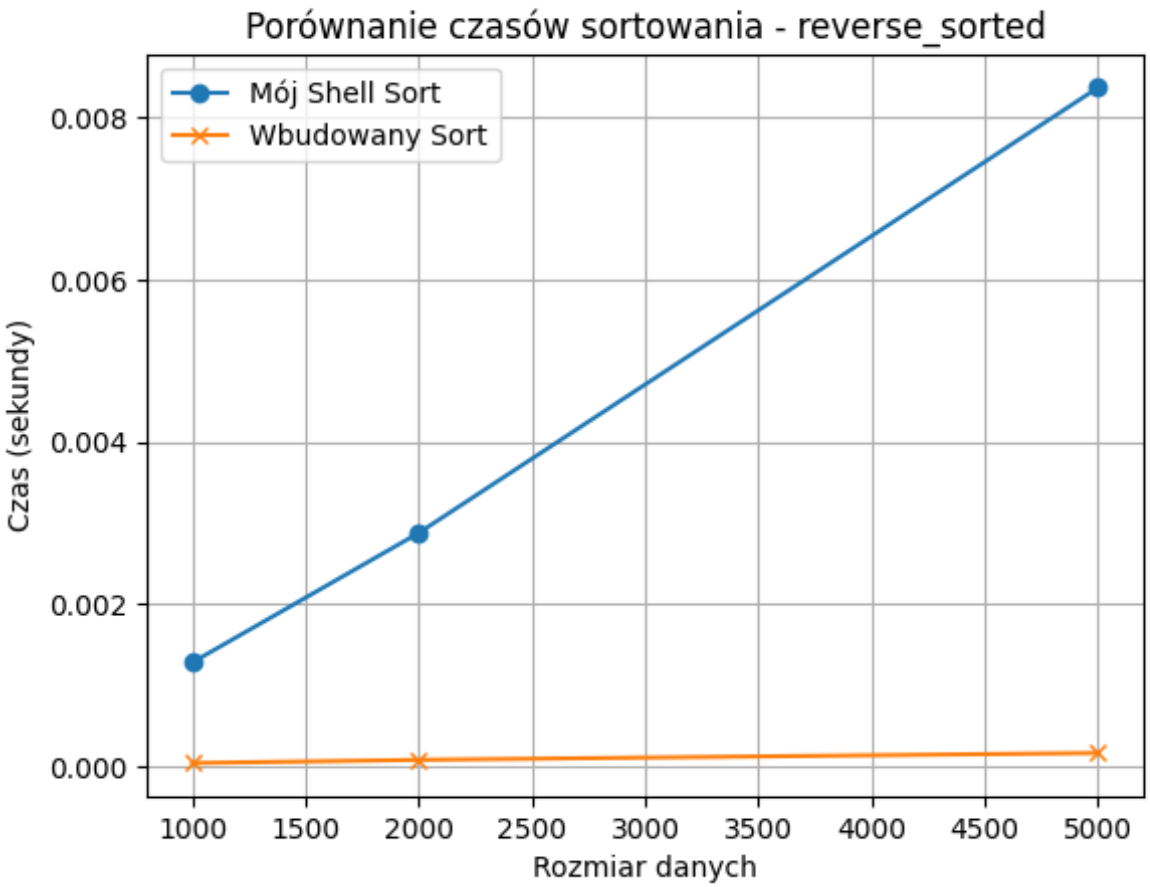
Dołączam pare wykresów (zrobionych w matplotlibie):




Porównanie czasów sortowania - 50_sorted
Mój Shell Sort
Wbudowany Sort
Czas (sekundy)
0.025
0.020
0.015
0.010
0.005
0.000
1000
1500
2000
2500
3000
3500
4000
4500
5000
Rozmiar danych


Porównanie czasów sortowania - 100_sorted
Mój Shell Sort
Wbudowany Sort
Czas (sekundy)
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
1000
1500
2000
2500
3000
3500
4000
4500
5000
Rozmiar danych


Porównanie czasów sortowania - random
Mój Shell Sort
Wbudowany Sort
Czas (sekundy)
0.030
0.025
0.020
0.015
0.010
0.005
0.000
1000
1500
2000
2500
3000
3500
4000
4500
5000
Rozmiar danych


Porównanie czasów sortowania - reverse_sorted
Mój Shell Sort
Wbudowany Sort
Czas (sekundy)
0.008
0.006
0.004
0.002
0.000
1000
1500
2000
2500
3000
3500
4000
4500
5000
Rozmiar danych